

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

Dissertação de Mestrado

*Descoberta de Relacionamentos entre Padrões de
Software utilizando Semântica Latente*

Rute Nogueira Silveira de Castro

Fortaleza – CE, 2006

RUTE NOGUEIRA SILVEIRA DE CASTRO

*Descoberta de Relacionamentos entre Padrões de
Software utilizando Semântica Latente*

Dissertação apresentada ao Mestrado em
Ciência da Computação da UFC, como
requisito para a obtenção parcial do grau
de MESTRE em Ciência da Computação

Fortaleza - CE, 2006

RUTE NOGUEIRA SILVEIRA DE CASTRO

Descoberta de Relacionamentos entre Padrões de Software utilizando Semântica Latente

Dissertação apresentada ao Mestrado em Ciência da Computação da UFC, como requisito para a obtenção parcial do grau de MESTRE em Ciência da Computação

BANCA EXAMINADORA

Rossana Maria de Castro Andrade, Phd.

Jerffeson Teixeira de Souza, Phd.

Maria del Rosario Girardi, Phd.

Javam de Castro Machado, Dsc.

"Do or do not. There is no try."

Master Yoda

Agradecimentos

A Deus, por ter me ajudado em mais uma etapa da minha vida.

À Prof^a Rossana Andrade, pela segura orientação e pelo incentivo constante.

Aos meus pais e marido, pela dedicação e compreensão em todos os momentos.

Ao Prof Jerffeson Souza, pelo apoio na construção das idéias.

Aos amigos e colegas da UFC, em especial aos integrantes do GREaT, pelo companheirismo.

Aos amigos Rafael Carmo, Aelinton, Joel Uchoa e Leonardo da UECE, que foram importantes ajudas durante a fase de implementação.

Ao Prof Javam Machado e à Prof^a Rosario Girardi por gentilmente comporem a banca.

Aos professores do Mestrado em Ciência da Computação da UFC, pelos valiosos ensinamentos.

A todos aqueles que, direta ou indiretamente, contribuíram para a elaboração desta dissertação.

Resumo

O reuso de padrões de *software* vem se tornando cada vez mais comum no desenvolvimento de sistemas, pois se trata de uma boa prática de engenharia de software que visa promover a reutilização de soluções comprovadas para problemas recorrentes. No entanto, existe uma carência de mecanismos que promovam a busca de padrões adequados a cada situação. Também há uma dificuldade na detecção de relacionamentos existentes entre os padrões de software disponíveis na literatura.

Este trabalho apresenta o uso de técnicas de mineração de texto em um conjunto de padrões de software com o objetivo de identificar como esses padrões se relacionam. A técnica de mineração de textos busca extrair conceitos inteligentes a partir de grandes volumes de informação textual.

O padrão de software deve ser tratado dentro de mineração de texto como um grande volume de texto com uma estrutura definida por seu *template*. Os graus de relacionamentos entre os padrões são determinados nos possíveis tipos de relacionamentos entre eles, bem como através de regras fundamentadas no conceito de Padrões de Software. Essas regras, aliadas à técnica de mineração de texto, geram as informações de relacionamento desejadas.

Palavras-chave: Padrões de Software, Mineração de Texto, Repositório de Padrões, Análise de Semântica Latente.

Sumário

1. Introdução	12
1.1 Caracterização do Problema e Motivação	12
1.2 Objetivos, Metodologia e Resultados esperados.....	16
1.3 Organização da Dissertação	16
2. Padrões de Software	18
2.1 Introdução	18
2.2 Histórico	19
2.3 Classificação de Padrões	20
2.3 Elementos de um padrão	24
2.4 Relacionamentos entre padrões de software	27
2.5 Agrupamentos de Padrões	29
2.5.1 Coleção de padrões	30
2.5.2 Catálogo de padrões	30
2.5.3 Linguagem de padrões	33
2.6 Repositório de Padrões de Software	34
2.7 Conclusão	36
3. Análise da Semântica Latente	37
3.1 Introdução	37
3.2 Mineração de Dados	38
3.2.1 Regras de Associação	41
3.2.2 Classificação	42
3.2.3 Agrupamento	43
3.2.4 CRISP-DM	44
3.3 Mineração de Texto	46
3.3.1 Análise das Ferramentas de Mineração de Texto	48
3.4 Análise da Semântica Latente	51
3.5 Conclusão	54
4. Descoberta de Relacionamentos entre Padrões de Software	55
4.1 Introdução	55
4.2 Metodologia	55
4.2.1 Construção do espaço semântico	56
4.2.2 Decomposição de Valor Singular (SVD)	56
4.2.3 Detecção de Relacionamentos entre Padrões	58
4.2.4 Correlação texto a texto	59
4.2.5 Regras de associação de relacionamento	59
4.2.6 Cálculo do grau de relacionamento	61
4.3 Conclusão	63
5. Conclusão.....	64
5.1 Resultados	64
5.2 Trabalhos Futuros	65

6. Referências Bibliográficas.....	66
APENDICE A - Implementação LSA.....	73

Lista de Figuras

<i>Figura 01 - Classificação dos Relacionamentos entre Padrões do GoF</i>	<i>27</i>
<i>Figura 02 - Catálogo de Padrões do GoF</i>	<i>32</i>
<i>Figura 03 - MoRaR - Uma Linguagem de Padrões para o Gerenciamento de Mobilidade e de Recursos de Rádio</i>	<i>33</i>

Lista de Abreviaturas e Siglas

GOF	Gang of Four
GUI	Graphic User Interface
HTML	Hypertext Markup Language
J2EE	Java 2 Enterprise Edition
JSP	Java Server Pages
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
MD	Mineração de Dados
MORAR	Mobility Radio Resources
MT	Mineração de Texto
MVC	Model View Controller
PLOP	Patterns Languages of Programming
POSA	Pattern-Oriented Software Architecture
SVD	Singular Value Decomposition
XML	Extensible Markup Language

Capítulo 1

Introdução

Este capítulo apresenta na Seção 1.1 uma visão geral sobre reuso de padrões de software e a caracterização do problema que conduz à motivação do trabalho. Na Seção 1.2, são apresentados os objetivos principais da dissertação e, na seção 1.3, é definida a organização da dissertação.

1.1 Caracterização do Problema e Motivação

A literatura propõe métodos a serem utilizados nas mais diversas etapas de desenvolvimento de software com o objetivo de garantir os requisitos de qualidade do software, por exemplo em [NBR19001][ISO9000-3][ISO9126-1] [ISO9126-2] e [ISO9126-3]. Para aperfeiçoar a qualidade do software pode-se, a princípio, definir quais dos seus atributos devem ser medidos para que se possa identificar um grau de qualidade. Alguns dos critérios mencionados por diversos autores são correção, facilidade de uso, reusabilidade, legibilidade, eficiência e portabilidade [ISO9126-2][PRE95]. Por um lado, critérios como correção, facilidade de uso, eficiência e portabilidade podem ser medidos pelos usuários. Por outro lado, critérios como reusabilidade e legibilidade, são medidos pela equipe responsável pelo desenvolvimento e manutenção do software. De acordo com [TSU97], a qualidade de software também é determinada pela qualidade dos processos utilizados para o desenvolvimento. Deste modo, a melhoria da qualidade de software é obtida pela melhoria da qualidade desses processos.

Nesse contexto, os padrões de software aparecem como uma estratégia para representar boas soluções para problemas freqüentes, possibilitando atingir uma melhor qualidade no desenvolvimento de software. O reuso de padrões de software vêm sendo cada vez mais difundido entre analistas e desenvolvedores de sistemas, reforçando a relevância desta área de pesquisa dentro da engenharia de software.

De acordo com [SAN04], os padrões podem ser utilizados individualmente (para resolver problemas isolados ou específicos) ou em conjunto (através do uso de coleções de padrões, por exemplo, para resolver problemas complexos ou para documentar e gerar arquiteturas de software). A utilização de padrões pode ser feita das seguintes maneiras:

- manualmente – onde o desenvolvedor é responsável por modelar ou codificar o padrão por conta própria, a partir de sua documentação ou através do uso de *frameworks* [AND01] [JON88];
- automaticamente – através do uso de ferramentas de modelagem, *refactoring* ou geração de código com suporte a padrões.

Um desenvolvedor que utiliza a aplicação de padrões de forma manual pode encontrar dificuldades em localizar os padrões desejados e, por conseguinte, padrões relacionados aos que estão sendo procurados. Além do mais, a tarefa de compreender a solução do padrão para aplicá-lo pode se tornar complexa, à medida que outros problemas decorrentes da utilização do padrão vão surgindo.

No caso da utilização automática de padrões, como mencionado anteriormente, existem ferramentas de apoio ao desenvolvimento de software que automatizam o processo de aplicação de padrões de software, auxiliando na modelagem de sistemas, geração e reestruturação de código [KRU99][BER02][TOM04]. No entanto, os padrões aplicados automaticamente através da utilização dessas ferramentas são geralmente restritos a um conjunto de padrões fornecido pelos fabricantes das ferramentas, que na maioria das vezes não é extensível, não suportando a inclusão de novos padrões de acordo com as necessidades dos desenvolvedores.

Uma solução para esses problemas é o armazenamento de padrões de software em um Repositório de Padrões, como proposto em [AND03] facilitando, assim, o processo de busca por padrões para que o desenvolvedor de software possa concentrar esforços em aprender a utilizar as soluções dos padrões e adequá-las aos seus problemas.

Um outro problema comumente encontrado é o fato dos modelos de padrões serem disponibilizados em formatos proprietários ou formatos específicos para uma determinada ferramenta, o que também limita as opções de escolha e reuso dos padrões pelos desenvolvedores. Em [SAN04], foi desenvolvida uma ferramenta que busca integrar diversos modelos de Padrões, em um formato não proprietário, a uma ferramenta de modelagem de Software, no caso, o Rational Rose.

É importante ressaltar ainda que os mecanismos manual e automático para armazenamento e busca de padrões de software [POR06][DIE06] não suportam os diversos formatos de padrões que existem atualmente, o que pode dificultar o processo de busca e tornar inapropriado o armazenamento do padrão, quando o seu formato original não é preservado.

Como os padrões de software disponíveis na literatura não estão isolados, alguns padrões relacionados aparecem no campo do *template* dos padrões *Related Patterns* (Padrões Relacionados). Além disso, algumas linguagens e catálogos de padrões já possuem representações gráficas dos relacionamentos em estruturas conhecidas como mapas. Entretanto, outros relacionamentos não são tão aparentes e necessitam ser explorados.

Um Repositório de Padrões de Software, como os propostos em [AND03][MIC05] e [PAT05], mostra-se como uma solução para alguns dos problemas citados anteriormente. Dessa forma, torna-se importante para facilitar o reuso de padrões. Para isso, existe a necessidade de armazenar esses padrões em um formato apropriado e disponibilizá-los de forma amigável e rápida para sua recuperação e reuso. É importante facilitar o processo de busca por padrões para que o desenvolvedor de software possa concentrar esforços em aprender a utilizar as soluções dos padrões e adequá-las aos seus problemas.

Por exemplo, uma funcionalidade interessante para um Repositório de padrões seria a possibilidade de detectar os graus de relacionamento entre os padrões automaticamente, resolvendo dessa maneira, outro problema detectado na reutilização de padrões. Através dessa funcionalidade, um desenvolvedor de

software detectaria as possíveis alternativas à solução proposta pelo sistema (padrão retornado da busca) ou, ainda, complementos à solução previamente proposta através de graus de relacionamentos dos padrões apresentados com quaisquer outros dentro de um Repositório.

No entanto, ainda é complicado determinar um grau de relacionamento entre padrões provenientes de agrupamentos diferentes. Isso decorre da dificuldade que os autores encontram em associar o conteúdo de um padrão a outro, já que os mesmos se encontram espalhados na literatura e em sites especializados de maneira dissociada entre si. Dessa forma, a não ser que os padrões estejam dentro de um mesmo agrupamento (linguagem ou catálogo), os graus de relacionamentos entre os padrões não são facilmente detectáveis.

1.2 Objetivos, Metodologia e Resultados Esperados

Este trabalho propõe o uso de técnicas de Mineração de Dados (MD) para a busca de relacionamentos de padrões dentro de um repositório de padrões. Através dessas técnicas, a extração de informações inteligentes de textos se torna possível, como acontece na Mineração de Textos (MT).

De forma geral, a MT busca extrair conceitos a partir de grandes volumes de informação textual. No caso dos padrões de software, o conteúdo dos mesmos é tratado como um grande volume textual com uma estrutura definida. A estrutura definida do padrão é formada pelos campos do seu *template*, os quais possuem propriedades específicas em relação ao seu conteúdo, determinando, assim, a formação de um padrão.

A técnica de mineração de textos, portanto, é utilizada nesta dissertação para investigar os graus de relacionamentos entre os padrões através de toda a fundamentação da área de Mineração de Dados detalhada no Capítulo 3 e aliada às regras baseadas nos possíveis tipos de relacionamentos bem como no conceito de Padrões de Software, que serão detalhados no Capítulo 2.

Como resultados esperados, têm-se os níveis de relacionamentos entre os padrões de software detectados de maneira automática.

1.3 Organização da Dissertação

Além do capítulo 1 aqui apresentado, esta dissertação está dividida em quatro capítulos e um anexo cujo conteúdo é resumido a seguir:

Capítulo 2: Padrões de Software – Este capítulo apresenta conceitos que envolvem padrões de software, aspectos importantes sobre critérios de classificação de padrões e sobre relacionamentos entre padrões, bem como a dificuldade encontrada na detecção automática desses relacionamentos, que é a principal motivação deste trabalho.

Capítulo 3: Semântica Latente – Este capítulo apresenta técnicas e teorias a respeito das áreas de Mineração de Dados e de Textos, motivando a sua importância para este trabalho e demonstrando a utilidade das mesmas para os padrões de software através da Análise da Semântica Latente, que é uma técnica de análise textual (MT). Também é apresentada a teoria a respeito dessa Análise Semântica, seu histórico e seu funcionamento.

Capítulo 4: Descoberta de Relacionamentos entre Padrões de Software – Este capítulo apresenta o processo e a metodologia desenvolvida para a utilização das técnicas de Análise da Semântica Latente no processo de detecção de relacionamento entre os padrões.

Capítulo 5: Conclusão – Este capítulo apresenta os resultados alcançados ao longo do desenvolvimento desta dissertação e as possibilidades para trabalhos futuros.

Apêndice A: Implementação LSA – Este apêndice exemplifica a Implementação LSA para definir graus de relacionamento aos padrões de software. É importante ressaltar algumas particularidades da notação utilizada na escrita desta dissertação que visam facilitar a leitura.

