UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Kuesley Fernandes do Nascimento

USO DE ONTOLOGIAS PARA DETECÇÃO DE PADRÕES DE ANÁLISE EM MODELOS CONCEITUAIS EM BIBLIOTECAS DIGITAIS DE COMPONENTES

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Raul Sidnei Wazlawick

Orientador

Florianópolis, Maio de 2008.

USO DE ONTOLOGIAS PARA DETECÇÃO DE PADRÕES DE ANÁLISE EM MODELOS CONCEITUAIS EM BIBLIOTECAS DIGITAIS DE COMPONENTES

Kuesley Fernandes do Nascimento

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação - Área de Concentração de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Dr. Mário Antônio Ribeiro Dantas
Coordenador do PPGCC

Banca Examinadora:

_____

Dr. Raul Sidnei Wazlawick
Orientador

_____

Dra. Renata Vieira

_____

Dr. Frank Augusto Siqueira

_____

Dr. Fernando Álvaro Ostuni Gauthier

“Os verdadeiros analfabetos são os que aprenderam a ler e não lêem”

(Mário Quintana)

Dedico este trabalho aos meus pais, Bento e Fátima e aos meus irmãos, Keyla e Keynes.
Dedico-o também à Priscila, minha esposa.

Agradecimentos

A **Deus** por permitir com saúde a conclusão deste trabalho.

À minha esposa **Priscila,** que me auxiliou na revisão deste trabalho, e pela compreensão quanto aos incontáveis momentos passados na companhia do computador.

Ao meu pai **Bento,** que na sua simplicidade sempre me incentivou a valorizar a ciência e buscá-la sempre que possível.

À minha mãezinha **Fátima**, sem dúvida, a mulher mais importante na minha vida.

Aos meus irmãos **Keyla** e **Keynes** que com muito bom humor e amizade estiveram comigo nos momentos difíceis. Enfim, a todos da minha família, em especial àqueles que ajudaram na minha formação pessoal e profissional: minha vó **Eliza**, meu tio **Antonio** e meu primo **Rômulo**;

Em especial ao meu professor, mestre e orientador **Dr. Raul Sidnei Wazlawick** que acreditou em mim ao conceder esta oportunidade.

Ao catedrático professor **Antonio Carlos Mariani** que me fez pensar e criticar este trabalho tantas vezes...

Às pessoas da secretaria do PPGCC, principalmente **Vera Lúcia Sodré Teixeira (Verinha)** a pessoa mais concentrada que já conheci.

Aos amigos que conheci durante o mestrado e me ajudaram a espairecer nos momentos certos: **Everton** e **Juliana** pelo bom humor e companhia; **Luiz** e **Tania** pelas risadas; e, **Rafael** pela paciência e pela estatística;

Aos amigos que sempre estiveram comigo: **Gastão**, **Natan**, **Ezequiel, Mauro e Julião**, ótimos profissionais e muitas idéias trocadas;

Ao amigo **Braga** pela ajuda nas figuras.

A todos meu sincero obrigado, pois cada um, da sua maneira, me ajudou a chegar até aqui!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

Apresenta-se neste trabalho um método de detecção de padrões de analise (PA's) em modelos conceituais utilizando ontologias. Um PA pode ter sido previsto ou não no momento em que o modelo conceitual foi concebido. Mesmo se a análise do sistema (fase onde surge o modelo conceitual) não for orientada pelos padrões de análise, é possível verificar a ocorrências destes dentro dos modelos produzidos. Esta ocorrência se dá a partir de algumas regras que são observadas e apresentadas neste trabalho.

Para detectar PA em modelos conceituais o artefato essencial integrante deste método é uma ontologia. A ontologia como ferramenta para representar conhecimento tem como papel no CompogeMatch (método apresentado neste trabalho) identificar os conceitos existentes nos modelos submetidos ao método.

Uma vez detectados os PAs existentes nos modelos, é possível criar índices a partir desses PA's encontrados e utilizá-los como filtros indexados no processo de recuperação em bibliotecas digitais de componentes ou modelos conceituais de software. Uma alternativa às buscas por meio de palavras-chaves que apresentam algumas limitações, como por exemplo, não identificação de palavras sinônimas.

Por fim, esta pesquisa indica como esse processo de busca pode trazer resultados superiores à busca por palavras-chaves quando o que está se procurando são modelos conceituais ou, mais precisamente, software.

Palavras-chave: engenharia de software, modelos conceituais, padrões de análise, ontologias, recuperação de informação.

ABSTRACT

This research presents a method for analysis patterns detection (AP's) using ontology in conceptual models. An AP may have been predicted or not at the generation moment of the conceptual model. Even if the system analysis (phase where the conceptual model is created) is not guided by analysis patterns, it is possible to discover their occurrences within the models that are produced. Those occurrences rely on some rules hat have been studied and presented in this research.

Ontology is a fundamental artifact to detect an AP in conceptual models. The ontology used to represent knowledge provides meaning to the existent concepts in the models submitted to the method CompogeMatch (the method proposed in the research).

Since the existent APs in the models are detected, it is possible to create indexes from these APs and use them as indexing filters for recovering processes in digital libraries of components or conceptual models of software. It is an alternative to simple searching through keywords that presents some limitations, as for instance, lack of synonyms identification.

Finally, this research indicates how this search process may bring better results than keywords search when what is seek for is a conceptual models or, more specifically, software.

Keywords: software engineering, conceptual models, analysis patterns, ontologies, information retrieval.

1. INTRODUÇÃO

O processo de busca e recuperação de informações é atividade presente em muitas áreas de conhecimento. Procurar por um script, um artigo científico, uma foto ou até mesmo uma imagem médica é uma atividade freqüente na vida dos usuários da Internet. É freqüente, também, a busca por aplicativos (software), sendo os mais procurados na rede: antivírus, ferramentas de áudio e vídeo, bem como ferramentas de manutenção para computadores. Tais buscas normalmente são executadas passando como filtro o próprio nome do software. Assim, pode-se dizer que se trata de uma busca de complexidade simples, pois se resume a encontrar um servidor (ftp ou http) que disponibiliza o software através de ferramentas de buscas (Google, Yahoo, Baixaki, Superdownloads). Essa busca torna-se complexa quando não se tem, ou não se sabe, o nome do software que se deseja procurar.

Encontrar sistemas ou componentes de software e medir similaridade entre eles é objeto de estudo da ciência da computação. Uma das formas de se conseguir isso consiste em utilizar os documentos de texto, que descrevem os sistemas e seus requisitos, juntamente com processamento de linguagem natural (GIRARDI & IBRAHIM, 1995) (MAAREK, BERRY e KAISER, 1994). Outra forma de descrever características (aspectos estáticos) de um sistema é o modelo conceitual.

O modelo conceitual descreve as informações que o sistema manipula. Este pode ser escrito em uma linguagem de modelagem, como por exemplo a UML, onde são elencados os conceitos do sistema em forma de diagrama.

Uma característica peculiar do modelo conceitual é a ocorrência de padrões de análise (FOWLER, 1997). Um padrão de análise (PA) consiste em uma solução conhecida que é dada a um problema recorrente. A utilização de padrões é bastante difundida na área de projeto e implementação de código (GAMMA & HELM et al, 1993).

Este trabalho descreve um método para identificar a ocorrência dos padrões de análise em modelos conceituais. Para isso, utiliza-se uma ontologia para mapear os nomes

dos conceitos existentes em modelos conceituais. Demonstra-se, dessa forma, como é possível utilizar PA's como índices de busca e recuperação de componentes de software em bibliotecas digitais. Discute-se também como os PA's podem ser utilizados como medida de similaridade entre sistemas e componentes de software.

A vantagem de procurar modelos conceituais a partir de PA's é a independência de especificidade de domínio, uma vez que padrões são genéricos e alheios ao assunto específico do sistema. Assim, procurar por modelos conceituais que têm um determinado PA pode melhorar o processo de recuperação, alcançando mais registros na busca.

1.1 APRESENTAÇÃO DO TRABALHO

Um sistema de informação pode ser descrito de diferentes formas, tais como:

a) Em linguagem natural (por exemplo, requisitos funcionais e casos de uso).
b) Em uma linguagem de programação.
c) Por meio de diagramas UML (por exemplo, diagrama de casos de uso, modelo conceitual, ou um diagrama de seqüência).