- Os termos em *itálico* são utilizados para:
 - representar palavras ou expressões em língua estrangeira, com exceção de nomes de ferramentas ou produtos;
 - citar definições de outros autores (também delimitadas por “aspas”);
- Os termos em **negrito** são utilizados para representar palavras ou expressões que estão sendo definidas ou que merecem algum destaque especial no texto. Podem vir acompanhados também de *itálico* para um maior destaque;

Além disso, alguns termos que aparecem na forma de itens ou com marcadores podem receber alguma forma de destaque com **negrito** e/ou *itálico*.

Capítulo 2

Padrões de Software

Este capítulo destina-se a dar uma visão atual sobre padrões de software, apresentando os conceitos de padrões de software mais relevantes para o desenvolvimento desta dissertação. Na Seção 2.1 são demonstradas algumas definições e considerações importantes sobre padrões de software e na seção 2.2 um breve histórico sobre os mesmos. Na Seção 2.3 são identificados os componentes que definem os formatos de padrões. Em 2.4 são definidos conceitos sobre relacionamentos entre padrões de software. A Seção 2.5 traz uma visão geral sobre agrupamentos de padrões. A seção 2.6 traz uma visão geral de como os mecanismos para armazenamento e busca de padrões vêm sendo apresentados na literatura e introduz o Repositório de Padrões. A seção 2.7 apresenta uma conclusão sobre o capítulo.

2.1 Introdução

Quando um desenvolvedor de software está projetando uma aplicação, muitas vezes, ele se depara com um problema que já havia surgido em um software anterior, e, então, ele passa a reusar a essência da solução que utilizou nesta ocasião intuitivamente. Essa prática é também difundida em diversas áreas, como na arquitetura, e se tornou um comportamento inteligente de reaproveitamento de conhecimento.

Padrões de software têm sido pesquisados na última década como uma forma promissora de reuso, não somente de código mas também de projeto, análise, arquitetura e processo de desenvolvimento. Por intermédio dos padrões podem ser documentadas soluções para diferentes tipos de problemas que ocorrem ao longo do processo de desenvolvimento de software. Essas soluções, quaisquer

que sejam seus níveis de abstração, podem ser reusadas por outros desenvolvedores ao confrontarem-se com problemas similares.

É através dessa experiência prática que os especialistas chegam a criação de um padrão e, então, passam a utilizá-lo para solucionar o problema com eficiência e elegância.

2.2 Histórico

A origem documentada do conceito de padrões (i.e., patterns) deu-se com o trabalho feito pelo arquiteto Christopher Alexander no final dos anos 70 através da escrita dos livros “A Pattern Language” [ALE77] e “A Timeless Way of Building” [ALE79] que, além de exemplificarem, descrevem seu método para documentação de padrões específicos para a arquitetura. O trabalho de Alexander é voltado para a arquitetura, no entanto, é possuidor de uma fundamentação básica que pode ser abstraída para a área de software.

Foi em uma conferência sobre programação orientada a objetos, (OOPSLA) de 1987, que o assunto “Padrões de Software” ganhou maior repercussão na Engenharia de Software. Beck, também autor de [BEC97], dissertou a respeito de uma linguagem de padrões para projetar janelas em Smalltalk. A partir de então, muitos artigos, revistas e livros têm aparecido abordando o conceito de padrões de software, que descrevem soluções para problemas que ocorrem freqüentemente no desenvolvimento de software e que podem ser reusadas por outros desenvolvedores.

Alguns trabalhos pioneiros sobre o uso de padrões aconteceram na década de 90 com o livro de Gamma [GAM95], publicando vinte e três padrões de projeto, e o trabalho de Coplien [COP96][COP00], definindo “idiomas”, que são padrões de programação específicos para a linguagem C++.

Definição

Um padrão descreve uma solução para um problema que ocorre com freqüência durante o desenvolvimento de software, podendo ser considerado como um par “problema/solução” [BUS96].

Um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem sucedidas para um problema repetido que ocorre sob um determinado contexto [APP00].

Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de sistemas orientados a objetos. O seu uso proporciona um vocabulário comum para a comunicação entre projetistas de um mesmo trabalho, criando abstrações num nível superior ao de classes e garantindo uniformidade na estrutura do software. Além do mais, eles podem atuar como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos [GAM95].

Diversos padrões têm sido propostos, seja para solucionar problemas de programação [COP96], de projeto [GAM95][PRE95] e de análise [COA95][FOW97]. Dessa forma, pode-se dizer que existem padrões arquitetônicos, em que o nível de abstração é bastante alto, padrões de análise, padrões de projeto, padrões de implementação, entre outros. As diversas classificações de padrões serão melhor detalhadas na próxima seção.

Conferências internacionais para apresentação e discussão de novos padrões e linguagens de padrões têm ocorrido desde 1994. Uma série de livros reúne os melhores artigos apresentados nessas conferências [COP96][VLI97][MAR98], nos quais podem ser encontrados padrões das mais variadas classes.

2.2 Classificação de Padrões

Conforme mencionado no início deste capítulo, padrões de software abrangem diferentes níveis de abstração e podem ser classificados em categorias de modo a facilitar sua recuperação e uso. Segundo Andrade [AND01], uma classificação

bastante usada para padrões de software é a que classifica os padrões de acordo com o estágio de desenvolvimento de software em que o padrão é aplicado. Desta forma, os padrões podem pertencer às seguintes categorias: padrões de requisitos, padrões de análise, padrões de projeto, padrões de implementação (também chamados de Idiomas) e padrões de testes [RIS00].

Padrões arquitetônicos e meta-padrões também são considerados padrões de projeto, apesar de constituírem duas categorias de padrões. Padrões arquitetônicos são os padrões que ajudam a especificar elementos de arquiteturas de sistemas e seus relacionamentos. Já os meta-padrões [PRE95] trabalham em um nível mais alto de abstração que os padrões de projeto, e podem especificar como as soluções adotadas por padrões de projeto se adequam a situações mais gerais através da observação dos seus aspectos comuns.

Além destes, foram aparecendo publicações de padrões voltados para as fases iniciais de desenvolvimento, padrões de níveis mais altos de abstração, como os padrões de análise, apresentados nos trabalhos de Fowler [FOW97], padrões de requisitos [KON02] e padrões para implementação.

Uma modalidade diferente é a dos anti-padrões que, de acordo com [APP00], representam uma “lição aprendida”, ao contrário dos padrões, que representam a “melhor prática”. Os anti-padrões podem descrever uma solução inapropriada para um problema que resultou em uma situação ruim ou descrever como escapar de uma má situação e como proceder para, a partir dela, atingir uma boa solução. Os anti-padrões tornam-se interessantes para que se perceba e entenda tanto soluções boas quanto não muito boas.

Uma outra definição para anti-padrões [CMG98] diz que um anti-padrão não é nada além de um padrão que diz como ir de um problema para uma solução ruim ou um padrão que diz como ir de uma solução ruim para uma solução boa.

Outro critério muito utilizado é a classificação dos padrões pelo seu domínio de aplicação específico. Estes são muito procurados por desenvolvedores com pouca experiência em padrões ou que não estão familiarizados com os tipos de padrões existentes, buscando, dessa forma, padrões pela natureza da aplicação. Por isso,

é normal encontramos padrões na literatura descritos em categorias como: padrões para Web, GUI, XML, Segurança de Sistemas, Redes, Banco de Dados, Sistemas Distribuídos, entre outras.

O Almanaque de Padrões 2000 [RIS00] possui um complexo índice de padrões por categorias, que consistem tanto de categorias que classificam os padrões segundo as categorias utilizadas pela Gang of Four [GAM95], quanto sobre o estágio de desenvolvimento da aplicação do padrão e até mesmo conforme o domínio da aplicação.

Um critério mais específico de classificação utiliza como base a camada de aplicação onde o padrão deve ser aplicado. O catálogo de padrões, Core J2EE [COR05] da Sun Microsystems traz um conjunto de padrões dividido em três categorias referentes às camadas de aplicação: Apresentação, Negócios e Integração. O catálogo de padrões IBM patterns for e-business [IBM05] também utiliza esse critério para a classificação de padrões.

Em [BUS96] foi publicado um catálogo de padrões para arquitetura de software (POSA – Pattern-Oriented Software Architecture). O livro apresenta os padrões agrupados em três categorias e os problemas em dez categorias, utilizando estes dois critérios para o esquema de classificação. A seguir são apresentadas as definições das categorias utilizadas pelos autores:

- Padrões Arquitetônicos: padrões que expressam um esquema de organização estrutural fundamental para sistemas de software. Fornecem um conjunto de subsistemas pré-definidos, especifica suas responsabilidades e incluem regras e diretrizes para organizar os relacionamentos entre eles.
- Padrões de Projeto: padrões que fornecem um esquema para refinar subsistemas, componentes de um sistema de software ou os relacionamentos entre eles. Descrevem uma estrutura comumente recorrente de componentes comunicantes que resolvem um problema geral de projeto em um contexto particular.
- Idiomas: padrões de baixo nível, específicos para uma linguagem de programação. Um idioma descreve como implementar aspectos particulares de

componentes ou os relacionamentos entre eles usando as características da linguagem.

Os autores da Gang of Four [GAM95] propõem um sistema de classificação baseado em dois critérios: quanto ao escopo e quanto ao propósito do padrão.

O critério escopo especifica se o padrão trabalha com:

- Escopo classe: padrão com foco em classes, subclasses e seus relacionamentos,
- Escopo objeto: padrão que trabalha com relacionamentos entre objetos.

Já o critério propósito reflete a função do padrão. Segundo este critério, os padrões são classificados em:

- De Criação (Creational): padrões que abrangem a configuração e o processo de criação de objetos ou classes;
- Estruturais (Structural): padrões que trabalham com a composição de classes ou objetos;
- Comportamentais (Behavioral): padrões que definem o modo com que grupos de classes ou objetos interagem entre si e distribuem responsabilidades.

Estas categorias tornaram-se populares, juntamente com os seus respectivos padrões e são, freqüentemente, utilizadas na classificação de padrões de projeto por diversos outros autores, como o já citado Almanaque de Padrões [RIS00].

Além destas categorias de padrões, existem ainda outras, como padrões de suporte [FOW97], padrões organizacionais, padrões para planejamento de projeto [APP00].

2.3 Elementos de um padrão

O formato de documentação dos padrões é muito importante, uma vez que os mesmos devem ser entendidos de forma clara e objetiva pela comunidade de desenvolvedores de software. Uma boa descrição de seus campos faz com que a essência do problema seja imediatamente percebida e permite que se considere as conseqüências de sua aplicação além de fornecer os detalhes necessários para implementá-lo.

Assim, um desenvolvedor de software pode verificar se um determinado padrão se aplica ou não ao problema que está tentando solucionar. Entretanto, o que se pode observar é que os padrões têm sido descritos em diferentes formatos. Ao conjunto formado pelos campos componentes dos padrões, dá-se o nome de *template*. Assim sendo, o *template* é a forma do Padrão de Software se apresentar organizado em seções ou itens.

Como mencionado anteriormente, há uma heterogeneidade grande em relação aos *templates* dos padrões. Cada autor tem liberdade para descrever uma solução, em formato de padrão, escolhendo os campos mais adequados para isso. No entanto, de acordo com [AND01] e [MES97], alguns destes campos são considerados imprescindíveis para um padrão de software. Dentre esses campos, pode-se destacar: o nome do padrão, os campos que descrevem as circunstâncias e situação em que o problema a ser resolvido se encontra, o problema propriamente dito, as forças ou pontos a serem levados em consideração na solução do problema e a solução. Segundo Meszaros, essas idéias descritas podem estar contidas em campos distintos e em ordens diferentes de apresentação no padrão, dependendo do estilo de cada um. Mas são as idéias necessárias e básicas para a documentação de um padrão.

De acordo com o estudo detalhado em [SAN04] um *template* genérico pode ser definido para agrupar diversos campos existentes na literatura. A seguir, a descrição do mesmo:

- *Name* (nome): Descreve um nome que geralmente referencia o problema ou a solução. O nome torna-se rapidamente parte do vocabulário do desenvolvedor.

Todo padrão deve ter um nome significativo. Pode ser uma única palavra ou frase curta que se refira ao padrão e ao conhecimento ou estrutura descritos por ele. Se o padrão possuir mais do que um nome comumente usado ou reconhecível na literatura, subseções “Aliases” ou “Also know as” devem ser criadas.

- *Problem* (problema): Estabelece o problema a ser resolvido pelo padrão, descreve a intenção e objetivos do padrão perante o contexto e forças específicas. Expresso por uma simples pergunta ou por uma formulação resumida do problema.
- *Context* (contexto): Descreve em que circunstâncias o problema surge através de pré-condições dentro das quais o problema e sua solução costumam ocorrer e para as quais a solução é desejável, o que reflete a aplicabilidade do padrão. O contexto pode trazer um histórico de que padrões foram aplicados anteriormente. Pode também ser considerado como a configuração inicial do sistema antes da aplicação do padrão.
- *Forces* (forças): Descreve as considerações positivas e negativas, influências e restrições relevantes a serem avaliadas a fim de definir por que a solução proposta deve ser empregada.
- *Solution* (solução): Descreve claramente o que é necessário para resolver o problema, podendo para isso utilizar texto, diagramas e figuras.

Para a solução, são possíveis, também, as seguintes subseções:

- *Structure* (estrutura): Descreve a forma e organização estática do padrão.
- *Participants* (participantes): Descreve cada um dos componentes.
- *Dynamics* (dinâmicas): Descreve o comportamento dinâmico do padrão.
- *Implementation* (implementação): Descreve detalhes de implementação do padrão
- *Variants* (variantes): Descreve possíveis variações e especializações da solução.

- *Examples* (exemplos): Descreve uma ou mais aplicações do padrão que ilustram, num contexto inicial específico, como o padrão é aplicado e transforma aquele contexto em um contexto final.
- *Resulting context* (contexto resultante): Descreve a conclusão do padrão, as pós-condições e efeitos colaterais do mesmo. Em alguns formatos de padrões vem descrito como *Consequences* (conseqüências).
- *Rationale*: Descreve uma explicação das regras ou passos do padrão que explicam como e porque ele trata suas influências contrárias, definidas em *Forces*, para alcançar os objetivos, princípios e filosofia propostos. Isso nos diz realmente como o padrão funciona, porque funciona e porque ele é bom.
- *Related Patterns* (padrões relacionados): Descreve os relacionamentos estáticos e dinâmicos desse padrão com outros dentro da mesma linguagem ou do sistema de padrões. Pode referenciar outros padrões usados em conjunto ou outras soluções para o problema, ou ainda, variações do padrão. Padrões relacionados geralmente compartilham as mesmas influências. Em alguns formatos é apresentado como *See also* (veja também).
- *Known Uses* (usos conhecidos): Descreve exemplos da aplicação do padrão e suas aplicação em sistemas reais existentes. Isso ajuda a validar o padrão, verificando se ele é realmente uma solução provada para um problema recorrente.