Independentemente de como estão feitas, sistemas de busca de software precisam interpretar essas descrições (um diagrama ou um texto), e compará-las para apontar se um componente de software, ou até mesmo um sistema, se assemelha a outro.

O modelo conceitual é um artefato que descreve as entidades que o sistema gerencia ou manipula (WAZLAWICK, 2004). É nele que está representada a solução investigada para o problema (o sistema propriamente dito) (LARMAN, 2001). Este trabalho apresenta um método de leitura do modelo conceitual (descrição), e, a partir dele, aponta os padrões de análise (FOWLER, 1997) encontrados. A partir disso, é possível comparar e apontar o quão similares são dois modelos conceituais e, conseqüentemente componentes de software e sistemas.

1.2 JUSTIFICATIVA

Encontrar um software que atenda às necessidades de uma empresa ou entidade governamental não é uma tarefa trivial. Sistemas de buscas (GIRARDI & IBRAHIM, 1995) (MAAREK, BERRY e KAISER, 1994) baseiam-se em palavras-chave ou descrições textuais. Mas essa técnica tem limitações, uma vez que seus textos são descritos em linguagem natural, susceptível à ambigüidade e incompletude.

O modelo conceitual é, no entanto, uma descrição estruturada da informação que o sistema manipula, usualmente bem maior do que simples descrições textuais ou palavras-chave.

Porém, os nomes de conceitos e atributos são dados de forma livre, pelas pessoas que concebem os sistemas. Desta forma, o uso de ontologias para comparar conceitos e estruturas conceituais apresenta-se como uma possibilidade de pesquisa para a construção de sistemas de busca de software mais eficazes.

1.3 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é apresentar uma técnica de como detectar padrões de análise em modelos conceituais de software utilizando uma ontologia. A partir dessa detecção, pode-se recuperar componentes de software em bibliotecas digitais utilizando como índice os padrões de análise.

Este trabalho destaca os seguintes objetivos específicos:

a) Definir uma linguagem de representação dos modelos conceituais e da ontologia utilizada no mecanismo de busca.

b) Definir os padrões de análise que podem ser detectados dentro dos modelos conceituais e como se dará a extensibilidade do modelo de busca.

c) Definir os critérios de detecção dos padrões de análise.

Os objetivos específicos dividem em problemas menores o objetivo principal para faciliar o alcance deste.

1.4 METODOLOGIA

Este trabalho apresenta uma revisão bibliográfica sobre os assuntos inerentes à recuperação de informação, mais especificamente de software. Nesta revisão são apresentados os elementos envolvidos no processo de recuperação e os métodos de busca de software, explorando, ainda, a ocorrência dos padrões de análise (FOWLER, 1997) em modelos conceituais. Nesse estudo, são identificados os métodos de BUCHLI e NIERSTRASZ (2003), MISHNE e RIJKE (2004) e MONTES-Y-GÓMEZ, LÓPEZ-LÓPEZ e GELBUKH (2000) para detecção de padrões e recuperação de componentes de software.

Levantados os métodos de busca e apontado seus pontos fortes e pontos fracos, é então definida a especificação do método CompogeMatch (método proposto por este trabalho), com o objetivo de detectar padrões de análise em modelos conceituais. A partir daí, foram implementados os algoritmos de detecção e um portal de submissão dos modelos conceituais para verificação do método. O método proposto lê o modelo conceitual submetido ao portal e aponta os padrões de análise encontrados.

Após a confecção dos algoritmos e do portal, uma coleção de modelos conceituais foi minerada e submetida para validação do CompogeMatch. Os modelos utilizados foram considerados realistas, uma vez que, todos fazem parte de sistemas de informação em utilização no mercado.

1.5 HIPÓTESE

Este trabalho tem como hipótese a melhoria do processo de recuperação de componentes de software em bibliotecas digitais com o uso de padrões de análise (PA) e ontologias no processo de indexação e recuperação. Essa proposta é uma alternativa às buscas por palavras-chave, que satisfazem somente buscas sintáticas - recuperação exata do termo sem uma análise semântica da palavra, enquanto que o método proposto e o uso de ontologias possibilitam inferência semântica.

1.6 RESULTADOS ESPERADOS

Espera-se, com este estudo, apresentar um índice (padrões de análise) de medida de similaridade entre modelos conceituais e componentes de software. Esta medida de similaridade poderia ser utilizada como critério de comparação e como índice nas buscas por componentes de software em bibliotecas digitais.

Além disso, poderá haver redução no tempo gasto para comparar modelos conceituais, uma vez que comparar todas as entidades entre dois modelos sem nenhum índice é mais demorado do que comparar apenas os índices.

1.7 LIMITAÇÕES DESTE TRABALHO

O presente trabalho não tem como objetivo resolver todos os problemas da busca e recuperação de modelos conceituais, mas sim apresentar um meio de recuperação mais completo do que os encontrados na literatura. Portanto, este trabalho apresenta as seguintes limitações:

a) Não contempla a detecção de todos os padrões de análise apresentados por FOWLER (1997).

b) Não faz a engenharia reversa do modelo de dados para o modelo conceitual, uma vez que, nem todo software contém o artefato modelo conceitual.

c) Não leva em consideração os atributos de cada conceito para medir similaridade entre modelos.

d) Os modelos conceituais deverão estar em formato XMI para leitura.

e) Os nomes dos conceitos deverão estar no mesmo idioma da ontologia. Se a ontologia estiver em português, o método só funcionará com modelos em português.

As limitações podem ser resolvidas com a continuação desta linha de pesquisa, que está sugerida na seção de trabalhos futuros no capítulo 6 deste ensaio.

1.8 ESTRUTURA DA DISSERTAÇÃO

Este estudo está organizado em seis capítulos. O capítulo 2 consiste numa revisão bibliográfica sobre conceitos e técnicas de recuperação de software. Ainda neste capítulo também é feita uma revisão sobre modelos conceituais e padrões de análise, bem como os métodos que utilizam padrões para recuperação de modelos e software.

No capítulo 3 é apresentado o método CompogeMatch, proposto por este trabalho, é descrito como ele utiliza ontologias para recuperação de modelos conceituais. Os padrões de análise detectados nesta fase do trabalho e principais algoritmos também podem ser encontrados neste capítulo.

No capítulo 4 são apresentados os elementos para serem utilizados no capítulo de experimentos. São apresentados os modelos conceituais utilizados nos experimentos, bem como, os padrões de análise encontrados.

O capítulo 5 apresenta um estudo de caso e experimentos com seus resultados. Este capítulo utiliza artefatos reais para apresentar uma verificação real do método CompogeMatch.

E, por fim, o capítulo 6 sumariza as contribuições deste trabalho e apresenta as conclusões tomadas a partir dos capítulos anteriores. Ainda são apresentadas algumas sugestões de trabalhos futuros para fins de melhoria e continuação do progresso do método proposto.

2. REVISÃO BIBLIOGRÁFICA

Para melhor compreensão da proposta deste trabalho, esta seção apresenta alguns conceitos, definições e pesquisas que precisam ser observados, e que estão relacionados no processo de detecção de padrões de análise em modelos conceituais.

2.1 MODELOS CONCEITUAIS

O processo de desenvolvimento de um software pode ser subdividido em duas fases: a fase de análise e a fase de projeto. Durante a primeira fase (análise), procura-se levantar e apontar todos os requisitos do sistema a ser desenvolvido. Já na segunda (projeto), o objetivo é executar e implementar o que foi levantado durante a primeira fase. O modelo conceitual é um documento criado na fase de análise com o objetivo de representar o modelo de informação do software em questão.

Algumas considerações sobre modelo conceitual por FOWLER (1997):

> *"When doing analysis you are trying to understand the problem."*
>
> *"It is important to remember that a conceptual model is a human artifact."*
>
> *"Models are not right or wrong; they are more or less useful."*

FOWLER (1997) apresenta o modelo conceitual como um artefato que ajuda a entender o problema - sistema em desenvolvimento.

LARMAN (2001) apresenta o modelo conceitual como uma decomposição do problema. Segundo suas considerações:

> *"O passo mais essencialmente orientado a objetos na análise ou na investigação é a decomposição do problema em conceitos e objetos individuais".*
>
> *"Em UML, um modelo conceitual é exibido como um conjunto de diagramas de estrutura estática, nos quais não se definem operações."*

A seguir apresentam-se algumas definições de WAZLAWICK (2004) sobre modelos conceituais e seu papel no desenvolvimento:

> *“O modelo conceitual deve descrever a informação que o sistema deve gerenciar. Trata-se de um artefato do domínio do problema e não do domínio da solução. Portanto, o modelo conceitual não deve ser confundido com a arquitetura do software (diagrama de classes de projeto)”*
>
> *“O modelo conceitual também não deve ser confundido com o modelo de dados, pois o modelo de dados enfatiza a representação e a organização dos dados armazenados, enquanto o modelo conceitual visa representar a compreensão da informação e não sua representação física.”*
>
> *“O modelo conceitual deve ser independente da solução física que virá a ser adotada e deve conter apenas elementos referentes ao domínio do problema em questão.”*

Das idéias apresentadas por FOWLER (1997), LARMAN (2001) e WAZLAWICK (2004) sobre modelos conceituais destacam-se os seguintes pontos:

a) O modelo conceitual de um software faz parte da investigação do problema.

b) Os conceitos do mundo real que o sistema deverá manipular ou gerenciar devem estar presentes no modelo conceitual.

c) Está diretamente relacionado com o domínio da aplicação (aplicação financeira, área médica, engenharia civil ou setor bancário), por exemplo.

d) Não deve ser confundido com o modelo físico dos dados (banco de dados).

e) Deve ser independente de tecnologias como plataforma operacional, linguagem de programação, modelo físico de dados (banco de dados).

f) Está fortemente relacionado com o paradigma de desenvolvimento orientado a objetos. Em outros paradigmas esse artefato pode sofrer alterações e em alguns casos nem existir (programação estruturada).

g) Apesar das técnicas de geração automática de modelos conceituais já existentes, este ainda é um documento gerado manualmente por analista, vez que é um processo que exige tarefas como, por exemplo, entrevistas com os usuários interessados no sistema em desenvolvimento.

A fig. 1 apresenta um modelo conceitual fictício de um sistema de vídeo locadora representado em UML (Unified Model Language).

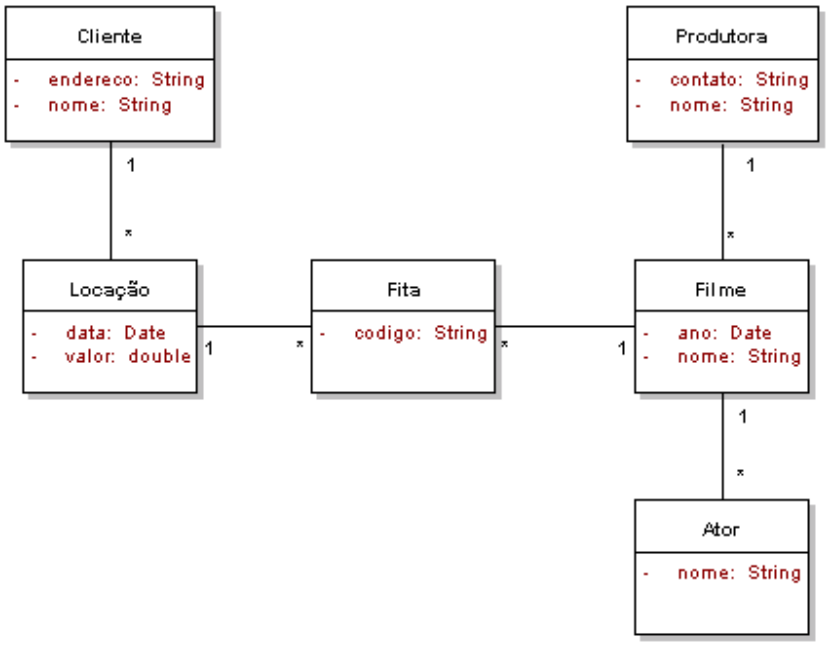


Figura 1: Modelo Conceitual de Locadora (simplificado).

A fig. 1 apresenta os conceitos em forma de retângulos, os atributos estão destacado na parte inferior de cada conceito bem como seu tipo de dado e as associações são as linhas que ligam os conceitos bem como sua multiplicidade.

2.2 BUSCA POR COMPONENTES DE SOFTWARE

A recuperação de informação é objeto de pesquisa da ciência da computação, alguns dos seus benefícios podem ser vistos no dia-a-dia, tais como: encontrar um artigo científico, localizar um mapa geográfico. A partir dos anos noventa, a forma de buscar informações mudou com o uso da Internet, tornando-se, conseqüentemente, mais ágil. Cita-se, como exemplo, um estudante que pode acessar bibliotecas digitais ao realizar suas pesquisas.

O processo de recuperação de uma informação segue um fluxo semelhante em quase todas as situações. Seja recuperando uma imagem médica, um documento texto, um registro de banco de dados, ou até mesmo, um digrama UML. A fig. 2 apresenta esse fluxo:

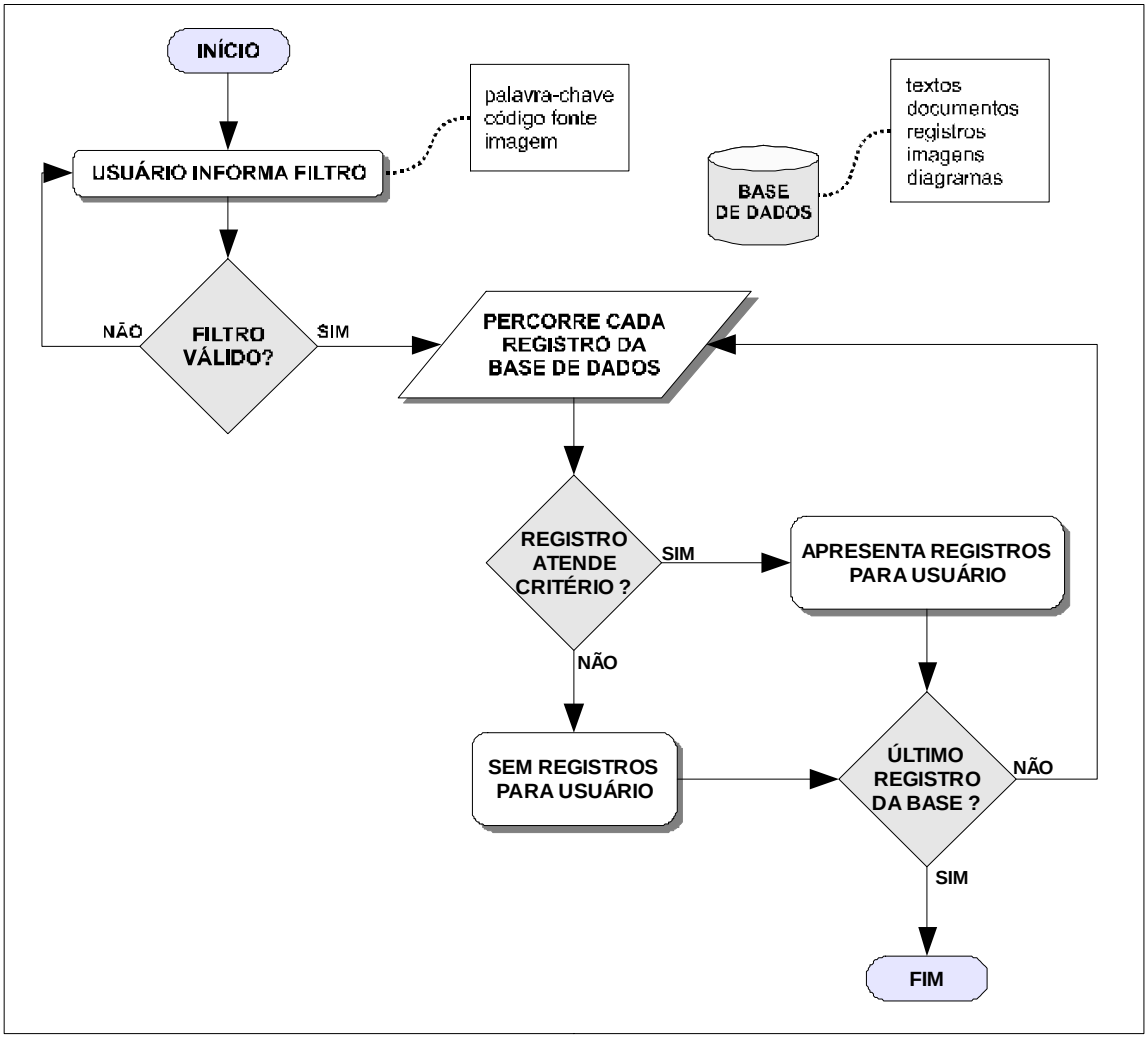


Figura 2: Processo de recuperação de informação.