Todos os campos do *template* apresentados nesta seção são usados na literatura para descrever padrões de software. Observa-se que muitos itens desse *template* genérico possuem um relacionamento entre si. Por exemplo, pode-se perceber que exatamente a mesma idéia de um campo pode ser representada por denominações diferentes, como *Resulting Context* e *Consequences*.

Além disso, observa-se que alguns campos apresentam um relacionamento de completude (antes e depois). Por exemplo, após ler o campo *Context* cria-se a expectativa para conferir o campo *Resulting Context*. O mesmo ocorre com *Problem* e *Solution*. Esse tipo de relacionamento entre os campos de um determinado padrão pode gerar algumas regras para a associação de um grau de relacionamento entre os padrões. Por exemplo, se o conteúdo do campo *Context*

de um padrão estiver relacionado a *Resulting Context* de outro padrão, pode ser percebido um tipo de relacionamento entre esses padrões, mesmo eles não sendo de um mesmo agrupamento. Os diversos tipos de relacionamentos entre os padrões serão explorados na próxima seção.

2.4 Relacionamentos entre padrões de software

Padrões de software podem se relacionar uns com os outros, sejam eles padrões pertencentes a uma mesma coleção, ou padrões de origens distintas. Diferentes soluções para um mesmo problema podem gerar padrões diferentes. Padrões podem ainda representar uma evolução de outros padrões já existentes. É comum também encontrarmos padrões que especializam ou generalizam soluções de outros padrões, ou ainda, padrões podem trabalhar em conjunto em linguagens ou catálogos de padrões. De uma forma geral, não existem padrões isolados: um padrão pode depender de outro no qual esteja contido, ou das partes que ele contém, ou ser uma variação de outro, ou ser uma combinação de outros.

Estes são exemplos que descrevem alguns dos tipos de relacionamentos existentes entre padrões de software. Muitos *templates* de padrões trazem como um de seus campos a seção Related Patterns, onde os autores descrevem de forma textual os padrões relacionados e o tipo de relacionamento entre eles. Essa descrição pode gerar mapas, que são representações gráficas dos relacionamentos entre padrões que algumas linguagens e catálogos de padrões possuem.

Zimmer [ZIM95] organiza os relacionamentos entre os padrões de projeto da Gang of Four [GAM95] em categorias a fim de facilitar o entendimento da estrutura completa do catálogo. Os relacionamentos entre um par (X,Y) de padrões foram divididos em três tipos descritos abaixo:

- X usa Y na sua solução,
- X é semelhante a Y (is similar to), e

- X pode ser combinado com Y.

Na figura 01, a seguir, os padrões estão relacionados de acordo com o campo *Related Patterns*, mostrando o modo como cada um se referencia o outro.

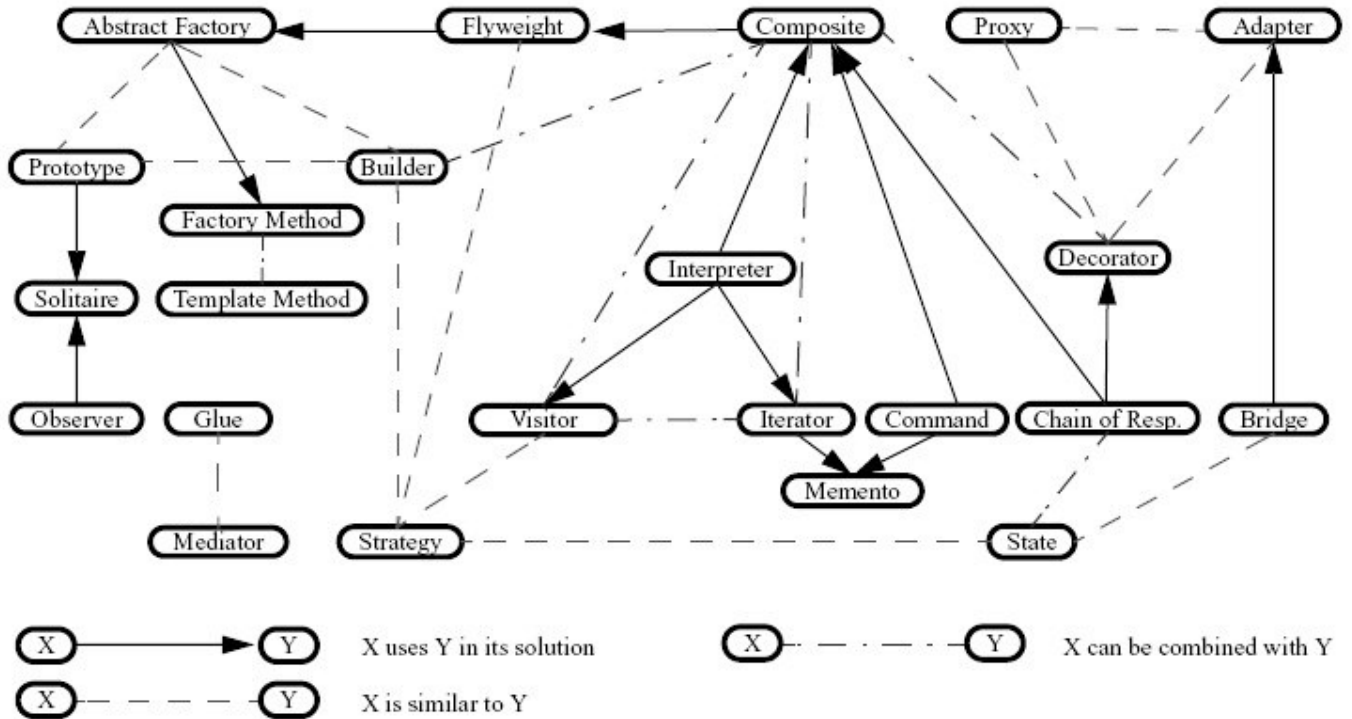


Figura 01 – Classificação dos Relacionamentos entre Padrões do GoF [ZIM95]

Em [CON02], são formalizados quatro tipos de relacionamentos possíveis entre padrões.

- Se um padrão P1 usa um padrão P2, então a solução do padrão P1 deve ser expressa usando P2;
- Se um padrão P1 refina um padrão P2, então o problema do padrão P1 deve ser uma especialização do padrão P2;
- Se um padrão P1 requer um padrão P2, então aplicação do padrão P2 é exigida na aplicação de P1;

- Se um padrão P1 é alternativo a um padrão P2, então os padrões P1 e P2 fornecem soluções diferentes para o mesmo problema.

Com estas definições, pode-se chegar a mais uma classificação sobre os tipos dos padrões. Os padrões que possuem relacionamento com um determinado padrão podem ser classificados como:

- Predecessores: são padrões cujas aplicações conduzam a um padrão;
- Sucessores: são padrões que devem ser aplicados após o uso do padrão em questão;
- Alternativos: são padrões que descrevem uma solução diferente para o mesmo problema, possivelmente diante de influências e restrições diferentes;
- Codependentes: são padrões que podem (ou devem) ser aplicados simultaneamente com esse padrão.

Dados os padrões de uma mesma linguagem, o discernimento dos relacionamentos existentes se torna fácil, pois os mesmos se apresentam de forma patente. No entanto, em um vasto número de padrões na literatura, ainda não existe uma técnica que viabilize associar um padrão ao outro independente deles estarem num mesmo agrupamento.

Um outro dado interessante é que os tipos de relacionamentos poderiam ser mais um artifício útil numa busca por padrões avançada dentro de um Repositório de Padrões. O usuário poderia estar interessado em saber, por exemplo, que padrões “especializam” o padrão *Strategy*.

2.5 Agrupamentos de Padrões

Ao documentarem um padrão de software, seus autores identificam outros padrões que estão relacionados com o mesmo, seja devido ao domínio de aplicação ou algum dos tipos de relacionamentos apresentados na seção anterior. Dessa forma, procuram-se maneiras de agrupar os padrões existentes seguindo algum critério, de forma a facilitar sua recuperação e reuso. Isso pode ser feito por

meio de coleções de padrões, catálogos de padrões ou linguagem de padrões. Esta seção pretende discutir esses tipos de agrupamento de padrões, explicitando a diferença entre eles e a utilidade de cada um deles em particular.

2.5.1 Coleção de padrões

É uma coletânea qualquer de padrões que não possuem nenhum vínculo entre si e, em geral, nenhuma padronização no formato de apresentação. Os padrões podem estar reunidos por terem sido apresentados em um mesmo congresso, por terem sido propostos pelo mesmo autor, ou por se referirem a um mesmo domínio, mas não possuem um relacionamento semântico significativo.

Em [VLI97] pode-se perceber uma coleção de padrões, no caso dos padrões apresentados em PLOP'95 (Pattern Language of Programming). Estudando esses padrões de forma detalhada, observa-se que eles abrangem domínios e níveis de abstração variados, e não possuem relacionamento explícito. São padrões escritos por diversos autores diferentes e de maneira independente, isto é, um autor não tinha conhecimento do que os demais autores estavam escrevendo. O editor do livro agrupou os padrões que pertencem ao mesmo domínio ou fase do processo de desenvolvimento em capítulos, mas isso não é suficiente para que essa coleção de padrões possa ser considerada um catálogo de padrões.

2.5.2 Catálogo de padrões

Trata-se de uma coleção de padrões mais estruturada e com algum tipo de relacionamento. Em geral, ele subdivide os padrões em um pequeno número de categorias mais abstratas, que abrangem uma grande quantidade de padrões, e pode incluir algumas referências cruzadas entre os mesmos [APP00]. Um catálogo de padrões pode oferecer um esquema de classificação e recuperação de seus padrões, já que eles estão subdivididos em categorias.

[GAM95] é um catálogo de padrões de projeto, pois possui mais estrutura e organização, exibindo também relações entre os padrões. Nesse catálogo, pôde-se perceber os critérios para classificação dos padrões: escopo e propósito.

- No critério “escopo” decide-se se o padrão atua sobre uma classe ou sobre objetos da classe. Padrões da classe lidam com os relacionamentos entre classes e suas subclasses, por meio do uso de herança, sendo portanto estabelecidos de forma estática (em tempo de compilação). Padrões do objeto lidam com relacionamentos entre objetos, que podem ser modificados durante a execução e são mais dinâmicos. A maioria dos padrões utiliza herança de alguma forma. Portanto os únicos padrões classificados na categoria “classe” são os que se concentram em relacionamentos de classes.

- No critério “propósito” existem padrões de criação, padrões estruturais e padrões comportamentais. Padrões de criação concentram-se no processo de criação de objetos. Padrões estruturais lidam com a composição de classes ou objetos. Padrões comportamentais caracterizam as formas pelas quais classes ou objetos interagem e distribuem responsabilidades.

Gamma também propõe uma outra forma de organizar seus padrões de projeto, mostrada na figura 02 a seguir.

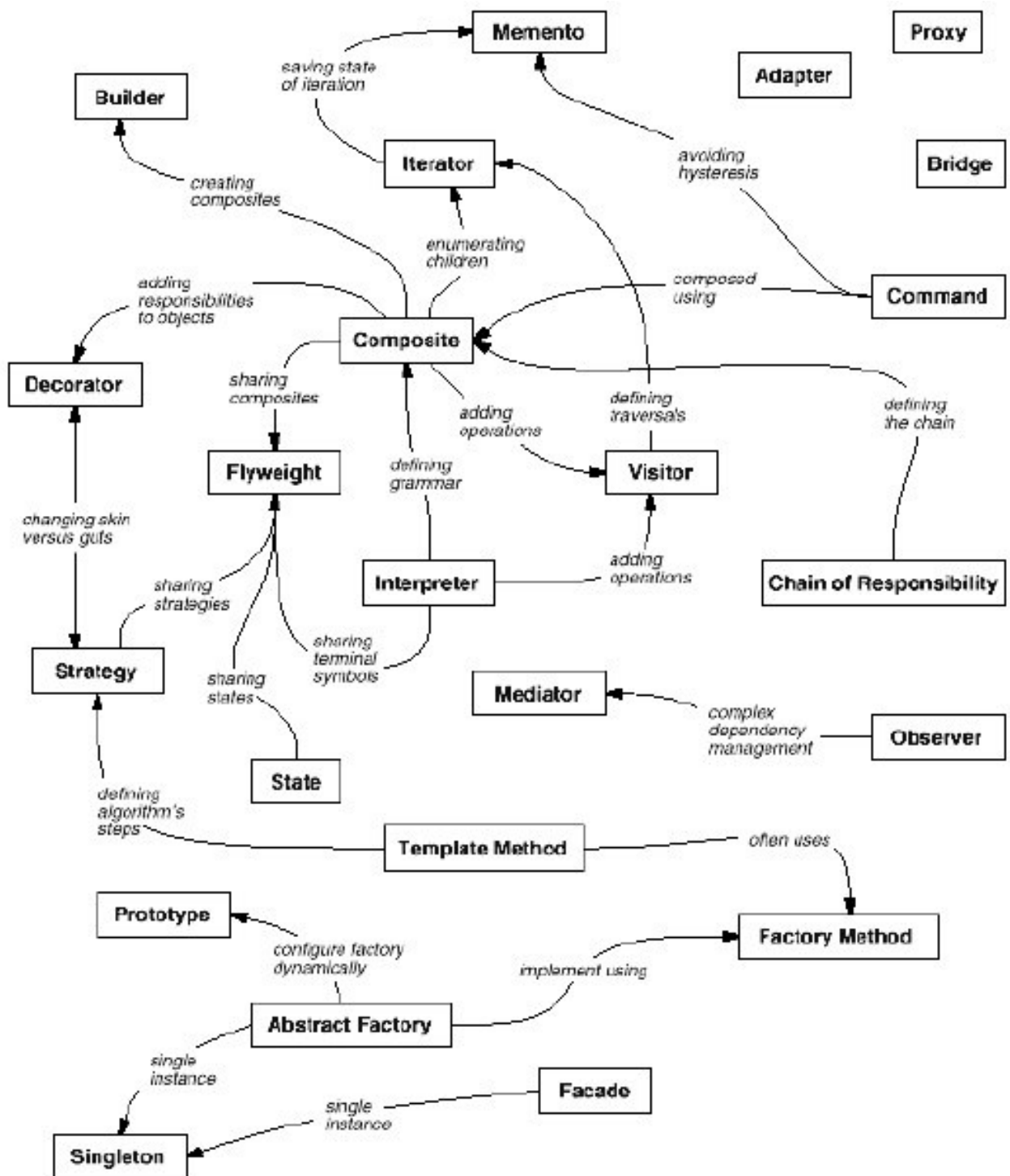


Figura 02 – Classificação dos Relacionamentos entre Padrões do GoF

2.5.3 Linguagem de padrões

É uma coleção estruturada de padrões que se apoiam uns nos outros para transformar requisitos e restrições numa arquitetura [COP98]. Os padrões que constituem uma linguagem de padrões cobrem todos os aspectos importantes em um dado domínio. Cada padrão da linguagem resolve um problema específico no contexto comum compartilhado pela linguagem. Uma linguagem de padrões é uma forma de subdividir um problema geral e sua solução complexa em um número de problemas relacionados e suas respectivas soluções.