Considera-se neste processo qualquer tipo de informação, textos, páginas da web, imagens, diagramas e como não poderia deixar de ser citado, o objeto deste trabalho: componentes de software.

Este trabalho limita-se a recuperação de componentes de software a partir de modelos conceituais. São apresentados nas seções seguintes alguns trabalhos que utilizam técnicas diversificadas para recuperação de software.

2.2.1 Recuperação por Processamento de Linguagem Natural (PLN)

A recuperação utilizando processamento de linguagem natural (PLN) é apresentada por GIRARDI e IBRAHIM (1995):

"*Natural Language Processing (NLP) techniques have been applied in information retrieval systems mainly at the lexical, syntactic and semantic levels. The availability of machine readable dictionaries made it possible to use lexical processing techniques in information retrieval. Machine-readable dictionaries have also been used in information retrieval to index text and queries by word senses instead of words themselves [16]. These approaches aim at increasing the precision in retrieval by providing a solution to the polysemy phenomenon, i.e. the fact that words have multiple meanings."*

Este método percorre todos os componentes do software fazendo uma varredura por textos que possam ser interpretados. Tais informações podem estar em diversos locais do software como, em títulos de janelas, comentários e documentação. Localizado os textos, o método os analisa em linguagem natural e os indexa. Possibilitando, dessa forma, futuras buscas e recuperação.

2.2.2 Recuperação por Grafos Conceituais

O método apresentado por MONTES-Y-GÓMEZ, LÓPEZ-LÓPEZ & GELBUKH (2000) mostra como recuperar informações a partir de uma medida de similaridade entre grafos conceituais. Trata-se de uma alternativa à abordagem do uso de palavras chaves (GIRARDI & IBRAHIM, 1995) para recuperar informações.

Esse método mapeia informações textuais em uma estrutura de grafo com todos os conceitos e relações. A partir desse grafo, o método indica como medir similaridade entre estruturas gráficas conceituais.

Uma característica que deve ser ressaltada neste método é o processo de *query* que pode ser realizado utilizando palavras chaves. Nesse processo o método transforma as

palavras submetidas à consulta (*keyword*) em um grafo conceitual, e realiza ao busca utilizando a técnica de comparação dos grafos.

2.2.3 Recuperação por Similaridade Conceitual

A recuperação de software utilizando grafos conceituais investigada por MISHNE e RIJKE (2004), embora tenha uma semelhança com a proposta de MONTES-Y-GÓMEZ, LÓPEZ-LÓPEZ & GELBUKH (2000), procede com uma singular diferença. O método proposto constrói um grafo a partir do código fonte seguindo algumas regras e padrões. Essas regras devem respeitar as particularidades do código-fonte (linguagem não natural, como Java, C++, Smalltalk) para transformar esse código em um grafo de conceitos. Antes do processo de recuperação, um parser analisa o código fonte e o transforma em um grafo conceitual, possibilitando, assim, medir a similaridade conceitual.

Esse método foca em recuperação de códigos-fonte (software), enquanto que a abordagem do método de grafos conceituais pode recuperar software mas também outros documentos textuais.

2.3 PADRÕES DE ANÁLISE EM MODELOS CONCEITUAIS

Uma referência bibliográfica relevante, sobre o uso de padrões de análise na engenharia de software foi apresentada por FOWLER (1997). Este livro cataloga aproximadamente cem padrões de análise e os categoriza em assuntos para facilitar a busca e entendimento de seu uso.

Uma das considerações de FOWLER (1997) sobre a dificuldade em se conceituar padrões: *"Certainly it is difficult to find any common definition of pattern."* (Certamente é difícil encontrar alguma definição comum para o conceito de padrão). Por isso, encontrar uma definição comum sobre o que venha a ser padrões de análise não é uma tarefa simples. Segue a definição mais popular do que é um padrão: um conjunto composto pelos elementos: um *nome*, um *problema* e uma *solução* dentro de um determinado contexto. Apesar disso essa afirmação é contestada por FOWLER (1997):

> *"It is commonly said that a pattern, however it is written, has four essential parts: a statement of the context where the pattern is useful, the problem that the pattern addresses, the forces that play in forming a solution, and the solution that resolves those forces. This form appears with and without specific headings but underlies many published patterns. It is an important form because it supports the definition of a pattern as 'a solution to a problem in context', a definition that fixes the bounds of the pattern to a single problem-solution pair."*

Contudo, ele concorda que todo padrão deverá ter um nome*:*

> *"One principle of pattern form that I do agree with unreservedly is that they should be named."*

Na seqüência, ele justifica seus benefícios:

> "*One advantage of patterns work is how it can enrich the vocabulary of development. By just saying 'use a protection proxy here' or 'we used observations to record product metrics', we can communicate our design ideas very effectively*."

A seguir FOWLER (1997) apresenta como os padrões são concebidos e formalizados:

> "*To many patterns people, one of the key elements of patterns is that they are discovered by looking at what happens in day-to-day development, rather than by academic invention. This is an element that I find particularly important. All the patterns in this book are the result of one or more actual projects and describe useful highlights in that work*."

Existe a aplicação de padrões em outras áreas:

> "*The idea of software patterns is not confined to the object-oriented community; David Hay has written a valuable book on data model patterns. The models follow relational data modeling style, but they are very conceptual models. This makes the models valuable even if you are using object technology*."

A seguir são apresentadas as definições por FERNANDEZ e YUAN (2000) sobre padrões de análise:

> "*An analysis pattern is a set of classes and associations that have some meaning in the context of an application; that is, it is a conceptual model of a part of the application. However, the same structure may be valid for other applications, and this is the aspect that makes them very valuable for reuse and composition*."

Nota-se que a definição apresenta um raciocínio um pouco diferente da defendida por FOWLER (1997). Mesmo assim, a noção de reuso e reaproveitamento está bem evidente

nas definições de ambos autores. Além disso, destaca-se a reaplicação dessas estruturas padrões (PA) em outras aplicações.

Para visualizar e consolidar os conceitos acima levantados, a seguir são apresentadas figuras com o padrão *Party* e *Organization Hierarchies*:

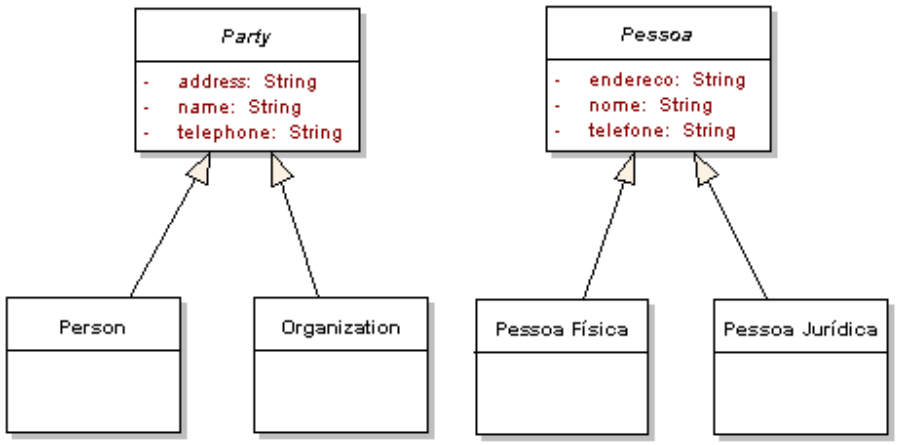


Figura 3: O padrão Party (Pessoa) original e em português

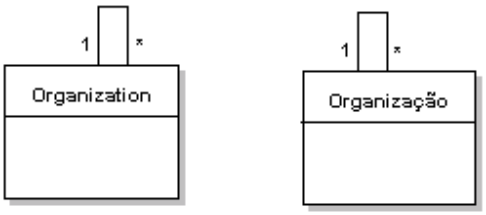


Figura 4: O padrão *Organization Hierarchies* (Hierarquia Organizacional) original e em português

Como se pode observar, padrões de análise são elementos que compõem o domínio do problema. Quando um analista precisa modelar o diagrama conceitual do software, freqüentemente este se depara com problemas recorrentes. Quando isto ocorre, os padrões de análise podem ser elementos importantes na solução desses problemas. A seguir são apresentadas três especificações de software (bem sintetizadas) para ilustrar a solução (modelo conceitual) para estas especificações:

Especificação 1: Um projeto para controle em uma biblioteca de um centro universitário, contemplando reserva, empréstimo e devoluções de itens do acervo. O centro universitário possui várias unidades, sendo uma biblioteca central e outras unidades espalhadas pelo centro universitário. O sistema deverá rodar tanto na unidade central quanto nas demais unidades.