É importante notar que cada padrão pode ser usado separadamente ou com um certo número de padrões da linguagem. Isso significa que um padrão sozinho é considerado útil mesmo se a linguagem não for usada em sua plenitude. A seguir, a figura 03 mostra a linguagem de padrões MoRaR:

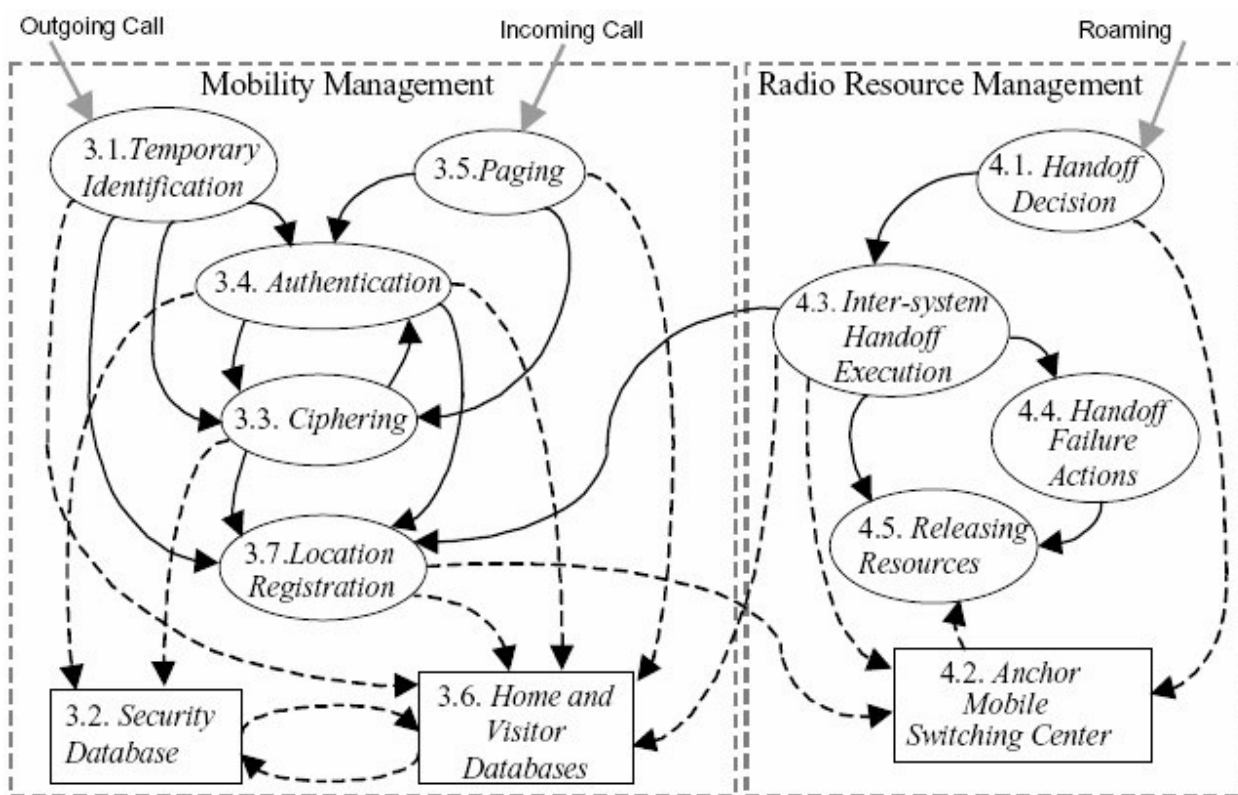


Figura 03 – MoRaR - Uma Linguagem de Padrões para o Gerenciamento de Mobilidade e de Recursos de Rádio [AND01]

2.6 Repositório de Padrões de Software

Seja em websites, em livros ou artigos das várias conferências PloP (*Pattern Language of Programming*), existe uma grande quantidade de padrões de software disponíveis. Esses padrões estão aptos a serem reusados por engenheiros de software nas diferentes fases de desenvolvimento de sistemas.

Os padrões de projeto da Gang of Four (GoF) [GAM95], o catálogo de padrões Core J2EE da Sun [COR05], os padrões de análise propostos por Fowler [FOW97] e os padrões voltados para a arquitetura de software (POSA) [BUS96] são exemplos de padrões disponíveis na literatura. Além disso, alguns sites já tradicionais nesta área trazem uma boa quantidade de padrões catalogados. O Grupo Hillside [HIL05], que é uma organização sem fins lucrativos que busca regular a área de padrões de software, mantém o site oficial de padrões que disponibiliza um sistema de meta-busca para padrões. Esse sistema traz como resultados: endereços de websites, referências para livros e, muitas vezes, os sites dos próprios autores dos padrões procurados.

Existem também sites especializados em catalogar padrões de domínios de aplicação específicos, o que é útil para pesquisadores e desenvolvedores que buscam soluções para estes domínios específicos em particular. É o caso do SecurityPatterns.org [SEC06], que traz vários padrões sobre de segurança de sistemas, o XMLPatterns.com [XML06], que traz um catálogo de padrões para XML, o phpPatterns [PHP06] que possui referências de trabalhos sobre padrões para desenvolvimento em PHP, e o Amsterdam Patterns Collection [WEL06], que traz uma coleção de padrões classificados como padrões de projeto para Web, GUI e MobileUI.

No entanto, todas essas abordagens ainda são limitadas, pois os padrões são apresentados em sites e literaturas diferentes, não possuindo qualquer correlação entre si. Eles são catalogados de maneira bastante específica.

Em [AND03], foi apresentada uma proposta de um repositório de padrões de software que posteriormente podia ser integrado automaticamente ao RUP (*Rational Unified Process*) [KRU99]. O objetivo principal desse repositório é tornar os mais diversos tipos de padrões disponíveis para busca e a posterior recuperação dos dados. A seguir, são apresentadas algumas das principais funcionalidades do repositório de padrões de software desenvolvido.

O repositório não adota um formato específico para os padrões o que torna possível incluir um padrão usando os campos dos *templates* de padrões mais usados na literatura. Associado ao padrão deve ser cadastrado um conjunto de palavras-chave. O repositório pode associar arquivos relacionados a cada padrão, tais como: códigos-fonte, modelo do padrão e imagens ilustrativas.

O repositório também permite o registro dos relacionamentos entre os padrões. Além disso, o repositório deve possuir uma busca eficiente que forneça como resultado padrões relacionados aos elementos da pesquisa informados pelo usuário.

Finalmente, o repositório disponibiliza aos usuários, todos os dados e arquivos relacionados ao padrão selecionado após uma pesquisa para a visualização em uma tela do próprio sistema ou via URL.

A utilização plena do Repositório facilita a reutilização de Padrões de Software por parte dos usuários, principalmente, por mostrar padrões de diversas origens e autores em um mesmo ambiente, possibilitando alternativas de projeto ao desenvolvedor que fizer uso do Repositório.

No entanto, dentro do Repositório, ainda há limitações referentes ao processo de busca por padrões, uma vez que a mesma é realizada no modo tradicional, com uma análise sintática do conteúdo dos campos. Outra característica limitante desses sistemas é a ausência da determinação automática de relacionamentos entre os padrões. Não há um mecanismo que determine um grau de relacionamento entre os padrões baseado numa análise semântica do seu conteúdo.

2.7 Conclusão

Neste capítulo foi descrito como o uso de padrões determina boas soluções que passam a ser documentadas, transmitidas e adotadas entre desenvolvedores de software. Também pôde-se perceber quão eficaz a utilização de padrões se torna para difusão do conhecimento, pois através do estudo de padrões encontrados em livros e em páginas da Internet pode-se aprender com a experiência de desenvolvedores do mundo inteiro boas práticas de desenvolvimento de software.

No entanto, melhorias ainda são necessárias na recuperação e disponibilização de Padrões de Software dos mais diversos tipos para os interessados em sua utilização. Para sanar parte dessa deficiência, foram propostos Repositórios de Padrões que armazenam os dados dos mesmos em um sistema que busca e captura esses documentos. Contudo, esses repositórios ainda se apresentam limitados quando se faz necessário conhecer um maior relacionamento entre os padrões cadastrados no sistema.

Baseado nessas características, nesta dissertação é proposta uma solução que utiliza técnicas de Mineração de Texto para facilitar a recuperação dos padrões e a determinação dos graus de relacionamentos entre os mesmos.

Capítulo 3

Análise da Semântica Latente

Neste capítulo, é apresentada uma visão geral da Análise da Semântica Latente, uma sub-área da Mineração de Dados e Mineração de Texto, aplicada no decorrer do trabalho. Na seção 3.1, é introduzida toda a Análise da Semântica Latente, foco principal na Mineração de Dados dentro da Inteligência Artificial e sua abrangência. Na seção 3.2, é apresentada a Mineração de Dados bem como técnicas comuns dessa área, suas utilidades e especificidades. Na seção 3.3 a Mineração de Textos é inserida. E na seção 3.4, a semântica latente é apresentada como uma possível solução para o problema em questão deste trabalho. Por fim, é apresentada uma conclusão na seção 3.5.

3.1 Introdução

Um dos atuais desafios na Ciência da Computação ainda é descobrir informações úteis e previamente desconhecidas dada uma grande quantidade de texto. O processo capaz de descobrir este conhecimento em banco de dados chama-se KDD (Knowledge Discovery Database). O processo de KDD foi proposto em 1989 para referir-se às etapas que produzem conhecimentos a partir dos dados e, principalmente, à etapa de mineração dos dados, que é o processo que transforma dados em informações [FAY96].

Uma abordagem técnica para o tratamento de textos com mineração de dados e escolhida para tratar o problema deste trabalho é a Análise da Semântica Latente [KIN02]. A Análise da Semântica Latente ou LSA (Latent Semantic Analysis) é usada para definir o tema de um texto ou gerar um sumário automaticamente. A informação sobre a temática do conteúdo do texto pode ser representada por um vetor no espaço semântico: o texto provê nova informação sobre este tema e pode potencialmente modificar e expandir o espaço semântico por ele mesmo. Vetores podem, do mesmo modo, representar subseções de um texto. LSA pode ser

usada para selecionar de cada subseção a sentença ou tópico mais relevante, além de gerar um tipo de sumário automaticamente.

3.2 Mineração de Dados

O processo de MD (Mineração de Dados) é descrito na literatura como diferente de uma receita ou um roteiro que deve ser seguido para se obter bons resultados [FAY96]. É esta fase que executa o algoritmo que diante da tarefa especificada, busca extrair o conhecimento implícito e potencialmente útil dos dados.

Este processo pode ser visto como uma atividade de engenharia, na qual são requeridas experimentação, análise e comparação de diferentes algoritmos e modelos na busca por resultados úteis.

De acordo com [VEL05], o processo de mineração de dados é interativo e iterativo: não se pode obter algum tipo de conhecimento importante simplesmente submetendo uma certa quantidade de dados a um sistema de MD. O usuário de um sistema de Mineração de Dados deve ter um forte embasamento no domínio da aplicação para poder selecionar os subconjuntos de dados adequados para descoberta de conhecimento de *padrões** bem como definir critérios apropriados para avaliá-los. Dessa forma, um sistema de MD deve ser visto como um conjunto de ferramentas interativas e não como uma ferramenta de análise automática. O processo de MD consiste de um conjunto de fases, que podem ser reunidas nos seguintes grupos principais:

1. processamento de dados (ou pré-processamento);
2. busca por *padrões**;
3. processamento de *padrões** (ou pós-processamento).

O processamento dos dados é a etapa mais desgastante em todo o processo de Mineração de Dados pois é necessário bastante tempo para a aquisição de conhecimento. Entender o domínio dos dados é um pré-requisito natural para a extração de qualquer informação útil, uma vez que bases de dados reais

geralmente apresentam características que impedem a aplicação imediata das ferramentas de extração de *padrões**, tais como dados dinâmicos, ruído, incerteza, dados incompletos, informações redundantes e dados esparsos.

A preparação dos dados envolve tarefas tais como a definição dos atributos que serão utilizados, extração e integração dos dados provenientes de diferentes fontes, transformação dos dados para formatos mais apropriados para a extração de conhecimento, limpeza e composição de atributos. A redução do conjunto de dados justifica-se quando o uso de grandes bases de dados pode exceder a capacidade de processamento do algoritmo de aprendizagem, ou como uma tentativa de diminuição do tempo de resposta para se encontrar uma solução.

A busca por *padrões** é a etapa na qual os *padrões** interessantes e que ocorrem mais freqüentemente são descobertos a partir dos dados. Essa etapa pode usar, entre outras, várias técnicas de Aprendizado de Máquina (AM), tais como indução de regras, árvores de decisão, agrupamento (clustering), redes neurais, programação lógica indutiva e métodos baseados em distância. Embora os métodos e algoritmos de AM estejam no núcleo da descoberta de conhecimento a partir dos dados, existem algumas diferenças entre esta e MD.

A ênfase do processo de Mineração de Dados é uma delas; uma grande parte da literatura em AM se concentra principalmente na fase de aprendizado ou indução, enquanto que em Mineração de Dados, a preparação dos dados e a interpretação do conhecimento induzido recebem uma maior atenção do que a fase de aprendizado.

A próxima diferença se concentra no papel relativo do conceito e dos dados. Nas pesquisas em AM, assume-se que exista um conceito oculto que cubra os dados e o objetivo é descobrir esse conceito. Os dados podem estar corrompidos por ruído, erros, etc., mas a idéia geral ainda é a de que exista um conceito por trás de todo o conjunto de dados.

Entretanto, em Mineração de Dados essa premissa não é necessariamente válida. Descobrir *padrões** úteis isolados, que somente cubram um subconjunto de todo o conjunto de dados pode ser suficiente para descobrir conhecimento.

Uma outra diferença é a quantidade de dados. Tradicionalmente, a pesquisa em AM se concentra em conjuntos de dados contendo centenas ou milhares de exemplos, enquanto que aplicações de Mineração de Dados se concentram em conjuntos de dados maiores. Entretanto, a principal fonte de complexidade em Mineração de Dados não é o número de exemplos do conjunto de dados, uma vez que esse problema pode ser contornado utilizando amostragem, mas sim o número de atributos utilizados para descrever os exemplos.

Uma outra diferença ainda refere-se à validação dos padrões descobertos. A maioria das abordagens de AM está fortemente baseada em medidas de validação que consideram a precisão do conhecimento induzido, enquanto que em MD outras medidas são necessárias, tais como níveis de interesse e novidade de alguns desses *padrões**. A capacidade de compreender e a usabilidade são também outros critérios que devem ser considerados em MD para a validação dos *padrões**.

O processo de Mineração de Dados não termina quando os *padrões** são encontrados. Os *padrões** descobertos devem ser entendidos, avaliados, confrontados com o conhecimento já existente, ordenados, filtrados e traduzidos em termos inteligíveis para os usuários. A seguir, tem-se a incorporação do conhecimento extraído, seja por meio da documentação e disponibilização desse conhecimento às partes interessadas ou pela tomada de decisões baseadas nesse conhecimento.

Dessa forma, percebe-se que a Mineração de Dados tem também uma forte correlação com outras áreas, tais como Estatística, Banco de Dados, Reconhecimento de Padrões, Inteligência Artificial, Aquisição de Conhecimento para Sistemas Especialistas, Visualização de Dados e Computação de Alto Desempenho. O ponto em comum é a extração de conhecimento de alto nível a partir de dados. Em particular, Mineração de Dados têm muito em comum com a Estatística, particularmente com métodos de análise exploratória de dados. A abordagem estatística oferece métodos precisos para quantificar a inerente

incerteza dos resultados quando tenta-se inferir padrões gerais a partir de um conjunto particular de dados extraídos de uma população.

Outro fator importante é que o processo de MD admite várias visões, dependendo da funcionalidade desejada. Na transformação do processo idealizado em procedimentos, este pode ser descrito em termos de um processo hierárquico [CHA00], consistindo de um conjunto de tarefas descritas em quatro níveis de abstração.

3.2.1 Regras de Associação

A tarefa de descobrir regras de associação foi introduzida por Agrawal [AGR93]. Na sua forma original essa tarefa é definida para um tipo especial de dados, freqüentemente chamado *basket data*. Nesse tipo de dado uma tupla consiste num conjunto de atributos binários chamados itens. Cada tupla corresponde à uma transação, e um campo pode assumir um valor verdadeiro ou falso, dependendo se o ele está ou não presente na transação. A transformação de uma tabela da base de dados para o formato binário é feita de modo que um atributo de cardinalidade K é substituído por K atributos binários [ADR97].

Uma regra de associação é um relacionamento da forma $X \Rightarrow Y$, onde X e Y são conjuntos de itens e a interseção entre eles, $(X \text{ interseção } Y)$, é o conjunto vazio. Cada regra de associação é associada a um fator suporte superior, denominado $FSup$, e a um fator de confiança, $FConf$. $FSup$ é definido como a razão do número de tuplas satisfazendo ambos X e Y sobre o número total de tuplas, isto é, $FSup = |X \text{ união } Y| / N$, onde N é o número total de tuplas. $FConf$ é definido como a razão do número de tuplas que satisfazem ambos X e Y sobre o número de tuplas que satisfazem X , isto é, $FConf = |X \text{ união } Y| / |X|$. A tarefa de descobrir regras de associação consiste em extrair do banco de dados todas as regras com $FSup$ e $FConf$ maiores ou iguais a um $FSup$ e $FConf$ especificado pelo usuário. A definição desses fatores serve para introduzir uma medida capaz de distinguir associações interessantes, dado que as regras geradas em forma binária elevam em muito o espaço de busca de qualquer algoritmo minerador.

A descoberta de regras de associação é geralmente executada em 2 passos. Primeiramente, um algoritmo determina todos os conjuntos de itens que têm FSup maior ou igual ao FSup especificado pelo usuário. Estes conjuntos são chamados conjuntos de itens freqüentes. Segundo, para cada conjunto de itens freqüentes, todas as possíveis regras candidatas são geradas e testadas com respeito ao FConf. Uma regra candidata é gerada por algum subconjunto de itens de um conjunto de itens freqüentes como antecedente da regra, e os itens remanescentes do conjunto de itens freqüentes como o consequente da regra. Apenas regras candidatas com FConf maior ou igual ao FConf especificado pelo usuário são dadas como saída do algoritmo.

3.2.1 Classificação

Na tarefa de classificação, cada tupla pertence a uma classe entre um conjunto pré-definido de classes. A classe de uma tupla, ou registro, é indicada por um valor especificado pelo usuário em um atributo meta, ou atributo objetivo. As tuplas consistem de atributos preditivos e um atributo objetivo, esse último indicando a que classe essa tupla pertence. O atributo objetivo é do tipo categórico, ou discreto, isto é, pode tomar apenas um valor dentro de um conjunto de valores discretos, determinando classes ou categorias. Esse atributo pode ter valores discretos como SIM ou NÃO, um código pertencente a um intervalo de números inteiros, tais como {1..10}, entre outros.

O princípio da tarefa de classificação é descobrir algum tipo de relacionamento entre os atributos preditivos e o atributo objetivo, de modo a descobrir um conhecimento que possa ser utilizado para prever a classe de uma tupla desconhecida, ou seja, que ainda não possui uma classe definida.

3.2.3 Agrupamento

O agrupamento é um método de descoberta de conhecimento utilizado para identificar co-relacionamentos e associações entre objetos, facilitando, assim, a identificação de classes. No caso de documentos, o clustering identifica os documentos de assunto similar e aloca-os em um grupo, gerando grupos de documentos similares. Esse método é extremamente útil quando não se tem uma idéia dos assuntos (das classes) tratados em cada documento e deseja-se separá-los por assunto.

Essa técnica é geralmente utilizada antes de um processo de classificação, facilitando a definição de classes, pois o especialista pode analisar os co-relacionamentos entre os elementos de uma coleção de documentos e identificar a melhor distribuição de classes para os objetos em questão. Isso significa que não há a necessidade de se ter conhecimento prévio sobre os assuntos dos documentos ou do contexto dos documentos. Os assuntos e as classes dos documentos são descobertos automaticamente pelo processo de agrupamento.

Em geral, o agrupamento possui diversas aplicações. Ele pode ser utilizado para facilitar a organização e a recuperação de informações ou em outros processos de análise textual que visam a descoberta de conhecimento a partir de textos.

A recuperação de informações é facilitada porque o método desenvolvido consegue processar uma grande quantidade de documentos (de assuntos diversos) e agrupá-los em clusters de documentos de assuntos similares. Os grupos de documentos similares são armazenados em um mesmo local no arquivo de dados e indexados de forma que todo um cluster seja recuperado quando um dos documentos que fazem parte dele for considerado relevante a uma consulta.

Já na área de descoberta de conhecimento em textos o agrupamento é comumente utilizado no processo de descoberta de associações entre palavras, facilitando o desenvolvimento de dicionários e thesaurus. Esses dicionários podem ser utilizados em ferramentas de busca ou editoração de documentos, expandindo consultas ou padronizando o vocabulário dos documentos em edição.

3.2.4 CRISP-DM

O grupo de pesquisa CRISP-DM [CRI05], referência na área de Mineração de Dados, apresenta um modelo de processo para mineração de dados que provê uma visão abrangente do seu ciclo de vida. Descreve as fases correspondentes do projeto, suas respectivas tarefas e relacionamentos entre as tarefas. Neste nível de descrição, não é possível identificar todos os relacionamentos, mas provavelmente existe relacionamentos entre todas as tarefas de mineração de dados dependendo de seus objetivos, *background* e interesse do usuário e, o mais importante, dependendo do tipo de dados.

Logo a seguir, são mostradas as principais fases desse processo:

- Entendimento do projeto: a fase inicial é focada no entendimento dos objetivos do projeto e seus requisitos. Esse conhecimento é convertido numa definição de problema de mineração de dados e um plano preliminary é desenvolvido para alcançar esses objetivos.
- Entendimento dos dados: a fase de entendimento dos dados se inicia com uma coleção de dados inicial e se procede através de uma série de atividades a fim de que os dados se tornem familiares e visando identificar problemas de qualidade dos dados ou descobrir subconjuntos interessantes para formar hipóteses de sobre informações escondidas.
- Preparação dos dados: esta fase sobre todas as atividades que constroem o conjunto final de dados. Essas atividades são executadas muitas vezes e sem uma ordem prevista. Algumas dessas tarefas incluem seleção de tabelas, registros e atributos bem como transformação e simplificação dos dados para as ferramentas de modelagem.
- Modelagem: nesta fase, várias técnicas de modelagem são selecionadas e aplicadas, e seus parâmetros são calibrados para os valores ótimos. Normalmente, existem várias técnicas para o mesmo tipo de problema de mineração de dados. Algumas delas, têm alguns requisitos específicos no

formato dos dados. Eventualmente, pode haver a necessidade de voltar à fase anterior.

- **Avaliação:** neste estágio do projeto o desenvolvedor já construiu um modelo (ou modelos) que aparenta ter alta qualidade, numa perspectiva de análise de dados. Antes de proceder com o fechamento do modelo, é importante avaliá-lo mais profundamente e revisar os passos executados para construí-lo para ter certeza de que o modelo proposto alcança os objetivos do projeto. No final desta fase, deve haver uma decisão a respeito do uso dos resultados da mineração de dados.

- **Depuramento:** a criação do modelo não é necessariamente o final do projeto. Mesmo se o propósito do modelo é aumentar conhecimento dos dados, o conhecimento obtido necessita ser organizado e apresentado de uma forma que o usuário possa usá-lo. Dependendo dos requisitos, esta fase pode ser simples como a geração de um relatório ou complexa como a repetição da implementação de um processo de mineração de dados.

3.3 Mineração de Texto

Na década de 90, com a disponibilidade de grandes quantidades de dados e os avanços tecnológicos para sua manipulação eletrônica, tornou-se possível a aplicação de métodos computacionais e estatísticos em larga escala. Esses métodos são comumente chamados métodos baseados em corpora e têm demonstrado que é possível, para alguns gêneros textuais ou domínios particulares do conhecimento, obter-se sumários bons e úteis para algum objetivo previamente estabelecido. Corpora on-line impulsionaram novos métodos para estudos em uma variedade de áreas, além da sumarização automática, como aquisição de conhecimento lexical, revisão gramatical, tradução automática e mineração de dados.

Corpora considerados relevantes para a investigação empírica em diversas áreas incluíram, inicialmente, o Brown Corpus [FRA82] e o Birmingham Corpus [SIN87], ambos com mais de um milhão de palavras. No entanto, atualmente, esses corpora já foram superados por outros corpora textuais. O University Centre for Computer Corpus Research on Language (UCREL), por exemplo, um centro de pesquisas da Universidade de Lancaster, reúne 21 corpora, totalizando cerca de 26 milhões de palavras, além de inúmeros textos em CD-ROM.

Outros corpora atualmente bastante explorados são aqueles desenvolvidos especialmente para tarefas dedicadas do PLN, como os utilizados nas TREC (Text REtrieval Conference) ou nasMUC (Message Understanding Conference).

Esses corpora, também utilizados na área de Data Mining, motivaram o surgimento da área de Mineração de Textos (Text Mining), ou Text Data Mining (TDM) [HEA99], área de bastante interesse para as áreas de Recuperação da Informação e Sumarização Automática. Enquanto em mineração de dados contempla-se dados estruturados, em mineração de textos busca-se o estudo das relações existentes entre componentes de textos não estruturados. Esse inter-relacionamento pode ser interno, isto é, relativo a apenas um texto, ou externo, abrangendo vários textos, dependendo do objetivo da aplicação. Para a recuperação da informação, por exemplo, é interessante buscar informações

representativas e, portanto, distintivas de algum texto em particular. Portanto, para distinguir um texto de outro, é necessário investigar vários textos.

Em mineração de textos, utiliza-se também grande volume de dados, agora textuais, a fim de se buscar *padrões* relevantes por comparação. A dificuldade, neste caso, é lidar com a vasta gama de informações, em geral desestruturadas e, assim, de difícil recuperação automática.

Portanto, a mineração de textos consiste na mineração de grandes coleções de bancos de dados de textos. Há uma larga quantidade de informação disponível na forma de publicações eletrônicas de livros, bibliotecas digitais, mensagens eletrônicas, documentos técnicos e de negócios, etc. Os mecanismos de busca na Internet (search engines), combinados com várias técnicas de análise de textos, têm ido na direção da mineração de textos online também [MIT03].

Verifica-se o extenso material produzido pela IBM, a qual considera a mineração de textos como a “aplicação da idéia de mineração de dados para textos não estruturados ou menos estruturados.” Também, entende a mineração de dados como a possibilidade do “proprietário ou usuário dos dados obter novas visões e conhecimento procurando padrões nos dados os quais não seriam reconhecidos usando consultas tradicionais de dados e técnicas relacionadas. Estas técnicas permitem comparações para serem feitas através de dados vindos de muitas fontes de diferentes tipos, extraindo informação que não poderiam ser óbvias ou mesmo visíveis para o usuário e organizar documentos e informação pelos seus assuntos ou temas.” [WOH98]

Uma outra conceituação encontra-se junto ao Text Mining Research [WAI05] da Universidade de Waikato, desenvolvedores da ferramenta Weka [WEK05], para quem a *“Mineração de textos é a procura por padrões em um texto em linguagem natural e pode ser definido como o processo de análise do texto para extrair informação dele para um propósito em particular.”*

Diante dessa conceituação verifica-se que a mineração de textos possui duas fases principais e seqüentes: a extração de informações e a mineração de dados propriamente dita. A primeira destina-se a extrair conceitos, estatísticas e palavras

relevantes de um conjunto textual para estruturá-los minimamente, preparando-os para a aplicação das técnicas de mineração de dados. Neste segundo momento, aplicam-se as diretrizes e algoritmos de mineração de dados destinados a gerarem regras, classificações ou agrupamentos.

O mesmo grupo de pesquisa [WAI05] afirma que *“A mineração de textos utiliza recall e precisão para medir a efetividade das diferentes técnicas de extração de informação, permitindo a realização de comparações quantitativas. Estes conceitos seguem os mesmos princípios da recuperação de informação.”*

A extração de informações cumpre um importante papel no processo de mineração de textos, visto que ele permite o pré-processamento do corpo dos documentos textuais de maneira a facilitar o processo de mineração. Geralmente, o processo de extração de informações gera pares de atributos e valores que representam o conteúdo do documento. Normalmente, os atributos são um conjunto previamente definidos.