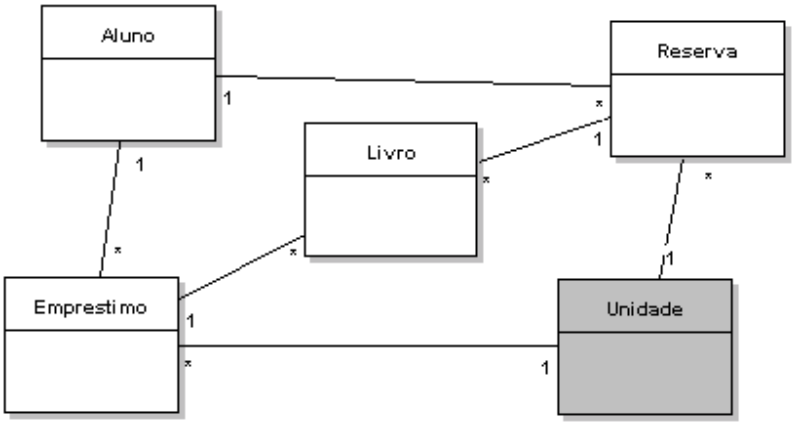


Figura 5: Modelo conceitual da especificação 1.

Especificação 2: Um sistema de acompanhamento de documentos (processos) em secretarias estaduais a fim de saber com precisão a localização do documento quando for consultado no sistema, além de apresentar a quantidade de departamentos em que o documento tramitou.

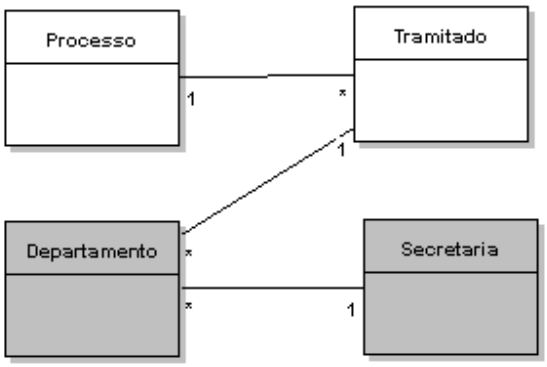


Figura 6: Fragmento do modelo conceitual da especificação 2

Especificação 3: Um sistema para rastreamento de carga para uma transportadora com 10 filiais.

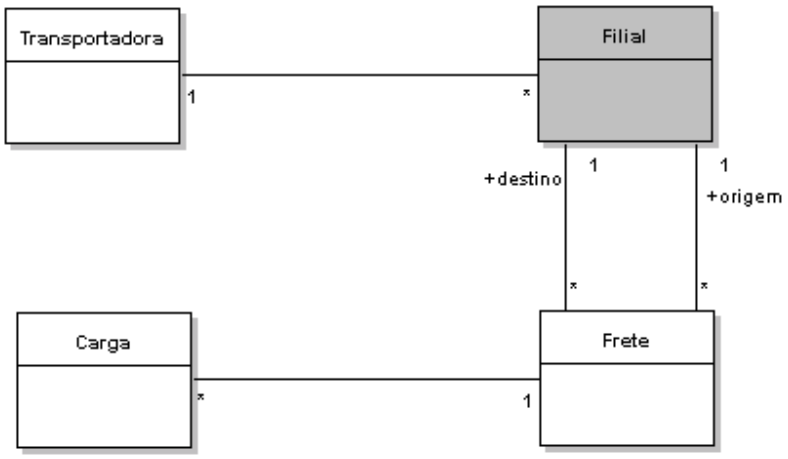


Figura 7: Modelagem da especificação 3

Os modelos conceituais das figuras 5, 6 e 7 destacam a possível ocorrência do padrão *Organization Hierarquies.* Poderiam ser citadas outras especificações, no entanto o objetivo é mostrar que é comum a recorrência de problemas em domínios distintos. Em todos os projetos uma necessidade evidente é o controle das informações (livros, carga e documentos) levando em consideração sua unidade organizacional (unidades, filiais, setores ou departamentos). Este é um exemplo de como projetos diferentes podem exigir

necessidades padrão. Nas três especificações de projetos apresentadas acima, a adoção do padrão *Organization Hierarquies* padroniza e ajuda na construção do modelo conceitual, uma vez que a falta de padrão pode e produz soluções distintas para o mesmo problema.

Por fim, a adoção de padrões de análise pode ajudar analistas a construir soluções mais elegantes quando estão modelando software. Analogamente os desenvolvedores podem escrever códigos mais limpos quando adotam padrões de projetos. É possível consultar mais informações em FOWLER (1997) sobre padrões de análise.

2.4 ONTOLOGIA

O conceito de ontologia utilizado neste trabalho não deve ser confundido com a ontologia do campo da filosofia. Na filosofia a ontologia estuda a natureza do ser e suas origens. Já na ciência da computação, a ontologia pode ser definida como uma subárea da inteligência artificial, que estuda a representação de um conhecimento para a máquina.

A ontologia é uma ferramenta que especifica para a máquina a definição dos conceitos do mundo real e suas relações. Além disso, ela deve ser direcionada a domínios (áreas de interesse) específicos, como por exemplo: negócios, saúde, área médica, engenharia civil. Não é usual a utilização de ontologias que contemplem todos os assuntos, até mesmo por que o processamento desta ontologia demandaria muito esforço computacional.

GUARINO e GIARETTA (1995) apresentam as seguintes idéias sobre o termo ontologia:

> "*Ontology as a philosophical discipline*" (Ontologia como uma disciplina da filosofia)
> "*Ontology as a an informal conceptual system*" (Ontologia como um sistema informal de conceitos)
> "*Ontology as a formal semantic account*" (Ontologia como um conjunto semântico formal)
> "*Ontology as a specification of a conceptualization*" (Ontologia como uma especificação de uma conceitualização)
> "*Ontology as a representation of a conceptual system via a logical theory*" (Ontologia como uma representação de um sistema conceitual por meio da uma teoria lógica)
> - *Characterized by specific formal properties* (Caracterizada por especificação de propriedades formais)
> - *Characterized only by its specific purposes* (Caracterizada somente para propósitos específicos)

> "*Ontology as the vocabulary used by a logical theory*" (Ontologia como um vocabulário usado por uma teoria lógica)
> "*Ontology as a (meta-level) specication of a logical theory*" (Ontologia como uma especificação (nível-meta) de uma teoria lógica)

As definições apresentadas por GUARINO e GIARETTA (1995) são amplas e conceituam de forma abrangente o domínio de ontologias.

SCHREIBER, WIELINGA e JANSWEIJER (1995) apresentam outra idéia de ontologia, como segue:

> "*An ontology is an explicit, partial specification of a conceptualization that is expressible as a meta-level viewpoint on a set of possible domain theories for the purpose of modular design, redesign and reuse of knowledge-intensive system components.*"

Diante de todas as definições apresentadas sobre ontologias, algumas idéias estão em consonância entre os autores citados, a saber:

a) Uma ontologia representa conhecimentos acerca de elementos de um domínio.

b) Um conjunto de conceitos fortemente relacionados entre si.

c) Uma contextualização e conceitualização do mundo real.

d) Uma ontologia está limitada a um contexto.

e) Uma especificação formal.

Com isso, pode-se afirmar que a ontologia na área de engenharia de software tem como proposta armazenar conhecimento previamente construído por um especialista, para ser utilizado por ferramentas e algoritmos. Maiores informações sobre o assunto (RUSSELL & NORVIG, 2004) (GÓMEZ-PÉREZ, RAMÍREZ e VILLAZÓN-TERRAZAS, 2007).