A mineração de textos deve, também, preocupar-se com os problemas básicos de processamento de linguagem natural, tais como: erros de ortografia e digitação, problemas de normalização de palavras, com suas flexões de gênero e plural, e ambigüidades semânticas. Para melhor entender isso, a explicação final de [WAI05] cita que *“A mineração de textos reconhece que o completo entendimento de linguagem natural é um objetivo que não pode ser atingido imediatamente e foca a extração de uma pequena quantidade de informação. Esta informação extraída pode ser o autor, o título e a data de publicação de um artigo, as siglas definidas em um texto ou os artigos mencionados em uma bibliografia.”*

3.3.1 Análise das Ferramentas de Mineração de Texto

Existem poucas ferramentas que suportam todo o processo de Mineração de Dados. Além disso, ferramentas comerciais, tais como Mineset™ e IntelliMiner™, geralmente têm um caráter mais exploratório e fazem uso de algoritmos e ferramentas proprietários, o que dificulta o seu uso por pesquisadores

que pretendem analisar e desenvolver novos algoritmos e ferramentas. Além disso, o custo de aquisição dessas ferramentas não seria permitido para universidades públicas, mesmo para um grande grupo de pesquisa.

Como exemplo dessas ferramentas [ANA05], pode-se citar a TextAnalyst[MEG05], Semio [SEM05], Intelligent Miner for Text e AlphaThemis.

TextAnalyst destaca-se como uma ferramenta para análise semântica, navegação e procura em textos não-estruturados. Está disponível como aplicação standalone, SDK de componentes COM para fácil integração, purificação do Resultado (Rede Semântica), navegação amigável, explicação em tópicos e sumarização.

A ferramenta Semio também procura extrair automaticamente dos conceitos chaves de grandes volumes de conteúdos baseados em textos, fazendo uma estruturação das informações em categorias lógicas, traçando conexões entre os conceitos e, por fim, fornecendo ao usuário acesso à informação estruturada por meio de interfaces relativamente amigáveis.

A Intelligent Miner for Text – IBM, segundo descrições da fabricante, oferece aos integradores de sistema, fornecedores de solução e desenvolvedores de aplicação uma ampla variedade de sofisticadas ferramentas de análise de texto, componentes de recuperação full-text e ferramentas de acesso a Web para enriquecer as soluções de inteligência de negócios. Pode-se construir uma grande variedade de aplicações - informações classificadas vindas de fontes de notícias; análises de portfólios de patentes, cartas de admissão de clientes e páginas Web de competidores, permitir suporte à decisão e conduzir buscas direcionadas na intranet. Muitas das ferramentas no Intelligent Miner for Text podem ser vistas como metadados sobre os documentos. Metadados são os dados estruturados que podem ser armazenados em uma base de dados e poderia ser usado objetivando a Mineração de Dados.

AlphaThemis aparece como uma ferramenta de busca e mineração de textos própria para o conhecimento jurídico, porém com a sua tecnologia flexível para a aplicação em outros campos. Inserido no processo definido de mineração de textos o aplicativo descrito inicia seus trabalhos extraíndo informações do texto

das súmulas, sejam explícitas ou implícitas, por técnicas de localização ou inferência. Segue essa atividade de forma dinâmica a cada nova entrada na base ou a partir de demandas da consulta. Executa, conseqüentemente, a mineração de dados, extraíndo estatísticas e agrupando conceitos.

Uma alternativa às ferramentas pagas seria o uso de ferramentas de domínio público, tais como MLC++ (Machine Learning in C++) [KOH97] e WEKA (Waikato Environment for Knowledge Analysis) [WEK05]. Entretanto, essas ferramentas também têm algumas características que de certa forma dificultam o trabalho de pesquisa na área.

MLC++ tem muitas facilidades, tais como interface para os principais algoritmos de aprendizado, formato padronizado para os dados de entrada, obtenção de estatísticas de desempenho e visualização gráfica das estruturas simbólicas (classificadores) obtidas por alguns algoritmos, mas trata esses classificadores como uma “caixa preta”, não fornecendo uma visão única dos classificadores simbólicos que podem ser extraídos utilizando-se a biblioteca. A incorporação de novos aplicativos à MLC++ também não é uma tarefa trivial, já que é necessário recompilar a biblioteca a cada novo aplicativo adicionado, além da dificuldade da adaptação do novo aplicativo aos padrões da MLC++. Além disso, a partir de 1995 a MLC++ passou a ser de responsabilidade da Silicon Graphics, e a última versão disponibilizada data de 1997.

WEKA, ao contrário da MLC++, reimplementa os algoritmos de aprendizado na linguagem Java. Essa abordagem padroniza as interfaces e produz código uniforme, facilitando a inclusão de novos aplicativos, mas as novas versões dos algoritmos originais podem não ser disponibilizadas no WEKA, pois exigem a sua conversão em código Java. Além disso, a recodificação de algoritmos sempre está sujeita a falhas, as quais podem causar um comportamento anômalo do algoritmo reimplementado que não ocorra no código fonte original.

Ainda outras ferramentas tais como Clasitex [CLA05], cujo projeto está em andamento, Textoscope [TEX05], Balie [BAL05] e Bow [BOW05] apresentam várias soluções e algoritmos, mas trabalham pouco com a semântica, algumas

vezes até nem a utilizam. Fazem as análises textuais levando em consideração, principalmente, a sintática do texto.

Por conta disso, essas ferramentas se mostraram ineficientes para o uso neste trabalho, pois o foco do mesmo está na utilização da semântica dentro do conteúdo de cada campo do padrão de software. Optou-se, portanto, por aprofundar-se no estudo da Análise da Semântica Latente e seus algoritmos para uma melhor abordagem na análise textual dos padrões de software.

3.4 Análise da Semântica Latente

Esta seção disserta sobre a origem e evolução da LSA, tomando em consideração as diferentes definições foram apontadas pelos cientistas desde o início da década de noventa até hoje. Para analisar a origem da LSA é necessário referir-se a um artigo escrito por Deerwester, Dumais, Furnas, Landauer e Harshman [DEE90] no qual descrevem uma nova aproximação para a indexação automática e recuperação de informação. Eles objetivavam superar um problema fundamental que se produz com as técnicas de recuperação que pretendem fazer coincidir palavras de busca com palavras em documentos (como por exemplo, nos programas de busca de informação na Internet). Eles supõem que:

"[..] There is some underlying latent semantic structure in the data that is partially obscured by the randomness of word choice with respect to retrieval" [DEE90].

A solução que propõem se origina através de um método de análise estatística. Implementa uma análise de indexação semântica latente (LSI), técnica estatística para estimar a estrutura latente. Esta técnica utiliza um valor singular de decomposição que segmenta uma grande matriz de dados de associação e permite construir um "espaço semântico" no que se associam entre si palavras e documentos. O sustento estatístico em definitivo está determinado pela co-ocorrência de palavras em a diversidade de documentos.

A partir das investigações em LSI [DEE90][FOL90], Foltz [FOL96]] define a LSA como um modo estatístico de uso de palavras que permite comparações de similaridade semântica entre termos de informação textual.

Mais tarde, Landauer e Dumais [LAN97], utilizando estes conceitos desenvolvem o que hoje se conhece como a LSA, desta vez adaptada ao campo da psicolingüística juntamente com que a recuperação de informação em geral.

Os autores definem LSA como uma nova teoria geral de aquisição e representação do conhecimento. Esta teoria baseia-se na noção de que alguns domínios de conhecimento contem imensos números de inter-relações fracas ou latentes, que se são aproveitadas, podem amplificar produzindo aprendizagem através de processos de inferência. O método de indução proposto depende da reconstrução de um sistema de relações de similaridades múltiplas em um espaço multidimensional. Supõe-se que a co-ocorrência de eventos, em particular de palavras, em contextos locais se geram e se refletem por sua similaridade em algum lugar deste espaço multidimensional. Utilizando os métodos estatísticos referidos em [DEE90], se conclui que a LSA pode usar-se para predizer fenômenos tais como a coerência textual, compreensão sem ambigüidade contextual de homógrafos e geração do significado central inferido de um parágrafo (sumário). Os cientistas definiram a Análise da Semântica Latente como uma teoria e um método para extrair e representar o significado contextual em uso de palavras através de computação estatística aplicada a um grande volume textual [LAN97]

A LSA, depois de processar um grande volume de linguagem inteligível pela máquina (digitalizado), representa as palavras usadas nesse texto, de modo que qualquer grupo destas palavras, seja uma frase ou parágrafos, é tomado do texto original como pontos em um "espaço semântico" multidimensional. O método de Análise Semântica Latente (LSA) pode ser utilizado para a construção de um espaço semântico onde os significados de palavras e textos são representados por vetores e a proximidade entre estes significados é proporcional ao ângulo entre estes vetores.

O modelo vetorial de semântica adotado pela LSA é baseado na co-ocorrência de palavras. A suposição é que palavras que tendem a ocorrer juntas dentro de um mesmo documento são consideradas como tendo alguma relação de similaridade semântica [HUA06]. A LSA foi desenvolvida primeiramente no contexto de recuperação de informações, mas recentemente, tem sido vista como uma poderosa ferramenta com um grande número de aplicações.

Algumas destas aplicações discutidas por Kintsch [KIN02] são:

- Avaliação automática de textos. A LSA pode ser utilizada para graduar textos escritos por estudantes, comparando-os com um conjunto de textos já graduados e escritos por especialistas. Em [LAN00] verifica-se que este método pode graduar textos tão bem quanto pessoas treinadas para isto.
- Seleção de textos instrucionais apropriados conforme o grau de conhecimento do aluno. Observando que materiais de instrução muito avançados ao conhecimento de um aluno seriam ineficientes para seu aprendizado e que da mesma forma materiais muito fáceis seriam de pouco valor, Wolfe [WOL98] demonstrou que a LSA poderia ser utilizada para selecionar materiais apropriados, proporcionando-lhe um melhor aprendizado.
- Graduação automática de resumos de casos clínicos. Tim Koschmann e colaboradores propõem a utilização da LSA para a avaliação de resumos de casos clínicos produzidos por estudantes de medicina, a partir de encontros com pacientes padrões [BAR93], utilizados justamente para o ensino e avaliação em escolas de medicina. Os resumos de casos preparados pelos estudantes seriam comparados com resumos já graduados por especialistas.

3.5 Conclusão

Neste capítulo, percebeu-se a abrangência das técnicas de Mineração de Dados, em especial da Análise da Semântica Latente e sua abordagem na detecção de relacionamentos entre os padrões.

Uma abordagem técnica para o tratamento de textos com mineração de dados é a Análise da Semântica Latente [KIN02]. A Análise da Semântica Latente ou LSA (Latent Semantic Analysis) é usada para definir o tema de um texto ou gerar um sumário automaticamente. A informação sobre a temática do conteúdo do texto pode ser representada por um vetor no espaço semântico: o texto provê nova informação sobre este tema e pode potencialmente modificar e expandir o espaço semântico por ele mesmo. Vetores podem, do mesmo modo, representar subseções de um texto. LSA pode ser usada para selecionar de cada subseção a sentença ou tópico mais relevante, além de gerar um tipo de sumário automaticamente.

Essa técnica pode ser muito útil no tratamento dos padrões de software para a detecção de graus de relacionamentos entre eles, pois se os campos do *template* do padrão de software forem associados entre si com seus respectivos relacionamentos pode-se retornar valores correspondentes. Uma observação interessante é que, agora, trata-se de relacionamento entre os campos do *template*, como por exemplo: *Problem - Solution*, *Context - Context Resulting* para detectar posteriormente relacionamento entre os padrões de software, propriamente ditos.

Nesse contexto, a análise da Semântica Latente se mostra como a possível solução para a análise textual dos padrões de software, dentro de uma quantidade grande de Padrões de Software. No próximo capítulo, LSA será abordada com a metodologia empregada no decorrer do trabalho.

Capítulo 4

Descoberta de Relacionamentos entre Padrões de Software

Neste trabalho, a LSA é utilizada para medir a proximidade semântica entre os diferentes campos do *template* de padrões a partir de uma coleção de documentos originalmente particionada em seções temáticas. Este capítulo, apresenta na seção 4.1 uma introdução sobre essa teoria e na seção 4.2 a metodologia utilizada na concepção deste trabalho.

4.1 Introdução

A Análise Semântica Latente (LSA) [LAN98] é um método estatístico utilizado para a construção de um espaço semântico, onde não só palavras, mas, sentenças, parágrafos, textos ou qualquer outro conjunto de palavras, podem ser representados por vetores. Também pode ser analisada sua utilização para a categorização de padrões, ou seja, a associação destes documentos a categorias pré-determinadas.

4.2 Metodologia

Uma seqüência de passos é definida para que os resultados práticos sejam atingidos. A fase inicial de construção de um espaço semântico e a decomposição SVD é característica da Análise da Semântica Latente. Os demais passos foram estudados e implementados no decorrer deste trabalho, tendo como o objetivo tratar padrões de software especificamente.

4.2.1 Construção do espaço semântico

Para construir o espaço semântico, foi utilizada uma coleção de padrões de software dos mais diversos formatos e classificações. Cada um destes padrões passa primeiramente por um pré-processamento, onde todas as letras maiúsculas foram transformadas em minúsculas e todos os caracteres que não são letras são excluídos. Feito isto, seleciona-se nesta coleção todas as palavras que aparecem ao menos em dois documentos (padrões) e que não pertenciam a uma lista de palavras chamadas *stopwords*. As *stopwords* são palavras de uso muito freqüente, como artigos, preposições e conjunções, que semanticamente não contribuem de forma relevante para um documento. A utilização destas implica apenas em um maior custo computacional para a construção do espaço semântico.

Em seguida, constrói-se uma matriz de representação da coleção acima, com as linhas correspondendo às palavras selecionadas e as colunas correspondendo aos documentos da coleção, ou seja, aos itens do *template* de cada padrão.

Inicialmente, a cada entrada desta matriz é atribuído o valor da frequência de cada palavra em cada documento.

A frequência de cada palavra, em cada uma de suas entradas na matriz é, então, transformada em seu logaritmo. Isto é feito baseando-se no fato de que um documento com, por exemplo, três ocorrências de uma mesma palavra, tende a ser mais importante do que um documento com apenas uma ocorrência, porém não três vezes mais importante. Em seguida, cada um dos novos valores de entrada é dividido pelo somatório do produto destes valores pelo logaritmo dos mesmos, para salientar a sua importância.

4.2.2 Decomposição de Valor Singular (SVD)

Através da Decomposição de Valor Singular(SVD) [FOR77], obtem-se a matriz original (M) como um produto de três matrizes:

$$M = WSP,$$

onde:

W = matriz de vetores singulares à esquerda;

S = matriz diagonal de valores singulares em ordem decrescente;

P = matriz de vetores singulares à direita.

Reduz-se, então, a dimensão destas matrizes, eliminando as linhas e colunas correspondentes aos menores valores singulares da matriz S assim como as colunas da matriz W e linhas da matriz P correspondentes.

A Decomposição de Valor Singular é normalmente utilizada para localizar a informação semântica essencial em uma matriz de co-ocorrência de palavras. Com isto, partir desta decomposição, é possível, com a redução de dimensão das matrizes W, S e P (mantendo somente os maiores valores singulares), descartar as informações acidentais que geralmente estão presentes.

Sendo assim, nosso objetivo com o produto destas três novas matrizes reduzidas, é obter um espaço semântico condensado que revelasse as melhores relações entre as palavras e os documentos.

Porém, o número de dimensões a ser reduzida de forma a otimizar o resultado é bastante questionado e parece estar bastante relacionado ao *corpus* (coleção de documentos) utilizado para a construção do espaço. Desta forma, varia-se a quantidade de dimensões reduzidas das matrizes W, S e P, obtendo os espaços semânticos correspondentes a cada redução e analisando os resultados obtidos com a utilização destes para a categorização de textos descrita mais adiante.

Em um espaço semântico, a proximidade entre duas palavras é obtida através do cálculo do cosseno do ângulo entre seus vetores (linhas da matriz) correspondentes. Quanto maior o cosseno do ângulo entre os vetores de duas palavras, maior a proximidade entre elas. O vetor de representação de um dado conjunto de palavras, como parágrafos ou textos, no espaço, pode ser obtido através do centróide (média) de todos os vetores das palavras deste conjunto. Permitindo, assim a obtenção da proximidade entre uma palavra e um texto, e até mesmo entre dois textos.

4.2.3 Detecção de Relacionamentos entre os Padrões

Dois níveis de relacionamentos são investigados neste trabalho. O nível do relacionamento entre os campos dos Padrões e o relacionamento entre os Padrões de Software, propriamente ditos.

O primeiro parte em busca de regras e associações entre os campos dos Padrões, de modo a constituir um algoritmo de alto nível para que se faça a análise do texto semanticamente dentro de cada campo. Por exemplo, tem-se conhecimento que um Padrão de Software pode apresentar uma solução cujo Contexto Resultante determine um outro tipo de problema. Neste caso, o campo *Context Resulting*, do padrão em questão, deve ser relacionado automaticamente com os campos *Problem* (ou até mesmo *Context*) de outros padrões. Nesse momento, a análise semântica textual será realizada para detectar os padrões que possuam o campo *Problem* com algo relacionado ao *Context Resulting* do primeiro padrão.

Isso se torna necessário devido às características inerentes a cada campo do padrão. Para que haja um melhor processamento da informação durante esse processo de mineração de texto, é preciso especificar de uma maneira mais clara quais os campos estão relacionados entre si.

A partir dos tipos de Relacionamentos de Padrões e tendo também como base o conhecimento do conteúdo de cada componente, é possível descrever um algoritmo de alto nível em que os componentes do padrão (itens do *template*) são similares ou complementares como, por exemplo, numa relação de causa e consequência entre si.

Esse tipo de relacionamento é transparente para o usuário da aplicação implementada, cuja necessidade é de conhecer os Padrões relacionados e não a existência de relacionamento entre os campos desses Padrões.

Por isso, o outro nível de Relacionamento é o que será retornado ao final da análise textual com semântica. Esses tipos de relacionamentos foram abordados na Seção 2.4.

4.2.4 Correlação texto a texto

Após a decomposição SVD, uma matriz Y pode ser obtida. Essa matriz Y apresenta uma coleção de palavras nas linhas pelos campos de cada *template* de cada padrão da coleção nas colunas. Os números retornados representam um grau de afinidade entre as palavras e os itens do *template* dos padrões.

Dessa forma, pode-se perceber quando uma palavra tem muita semelhança com um determinado campo ou com outro, pois será de acordo com o grau retornado na matriz. Isso demonstrará o relacionamento entre os textos dentro daquelas seções.

A partir dessa matriz Y , chega-se a uma tabela com os índices de correlação texto a texto. No trabalho em questão, trata-se de uma tabela com linhas e colunas representadas por cada campo de cada padrão da coleção inicial e um grau de afinidade entre eles.

Dependendo do que se trata cada campo ali relacionado, os graus de relacionamentos entre os padrões já podem ser detectados a partir de uma série de regras geradas.

4.2.5 Regras de associação de relacionamento

Para que algumas regras possam ser implementadas a fim de detectar relacionamentos consistentes entre padrões, foi necessária uma análise profunda a respeito dos relacionamentos existentes entre os mesmos, como demonstrada na seção 2.4.

A partir daí e tendo como base para análise a tabela de correlação texto a texto, algumas regras podem, enfim, ser aplicadas com o objetivo de retornar um grau de afinidade entre padrões, dois a dois.

Depois de encontrar os graus de relacionamento entre as seções dos padrões, é preciso verificar do que se trata cada seção, ou seja, se é a seção *Problem*, *Solution*, *Applicability* e também é preciso saber de qual padrão aquela seção se refere.

Nesse momento, a associação de relacionamento entre padrões dá-se início com um par $(x,y) \rightarrow$ que são os padrões entre os quais irá se encontrar um nível de relacionamento.

Sejam X e Y padrões de software, os tipos de relacionamentos existentes entre eles são:

I. X é similar a Y

Quando esse tipo de relacionamento ocorre, é possível que se encontre relacionamento entre os seguintes campos dos *templates* dos padrões X e Y respectivamente:

- Problem[X] \leftrightarrow Problem[Y]
- Forces[X] \leftrightarrow Forces[Y]
- Context[X] \leftrightarrow Context[Y]
- Intention[X] \leftrightarrow Intention[Y]
- Abstract[X] | Description[X] \leftrightarrow Abstract[Y] | Description[Y]
- Applicability[X] \leftrightarrow Forces[Y] | Problem[Y] | Context[Y] | Intention[Y]*

(*Explicação: Applicability com qualquer um do outro lado e o sentido inverso tbm)

- Name[X] | As Known as[X] | Aliases[X] \leftrightarrow Name[Y] | As Known as[Y] | Aliases[Y]
- Classification[X] \leftrightarrow Classification[Y]

II. X usa Y na sua solução

Quando esse tipo de relacionamento ocorre, é possível que se encontre relacionamento entre as seguintes seções dos padrões X e Y respectivamente:

- Solution[X] \leftrightarrow Solution[Y]
- Solution[X] \leftrightarrow Implementation[Y]
- Solution[X] \leftrightarrow Name[Y] | As Known as[X] | Aliases[X]

III. X pode ser combinado a Y

Quando esse tipo de relacionamento ocorre, é possível que se encontre relacionamento entre as seguintes seções dos padrões X e Y respectivamente:

- Resulting Context[X] | Consequences[X] \leftrightarrow Context[Y] | Problem[Y] | Forces[Y] | Applicability[Y] | Intention[Y]

Esses relacionamentos estão em ordem decrescente de força, ou seja, o relacionamento I é mais forte que o II que, por sua vez, é mais forte que o III. Isso revela que, quando dois padrões apresentam ocorrências mais freqüentes de relacionamento do tipo I, pode haver maior afinidade entre eles do que se existisse apenas relacionamentos do tipo III. Por isso, diferentes graus são atribuídos às comparações observadas a partir desses relacionamentos.

4.2.6 Cálculo do grau de relacionamento

Para efeito de cálculo, verifica-se na tabela resultante final em cada coluna quais seriam os maiores graus de relacionamento, ou seja, os números maiores ou iguais a 0,5, pois verificou-se empiricamente este ser um bom valor de corte, uma vez que os graus de afinidade podem estar definidos entre 0 e 1. Depois, localiza-se quais os outros itens com quem aquele determinado campo do padrão demonstrado na coluna obteve esse alto grau de afinidade.

Se os campo detectados forem originados de padrões diferentes, forma-se o par de padrões (X,Y) e o processo do cálculo do grau de relacionamento entre eles é iniciado.

A cada relacionamento desse detectado, atribui-se o grau correspondente, calculando, no final, uma média aritmética para definir quão relacionados os padrões estariam.

Para o relacionamento do tipo I, é multiplicado o número 1 pelo grau retornado na tabela. Para o relacionamento do tipo II é multiplicado o valor de 0,75 e, por fim, para o relacionamento do tipo III multiplica-se o grau de afinidade entre os textos por 0,5. Esses valores foram observados como os mais adequados para a atribuição de graus aos relacionamentos dos padrões, de acordo com a força que esse relacionamento transmite.

Tabela de correlação texto a texto:

	Campo1Padrã oX	Campo2Padrã oX	Campo3Padrã oX	...	Campo1Padrã oY	Campo2Padrã oY
Campo2Padrã oX	0,91					
Campo3Padrã oX	0,52	0,36				
...			
Campo2Padrã oY	0,88	0,68	0,74	...	0,45	
Campo3Padrã oY	0,12	0,25	0,85	...	0,92	0,69

Tabela 01. Correlação texto a texto

Percebe-se já na primeira coluna graus maiores a 0,5 associados ao campo 1 do Padrão X. Dessa forma, procura-se identificar quais desses itens correspondem a um outro padrão que não seja o padrão X.

No exemplo proposto, são detectadas quatro ocorrências de relacionamentos entre o par de padrões (X,Y) a seguir:

- 0,88 (supondo um relacionamento tipo I)
- 0,68 (relacionamento tipo III)
- 0,74 (relacionamento tipo II)
- 0,85 (relacionamento tipo I)

Cálculo de relacionamento entre os padrões (X,Y):

$$= (0,88*1 + 0,68*0,5 + 0,74*0,75 + 0,85*1) = 0,65625$$

4.3 Conclusão

A partir de um grande conjunto de padrões de software, é possível capturá-los em forma de texto e empregar o processo descrito neste capítulo, de modo que os resultados sejam semanticamente adequados com o esperado, ou seja, um grau de afinidade entre dois textos submetidos ao processo é informado. Isso foi possível devido às características próprias da LSA, que trabalha com métodos estatísticos precisos demonstrados no decorrer desta dissertação.

Uma das vantagens dessa utilização é a automatização do processo de detecção de relacionamentos e o fato do processo ser facilmente expansível, pois a sua entrada é um texto simples.

Capítulo 5

Conclusão

Este trabalho apresentou uma maneira de se descobrir relacionamentos entre padrões de software utilizando uma técnica de Mineração de Dados e Texto, a Análise da Semântica Latente. A partir das pesquisas realizadas nesse trabalho, pode-se verificar a existência de uma grande variedade de perspectivas de novas pesquisas complementares. Os resultados mais evidentes são discutidos na seção 5.1 e algumas propostas de estudos complementares são detalhadas na seção 5.2.

5.1 Resultados

Desenvolver métodos que implementem um mecanismo eficiente para detectar relacionamentos e afinidades entre textos, mais precisamente entre padrões de software, não é apenas viável, como foi mostrado com esse embasamento teórico e com a implementação em Java no Anexo A, como também aumenta a confiança de que usar a LSA como uma forma para o processo de descoberta através de métodos estatísticos pode oferecer uma efetiva contribuição para o desenvolvimento de ferramentas de busca e armazenamento de padrões de software, cuja necessidade se faz tão patente.

Essa realidade oferece ao desenvolvedor um alto grau de flexibilidade para escolher, dentre um conjunto de padrões, o mais apropriado padrão de software, bem como os padrões relacionados ao mesmo, para cada etapa do processo de desenvolvimento do software.

5.2 Trabalhos Futuros

Após elaboradas as regras e algoritmos para a execução da Mineração Textual dos Padrões do Software, faz-se necessária uma integração com um Repositório existente, como o proposto em [AND03].

A escolha da linguagem Java foi feita de acordo com uma tendência do uso desta linguagem pela vantagem de ser multiplataforma, permitindo deste modo a construção de aplicações inteligentes portáteis. Além disto, destaca-se pela integração com a Internet, facilitando a integração do Repositório de Padrões.

A integração com qualquer repositório de padrões dá-se de maneira simples, uma vez que a aplicação implementada recebe o padrão no formato de texto simples.

Dentro de um Repositório, a análise da semântica latente pode ser adaptada para obter outras informações, como uma classificação automática, sumarização dos padrões e até mesmo uma busca mais eficaz.

Capítulo 6

Referências Bibliográficas

[ADRI97] Adriaans, P., Zantinge, D., Data Mining. Addison-Wesley, 1996.

[AGRA95] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo: Fast Discovery of Association Rules, In Knowledge Discovery in Databases, Volume II, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), AAAI/MIT Press, 1995.

[ALE77] ALEXANDER, C. et al. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, NY, 1977.

[ALE79] ALEXANDER, C. The Timeless Way of Building. Oxford University Press, New York, NY, 1979.

[ANA05] ANANYAN, Sergei. Text Mining Applications and technologies. Disponível em: <http://www.megaputer.com/down/tm/Text_Mining.pps>. Acesso em: 14/07/2005.

[AND01] ANDRADE, R. *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems*. Ph.D. Thesis, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, May 2001.

[AND03] ANDRADE, R.; MARINHO, F.; SANTOS, M.; NOGUEIRA, R. Uma Proposta de um Repositório de Padrões Integrado ao RUP. Session Pattern Application (SPA), SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, Ago. 2003.

[APP00] APPLETON, B. Patterns and Software: Essential Concepts and Terminology. <http://www.enteract.com/~bradapp/>

[BAL05] Balie <http://balie.sourceforge.net> Acesso em 15/07/2005

[BAR93] BARROWS, H.S., "An Overview of the Uses of Standardized Patients for Teaching and Evaluation Clinical Skills". *Acad Med*, v. 68, p. 443-451, 1993.

[BEC97] BECK, K. *Smalltalk Best Practice Patterns*. Prentice Hall, Upper Saddle River, NJ, 1997.

[BOW05] BOW <http://www.cs.cmu.edu/~mccallum/bow> Acesso em 15/07/2005

[BUS96] BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. *Pattern-Oriented Software Architecture*. John Wiley and Sons, New York, NY, 1996.

[CHA00] Chan, S. W. K., Lai, T. B. Y., Gao, W. J. and T'sou, B. K. Mining Discourse Markers for Chinese Textual Summarization. In: *Proceedings of the ANLP/NAACL Automatic Summarization Workshop*, pp. 11-20. Seattle, Washington, 2000.

[CLA05] Clasitex <http://dlsi.ua.es/projectes/projecte.cgi?id=val&projecte=37>
Acesso em 15/07/2005

[COA95] COAD, P.; North, D.; Mayfield, M. *Object Models: Strategies, Patterns and Applications*, Yourdon Press, 1995.

[COP00] COPLIEN, J. O. *C++ Idioms Patterns*. In Brian Foote, Neil Harrison, and Hans Rohnert, editors, *Pattern Languages of Program Design 4*, chapter 10, 167-197. Addison Wesley, Reading, MA, 2000.

[COP96] COPLIEN, J. O. *Software Patterns*. SIGS books and Multimedia, June 1996.