Por fim, resumidamente, é possível dizer que a ontologia contém o conhecimento que teria um ser humano a respeito de um determinado assunto. Assim, algoritmos podem fazer uso desse conhecimento (contido na ontologia) para a tomada de decisões e inferências.

A fig. 8 apresenta um exemplo de uma ontologia para ajudar a esclarecer as idéias apresentadas até aqui:

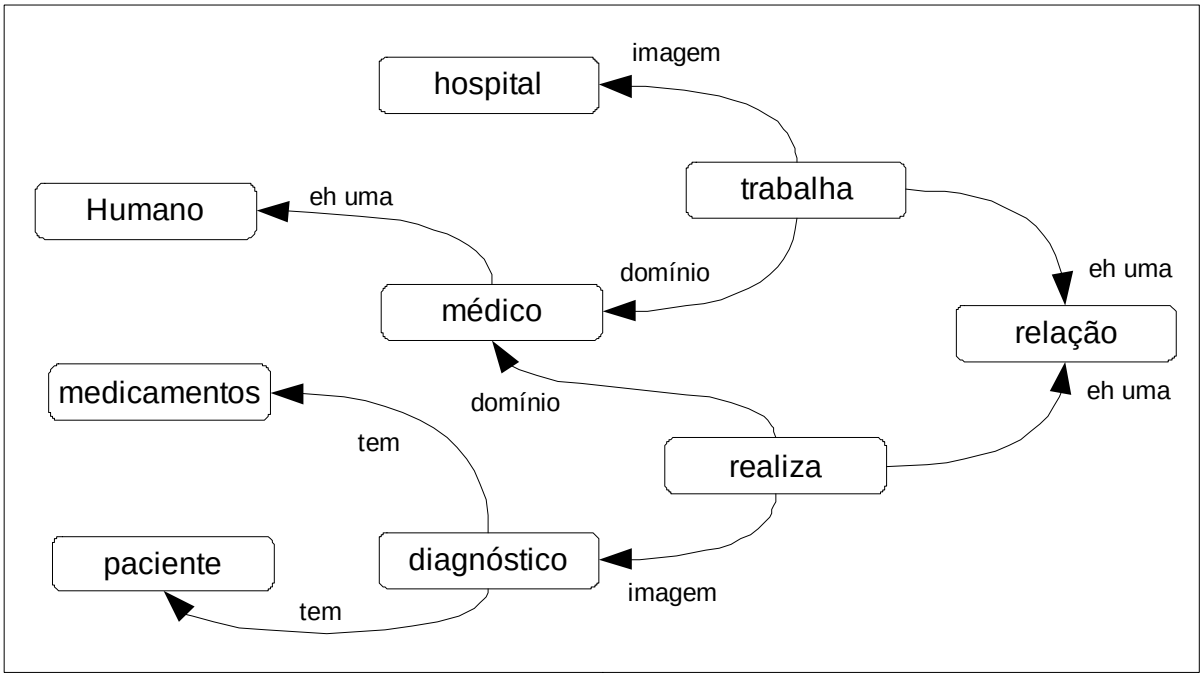


Figura 8: Ontologia Exemplo da Área Médica

Observa-se que a ontologia é uma representação de conhecimento - informações e relações do mundo real. A partir dessa estrutura ou diagrama (rede semântica) é possível escrever um algoritmo que possa, por exemplo, inferir que Médico é um Humano. Isso por que existe uma relação *eh uma* entre essas duas entidades. Seguindo o mesmo raciocínio, pode-se verificar que um Médico *realiza* um Diagnóstico em um Paciente. Sempre observando as relações existentes na ontologia. Esse é o papel da ontologia nesta pesquisa, representar conhecimento e dar subsídios para que os algoritmos implementados possam saber o que significam as entidades presentes nos modelos conceituais. Isso será mais detalhado no próximo capítulo.

3 DETECÇÃO DE PADRÕES DE ANÁLISE EM MODELOS CONCEITUAIS

Os mecanismos de busca e recuperação de componentes de software, levantados na revisão bibliográfica, possuem limitações que são apresentadas neste capítulo. Esse estudo, segue numa abordagem de minimizar essas limitações no tocante às comparações entre modelos conceituais.

Este capítulo tratará de forma detalhada o método CompogeMatch e seus elementos: modelo conceitual, padrões de análise e ontologia, bem como o papel de cada elemento no funcionamento do método.

3.1 ARTEFATOS DO MÉTODO CompogeMatch

Para melhor compreender a idéia e o funcionamento do método proposto, é imprescindível a apresentação dos elementos, seus papéis e atributos essenciais de cada artefato.

3.1.1 Modelo Conceitual

Um modelo conceitual é composto por conceitos, atribuitos, associações, nomes dos conceitos, multiplicidade entre outros elementos. Nem todos esses elementos são necessários para a execução do método.

Para detectar PAs (padrões de análise) em modelos conceituais os seguintes elementos são considerados pelo método:

a) O nome do conceito ou entidade.

b) As associações e sua multiplicidade.

Na versão apresentada por este trabalho optou-se por não adotar os atributos dos conceitos no processo de detecção de padrões de análise, sugere-se como trabalho futuro para complementar esta pesquisa.

O modelo conceitual é essencial no processo de recuperação de componentes, pois a partir dele o CompogeMatch pode definir o domínio do sistema. Com as entidades desse modelo conceitual, o CompogeMatch pode diferenciar, por exemplo, um sistema de registro de pacientes, de um sistema para locadora de carros.

A seguir são apresentados dois exemplos de modelos conceituais de áreas distintas:

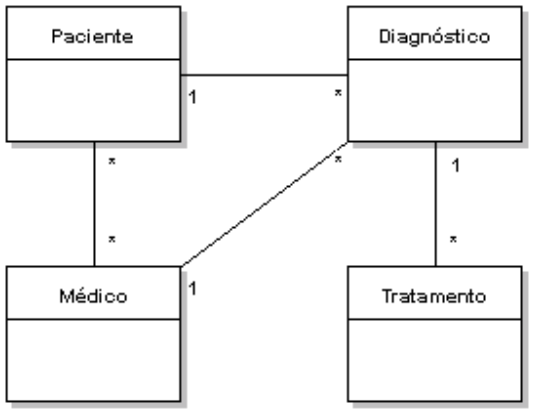


Figura 9: Um Modelo Conceitual Exemplo da Área Médica

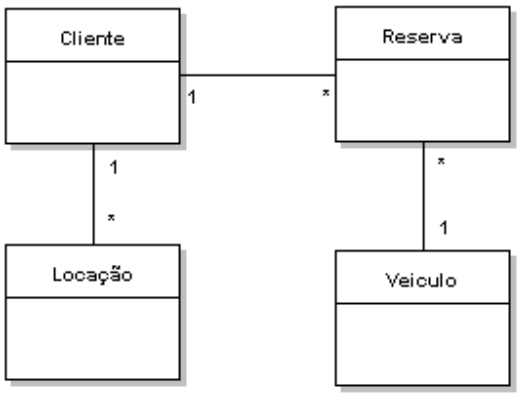


Figura 10: Um Modelo Conceitual Exemplo do Domínio de Locadora de Veículos

3.1.2 Padrões de Análise (Analysis Patterns)

Uma revisão sobre o conceito de padrões de análise foi apresentado no segundo capítulo deste trabalho. Nesta seção serão mostrados apenas os elementos dos padrões de análise que são relevantes no processo de detecção.

Um padrão de análise pode ser composto por: entidades, associações com multiplicidade e atributos. Mesmo assim, nem todas as propriedades são utilizadas no processo de detecção. As propriedades envolvidas no processo de busca são:

a) O nome do padrão de análise.

b) Os conceitos que compõem o padrão.

c) As associações entre os conceitos, caso o padrão de análise compreenda mais de um conceito/entidade.

O padrão de análise é o elemento-chave a ser encontrado dentro dos modelos conceituais, as propriedades acima foram consideradas suficientes para efeito desta pesquisa. Ainda assim, outras propriedades podem ser utilizadas nesse processo de detecção, como por exemplo, os atributos dos conceitos.

3.1.3 Ontologia

O papel da ontologia no método CompogeMatch é representar o conhecimento sobre o domínio do qual o modelo conceitual faz parte. Significa que, a ontologia deverá conter todos os conceitos dos modelos conceituais submetidos ao método. Por exemplo, ao serem submetidos os modelos conceituais das figuras 9 e 10, têm-se: *Cliente*, *Reserva*, *Locação*, *Veículo*, *Médico*, *Paciente*, *Diagnóstico* e *Tratamento*, que deverão estar contidos na ontologia.

Para que a ontologia seja utilizada pelo método proposto, algumas regras devem ser seguidas, a saber:

a) Todos os conceitos dos modelos conceituais submetidos devem estar presentes como termos (entradas) da ontologia. Qualquer omissão pode comprometer o rendimento da busca.

b) O idioma da ontologia deve estar em conformidade com o idioma dos modelos conceituais. Assim, se os modelos conceituais forem submetidos em português, os conceitos da ontologia deverão constar em português.

c) As classes da ontologia devem ter propriedades, que destaquem e definam do que se trata esse conceito, já que serão utilizados pelos algoritmos de detecção. Isso é discutido em mais detalhes a partir da seção 3.4.

Além das regras acima, vale ressaltar que, no processo de detecção, uma das vantagens no uso de ontologias é a possibilidade de extrair informações a partir das relações “é um” ou “tem um”.

3.2 O MODELO DE INDEXAÇÃO E RECUPERAÇÃO PROPOSTO

Uma vez apresentados os artefatos e os atributos importantes nas seções anteriores, esta seção cuidará de como o método, CompogeMatch, utiliza esses elementos para o processo de indexação e recuperação dos componentes.

Os métodos de recuperação supracitados, (GIRARDI & IBRAHIM, 1995, MAAREK, BERRY e KAISER, 1994, MISHNE & RIJKE, 2004, MONTES-Y-GÓMEZ, LÓPEZ-LÓPEZ & GELBUKH, 2000) apresentam algumas limitações. Por exemplo, não distinguem palavras sinônimas. Mesmo que haja um mapeamento prévio para eliminar essa limitação, o mapeamento está condicionado a uma relação direta entre os termos equivalentes, o cenário que segue demonstra essa situação. Considere os modelos das fig. 7 e 8, como dois sistemas criados por equipes distintas:

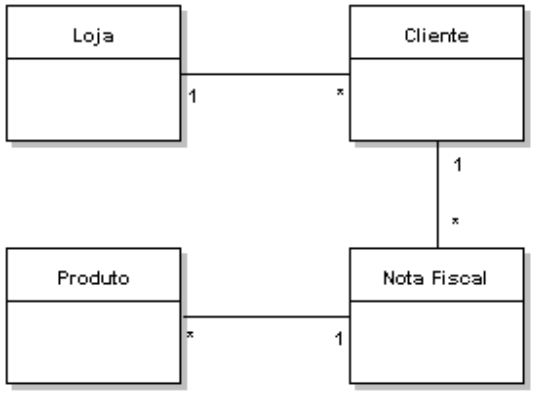


Figura 11: Modelo Conceitual 1 - Equipe A

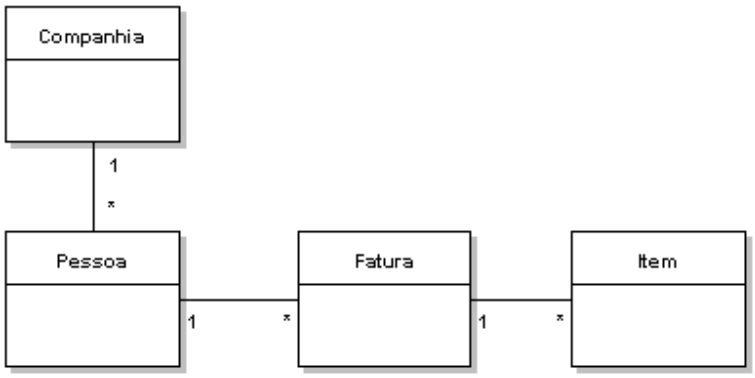


Figura 12: Modelo Conceitual 2 - Equipe B

Apesar dos modelos conceituais (equipe A e B) apresentarem nomes de conceitos diferentes, pode-se dizer que eles tratam do mesmo assunto.

É difícil, utilizando-se apenas sintaxe, apontar a semelhança entre modelos conceituais aparentemente distintos como os modelos das figuras 11 e 12. Um ser humano pode dizer que esses modelos são equivalentes, pois compreende a equivalência entre os termos Loja e Companhia, Cliente e Pessoa, Produto e Item, Nota Fiscal e Fatura. No contexto de sistemas de faturamento, esses termos possuem a mesma conotação. Dessa forma, uma busca por palavras-chave, sem o mapeamento prévio desses termos, comprometeria o resultado da pesquisa.

3.3 A ONTOLOGIA UTILIZADA NO PROCESSO DE DETECÇÃO

A figura 13 apresenta uma ontologia utilizada no processo de detecção. Para melhor compreender as etapas que serão apresentadas na próxima seção deve ser observada a ontologia e suas propriedades. Por mera organização, alguns termos da ontologia foram suprimidos. A ontologia completa pode ser consultada no anexo 2:

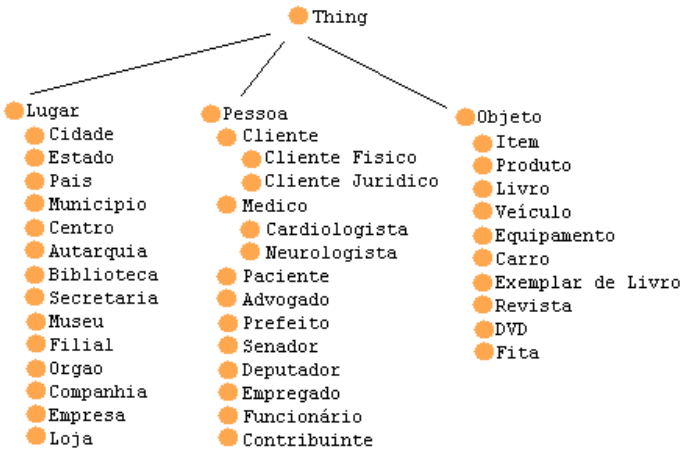


Figura 13: Ontologia usada na apresentação do método

Nota-se que os conceitos descendentes de *Pessoa* são considerados *Pessoa*, pela relação de herança “e-um” da ontologia. O mesmo ocorre com os conceitos *Lugar* e *Objeto.* Sendo assim, basta perguntar pra ontologia se o conceito *Contribuinte é uma Pessoa* que a resposta será verdadeira, a mesma relação ocorre entre os conceitos *Companhia* e *Lugar*.

3.4 PADRÕES DE ANÁLISE DETECTADOS

Os PAs que podem ser detectados na versão do método proposto neste trabalho, são: *Party* (figura 14), *Organization Hierarchies* (figura 15) e *Contract* (figura 16).

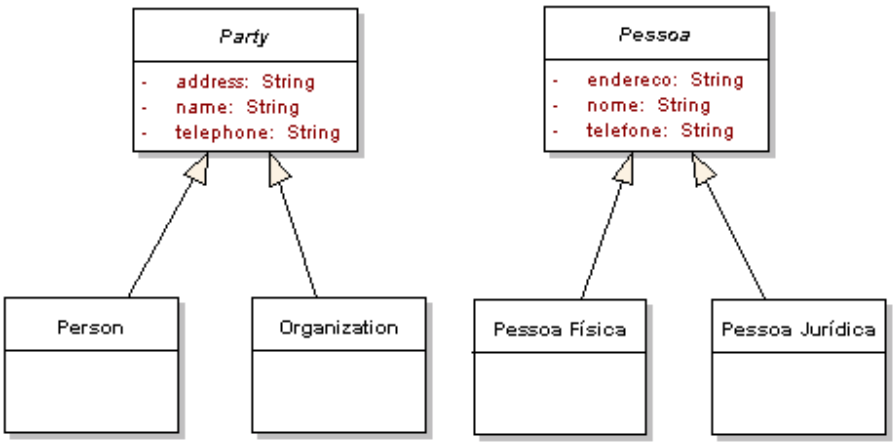


Figura 14: O padrão *Party* (Pessoa) original e em português

O padrão *Party* é muito utilizado quando o sistema precisa representar informações referentes a pessoas como, por exemplo: clientes, vendedores, usuários, fornecedores, alunos, estudantes, etc. Todos esses termos são candidatos ao padrão *Party.* Nota-se a especialização do padrão *Party* em pessoas físicas e jurídicas, pois dependendo do tipo de aplicação essa divisão é importante e faz parte do contexto da aplicação.