[COR05] Core J2EE Pattern Catalog. Disponível em:
<http://java.sun.com/blueprints/corej2eepatterns>. Acessado em: 13/06/2005.

[CRI05] CRISP-DM <http://www.crisp-dm.org/> Acessado em: 15/09/2005.

[DEE90] DEERWESTER, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., e Harshman, R. Indexing by Latent Semantic Analysis. *Journal of the American Society For Information Science*, 41, 391-407, 1990.

[DIE06] The Diemen Repository of Interaction Design Patterns, Disponível em:

<<http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/>>. Acessado em: 14/06/2006.

[DRO99] DRONKERS, N. Language, Neural Basis of. em R.A. Wilson & F.C.Keil (Eds), The MIT encyclopedia of the Cognitive Science. 448-451. Cambridge, MA: MIT Press, 1999.

[FAY96] Fayyad, U. M., Piatetsky Shapiro, G., Smyth, P. & Uthurusamy, R. – “Advances in Knowledge Discovery and Data Mining”, AAAIPress, The Mit Press, 1996.

[FOL90] FOLTZ, P. W. Using Latent Semantic Indexing for Information Filtering. em R. B. Allen (Ed.) Proceedings of the Conference on Office Information Systems, 40-47. Cambridge, MA,: MIT Press, 1990.

[FOL96] FOLTZ, P. W. Latent Semantic Analysis for text-based research. *Behavior Research Methods, Instruments and Computers*. 28(2), 197-202, 1996.

[FOL98] FOLTZ, P, Kintsch, W.y Landauer, Th. The Measurement of Textual Coherence with Latent Semantic Analysis. *Discourse Processes*, 25(2&3), 285-307, 1998.

[FOR77] FORSYTHE, G.E., Malcom, M.A., Moler, C.B., Computer Methods for Mathematical Computations, New Jersey: Prentice Hall, 1977.

[FOW97] FOWLER, M. Analysis Patterns: Reusable Object Models. Menlo Park, CA: Addison Wesley Longman, 1997.

[FRA82] Francis, W. e Kucera, H.. Frequency Analysis of English Usage, Houghton Mifflin, 1982.

[GAM95] GAMMA, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1995.

[GRA01] GRAESSER, A., Person, N., Harter, D., e TRG. Teaching tactics and dialog in AutoTutor. *International Journal of Artificial Intelligence in Education*, 2001.

[HEA99] Hearst, M. A. Untangling Text Data Mining. In: Proceedings of ACL´99, the 37th Annual Meeting of the ACL, University of Maryland, USA. 1999.

[HIL05] *The Hillside Group*. Disponível em <http://www.hillside.net>. Acessado em: 01/07/2005.

[HUA06] HUANG, Y., "Support Vector Machines for Text Categorization Based on Latent Semantic Indexing", Artigo obtido do site <http://bach.ece.jhu.edu/gert/courses/774/2001/lisa.pdf>, última visita em 01/03/2006.

[JON88] JOHNSON, R.; FOOTE, B. Designing Reusable Classes. *Journal of Object-Oriented Programming*. SIGS, 1, 5 (June/July. 1988), 22-35. Disponível em: <http://www.lifia.info.unlp.edu.ar/poo2001/DRC.pdf>. Acessado em: 01/12/2005.

[KIN00] KINTSCH, W. A computational theory of metaphor comprehension. *Psychonomic Bulletin & Review*, 7,257266, 2000.

[KIN01] KINTSCH, W. Predication. *Cognitive Science*. 25, 173202. 2001.

[KIN02] KINTSCH, W., "The Potential of Latent Semantic Analysis for Machine Grading of Clinical Case Summaries", *Journal of Biomedical Informatics*, v. 35, p. 3-7.

[KIN02] KINTSCH, W. On the notions of theme and topic in psychological process models of text comprehension. Department of Psychology - University of Colorado, 2002.

[KIN98] KINTSCH, W. *Comprehension: A paradigm for cognition*. New York : Cambridge University Press, 1998.

[KON02] KONRAD, S.; CHENG, B. H. C. Requirements patterns for embedded systems. In Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE02), Essen, Germany, September 2002.

[KRU99] KRUCHTEN, P. *The Rational Unified Process, An Introduction*. Addison-Wesley, 1999.

[LAN00] LANDAUER, T.K. , Laham, D., Foltz, P.W., "The Intelligent Essay Assessor", *IEEE Intel Syst* 2000, sep/oct, p. 27-31.

[LAN02] LANDAUER, Th. On the computational basis of learning and cognition: Arguments from LSA. em Ross, N. (Ed.), *The psychology of learning and motivation*, 41, 1-63. 2002.

[LAN97] LANDAUER, Th., e Dumais, S. A solution to Platos problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104, 211-240. 1997.

[LAN98] LANDAUER, T.K., Foltz, P.W., Laham, D. (1998), "Introduction to Latent Semantic Analysis", *Discourse Processes*, v. 25, 259-284.

[LAN99] LANDAUER, Th. Latent Semantic Analysis: A Theory of the Psychology of language an Mind. *Discourse Processes*, 27(3), 303-310. (1999)

[MAR98] MARTIN, R.C.; Riehle, D.; Buschmann, F. (eds.) *Pattern Languages of Program Design 3*, Reading-MA, Addison-Wesley, 1998.

[MEG05] Megaputer – White Papers. <http://www.megaputer.com/tech/wp/tm.php3>
Acesso em: 16/09/2005.

[MIC05] Microsoft - patterns & practices
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/MSpatterns.asp> Acessado em: 10/09/2005.

[MIT03] MITRA, S. e ARYA, T. *Data Mining: multimedia, soft computing, and bioinformatics*. Wiley-Interscience, 2003.

[MIT97] Mitra, M., Singhal, A. and Buckley, C. (1997). Automatic Text Summarization by Paragraph Extraction. In: I. Mani and M. Maybury (eds.) *Intelligent Scalable Text Summarization ACL 1997 Workshop*, pp. 39-46. Madrid, Spain.

[NOR99] NORRIS, D. Computacional psycholinguistics. em R.A. Wilson e F.C. Keil, (Eds.) *The Mit encyclopedia of the cognitive sciences*. 168-170). Cambridge, Ma: MIT Press. 1999.

[PAT05] Patternshare <http://patternshare.org/>

[PER98] PERFETTI, Ch. The limits of Co-Occurrence: Tools and Theories in Language Research. *Discourse Processes*, 25(2&3), 365-377. 1998.

[PHP06] phpPatterns. Disponível em: <http://www.phppatterns.com>. Acessado em: 14/06/2006.

[POR06] Portland Patterns Repository. Disponível em: <http://c2.com/ppr/index.html>. Acessado em: 14/06/2006.

[PRE95] PREE, W. Design Patterns for Object-Oriented Software Development. Reading, MA: Addison Wesley, 1995.

[REH98] REHDER, B.; Schreiner, M. E.; Wolfe, M. B.; Laham, D.; Landauer, T. K. e Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes*, 25, 337-354. 1998

[RIS00] RISING, L. The Pattern Almanac 2000. Software Pattern Series, Addison-Wesley, 2000. ISBN 0-201-61567-3.

[SEC06] SecurityPatterns.org. Disponível em: <http://www.securitypatterns.org>. Acessado em: 14/06/2006.

[SEM05] SEMIO. Semio Products and Solutions Overview. Disponível em: <<http://www.semio.com/products/overview.html>>. Acesso em: 30/06/2005

[SIN87] Sinclair, J., ed. (1987). Looking Up: An Account of the COBUILD Project in Lexical Computing. Collins.

[TEX05] Textoscope <http://citeseer.ist.psu.edu/feldman98text.html> Acesso em 15/07/2005

[VEG02] DE VEGA, M. "Compreensão do discurso: representação do significado e métodos de investigação", Seminario dictado em Universidad de Concepción, Chile, 2002.

[VEL05] VELLASCO, Marley. Aurélio, Marco. Lopes, C. Henrique. Descoberta de Conhecimento e Mineração de Dados – Apostila, ICA – Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica, PUC–Rio

[VLI97] VLISSIDES, J. *Patterns: The Top Ten Misconceptions*. Object Magazine, Mar, 1997. Disponível em: <<http://hillside.net/patterns/papersbibliographys.htm>>. Acesso em: 20/07/2005

[WAI05] TEXT MINING RESEARCH GROUP AT THE UNIVERSITY OF WAIKATO. Text Mining. Disponível em: <<http://www.cs.waikato.ac.nz/~nzdl/textmining/>>. Acesso em: 30/08/2005.

[WEK05] Ferramenta Weka <http://www.cs.waikato.ac.nz/ml/weka/>. Acessado em: 11/03/2006

[WEL06] WELIE, M. V. Amsterdam Patterns Collection. Disponível em: <http://www.welie.com/patterns>. Acessado em: 14/06/2006.

[WOH98] WOHL, Amy D. Intelligent Text Mining Creates Business Intelligence. In: IBM Business Intelligence Solutions CD. EUA: 1998.

[WOL98] WOLFE, M. B.; Schreiner, M. E.; Rehder, B.; Laham, D.; Foltz, P. W.; Kintsch, W. e Landauer, T. K. Learning from text: Matching readers and text by Latent Semantic Analysis. *Discourse Processes*, 25, 309-336, 1998.

[XML06] XMLPatterns.com. Disponível em: <http://www.xmlpatterns.com>. Acessado em: 14/06/2006.

[ZIM95] ZIMMER, W. *Relationships Between Design Patterns*. Pattern Languages of Program Design, Addison-Wesley, 1995.

Apêndice A

Implementação LSA

Main.java

```

import java.io.*;
import java.util.HashSet;
import Jama.*;

public class Main{
    public static void main(String args[]) throws
IOException{
        Texto t1 = new Texto(args[0]);
        Texto t2 = new Texto(args[1]);

        HashSet set =
(HashSet)((Object)(t1.getWords()).clone());
        /*Adiciona os elementos de t2*/
        Object aux[] = (t2.getWords()).toArray();
        for(int i = 0;i < aux.length;i++)
            set.add((String)aux[i]);

        double m[][] = new
double[set.size()][t1.getNumSecoes() + t2.getNumSecoes()];
        int n1 = t1.getNumSecoes();
        int n2 = t2.getNumSecoes();
        /*Preenche matriz com valores certos*/
        aux = set.toArray();
        for(int i = 0;i < aux.length;i++){
            for(int j = 0;j < n1;j++){
                m[i][j] = t1.getValue((String)aux[i], j);
            }
            for(int j = 0;j < n2;j++){
                m[i][n1 + j] =
t2.getValue((String)aux[i], j);
            }

            System.out.println("Matriz de relações
palavras/seções:");
            print(new Matrix(m));

            Matrix ma = new Matrix(m);

            SingularValueDecomposition svd = new
SingularValueDecomposition(ma);

```

```

Matrix u = svd.getU();

Matrix s = svd.getS();

Matrix v = svd.getV();

System.out.println("\nDecomposição SVD:\nMatris
S:");

print(u);
System.out.println("Vetor V:");
print(s);
System.out.println("Vetor D:");
print(v);

System.out.println("\nMatriz inicial recomposta com
novos valores:");
ma = multiplica(u, s);
ma = multiplica(ma, v);
print(ma);
}

public static void print(Matrix ma){
    double m[][] = ma.getArray();

    int mi = ma.getRowDimension();
    int mj = ma.getColumnDimension();

    for(int i = 0;i < mi;i++){
        for(int j = 0;j < mj;j++){
            System.out.print(m[i][j] + " ");
            System.out.println("");
        }
        System.out.println("");
    }

    public static Matrix multiplica(Matrix ma, Matrix mb){
        double a[][] = ma.toArray();
        double b[][] = mb.toArray();
        double c[][] = new
double[ma.getRowDimension()][mb.getColumnDimension()];
        int tamx = ma.getRowDimension();
        int tamy = mb.getColumnDimension();
        int meio = ma.getColumnDimension();

        for(int i = 0;i < tamx;i++)
            for(int j = 0;j < tamy;j++)
                for(int k = 0;k < meio;k++)

```

```

        c[i][j] += a[i][k]*b[k][j];

        return new Matrix(c);
    }
}

```

Matrix.java

```

import java.io.*;
import java.util.HashSet;
import Jama.*;

public class Main{
    public static void main(String args[]) throws
    IOException{
        Texto t1 = new Texto(args[0]);
        Texto t2 = new Texto(args[1]);

        HashSet set =
        (HashSet)((Object)(t1.getWords()).clone());
        /*Adiciona os elementos de t2*/
        Object aux[] = (t2.getWords()).toArray();
        for(int i = 0;i < aux.length;i++){
            set.add((String)aux[i]);
        }

        double m[][] = new
        double[set.size()][t1.getNumSecoes() + t2.getNumSecoes()];
        int n1 = t1.getNumSecoes();
        int n2 = t2.getNumSecoes();
        /*Preenche matriz com valores certos*/
        aux = set.toArray();
        for(int i = 0;i < aux.length;i++){
            for(int j = 0;j < n1;j++){
                m[i][j] = t1.getValue((String)aux[i], j);
            }
            for(int j = 0;j < n2;j++){
                m[i][n1 + j] =
                t2.getValue((String)aux[i], j);
            }
        }

        System.out.println("Matriz de relações
        palavras/seções:");
        print(new Matrix(m));

        Matrix ma = new Matrix(m);
    }
}

```

```

        SingularValueDecomposition svd = new
SingularValueDecomposition(ma);

        Matrix u = svd.getU();

        Matrix s = svd.getS();

        Matrix v = svd.getV();

        System.out.println("\nDecomposição SVD:\nMatris
S:");
        print(u);
        System.out.println("Vetor V:");
        print(s);
        System.out.println("Vetor D:");
        print(v);

        System.out.println("\nMatriz inicial recomposta com
novos valores:");
        ma = multiplica(u, s);
        ma = multiplica(ma, v);
        print(ma);
    }

    public static void print(Matrix ma){
        double m[][] = ma.getArray();

        int mi = ma.getRowDimension();
        int mj = ma.getColumnDimension();

        for(int i = 0;i < mi;i++){
            for(int j = 0;j < mj;j++){
                System.out.print(m[i][j] + " ");
                System.out.println("");
            }
            System.out.println("");
        }

        public static Matrix multiplica(Matrix ma, Matrix mb){
            double a[][] = ma.toArray();
            double b[][] = mb.toArray();
            double c[][] = new
double[ma.getRowDimension()][mb.getColumnDimension()];
            int tamx = ma.getRowDimension();
            int tamy = mb.getColumnDimension();
            int meio = ma.getColumnDimension();

```

```
for(int i = 0;i < tamx;i++)
    for(int j = 0;j < tamy;j++)
        for(int k = 0;k < meio;k++)
            c[i][j] += a[i][k]*b[k][j];

return new Matrix(c);
}
}
```