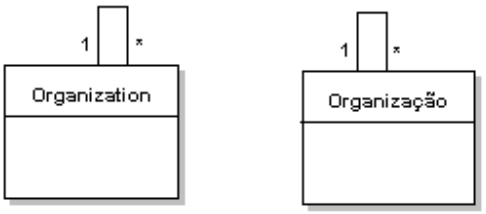


Figura 15: O padrão *Organization Hierarchies* (Hierarquia Organizacional) original e em português

O padrão *Organization Hierarquies* é um padrão muito utilizado e apropriado quando se deseja representar informações em unidades organizacionais. Organizações e suas filiais, secretarias de estados e seus departamentos ou setores de uma universidade são exemplos de conceitos que podem ser modelados conforme o padrão *Organization Hierarquies.*

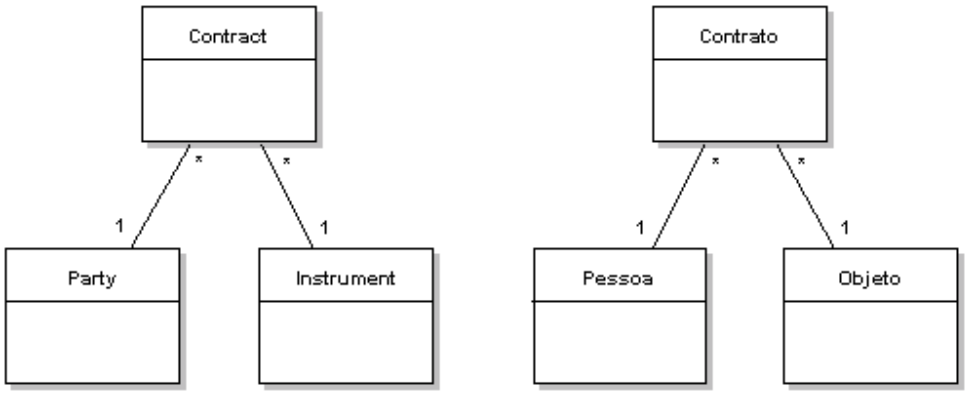


Figura 16: O padrão *Contract* (Contrato) original e em português

O padrão *Contract* é muito utilizado para representar, por exemplo, uma transação comercial (venda, locação, empréstimo), transação esta que envolve uma pessoa, uma operação e um ou mais objetos. A operação que consiste em um cliente comprar/faturar um produto de uma loja pode ser representada pelo PA *Contract*. Uma vez que estão envolvidos nestas operações: um cliente, uma transação e um item (normalmente um produto). Neste padrão nota-se a ocorrência também do padrão *Party* (a pessoa envolvida no contrato), assim é requisito para a ocorrência do padrão *Contract* antes ocorrer o PA *Party*.

De mão dos artefatos apresentados acima, a seção seguinte mostrará as etapas do método e sua implementação.

3.5 O MÉTODO COMPOGEMATCH

Agora que foram apresentados os artefatos necessários para o funcionamento do método CompogeMatch, será apresentado nesta e nas próximas seções como o método utiliza-os e produz os objetivos propostos nesta pesquisa. O método consiste em apresentar a detecção de PAs em modelos conceituais. Dados os conceitos de um modelo conceitual é possível através da ontologia saber os seus significados, como por exemplo, saber que *Paciente* e *Médico* correspondem a um ser humano ou uma *Pessoa* (se estes conceitos estiverem mapeados e relacionados na ontologia). Essa afirmação só é possível com a ajuda da ontologia e suas relações, por isso a ontologia é o artefato chave deste método. A partir

daí, é possível estender o campo de perguntas à ontologia, e definir as regras para apontar a ocorrências dos padrões.

Sabe-se que o padrão *Party* é por definição uma pessoa ou um participante FOWLER (1997) como exemplos: médico, cliente, advogado, professor, estudante, aluno entre outros, dependendo do domínio da aplicação. Assim, submetido um modelo conceitual para o método, este percorre cada conceito e através da string do nome dos conceitos, verifica se existe uma string correspondente na ontologia que possa dizer que este conceito é uma *pessoa*. Se for encontrado o conceito na ontologia e este termo for uma *pessoa*, o método aponta o conceito como candidato ao PA *Party.*

Essa foi apenas uma visão geral do método: a seguir são apresentadas todas as etapas propostas pelo método, bem como os algorimos que as implementam. As etapas que seguem foram enumaradas para referencias posteriores neste capítulo:

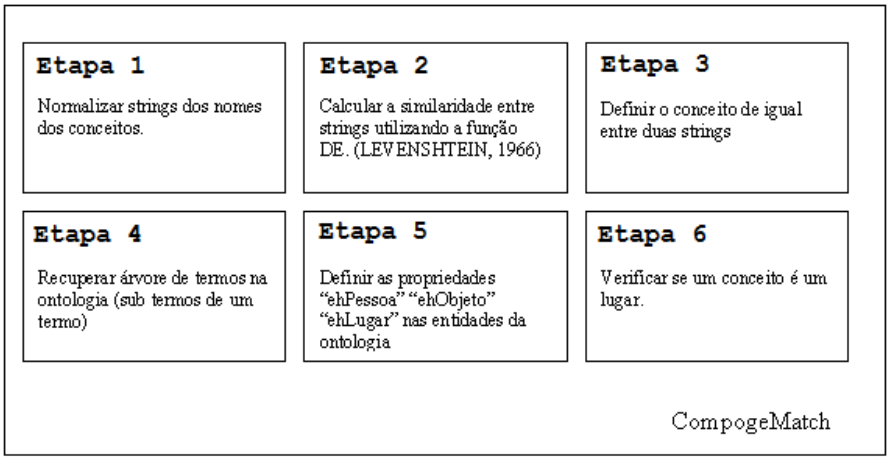


Figura 17: Etapas contempladas no método CompogeMatch

As etapas apresentadas na fig. 17 foram observadas, e consideradas necessárias para o objetivo principal desta pesquisa durante a fase de revisão bibliográfica e definição dos objetivos.

Vê-se, a seguir, a explicação de cada etapa para melhor compreender o funcionamento do método:

ETAPA 1: Normalizar strings dos nomes dos conceitos:

A etapa de normalizar string no processo de matching foi definida para evitar com que a análise/verificação sintática dispense termos semanticamente iguais. Por exemplo, se as strings “Computação” e “Computacao” forem submetidas ao método, este considerará a mesma palavra, pois no processo de normalização serão cortados os sinais de acentuações. Outra situação comum é a aparição de sinais como “_” (undescore), exemplo: “Engenharia de Software” e “Engenharia_de_Software” neste caso também serão consideradas equivalentes as strings.

ETAPA 2: Calcular a similaridade entre strings utilizando a função de LEVENSHTEIN (1966). Um dos métodos utilizados para comparar a similaridade entre strings é a função definida por LEVENSHTEIN (1966) (GILLELAND, 2007) chamada de distância por edição (DE). O CompogeMatch utiliza esta função para evitar com que strings muito parecidas como “Médico” “Médicos” sejam consideradas diferentes. O resultado do cálculo desta função para as strings “Médico” e “Médicos” é 0.83. A seguir uma tabela com algumas palavras e o resultado da função DE:

Tabela 01: Resultado da função de LEVENSHTEIN (1966) com e sem nomalização

String 1	**String 2**	**Levenshtein sem normalização**[1]	**Levenshtein com normalização**
Médico	Medicos	0,666	0,833
Computação	Computacao	0,800	1,000
Engenharia de Software	Engenharia_de_Software	0,909	1,000
Profissão	Professor	0,666	0,666
Advogado	Advogados	0,875	0,875
Professor	Professar	0,888	0,888

Função DE por Levestein é dada por:

[1] Nessa coluna as strings não passaram pela função de normalização. Observa-se que, dessa forma o resultado sempre se mostrou inferior ao resultado apresentado na coluna em que as strings foram normalizadas (quarta coluna da tabela 1) antes de passar pela função de LEVENSHTEIN (1966).

$$DE(:=max(0,\frac{(min(|L_i|,|L_j|)-de(L_i,L_j))}{(min(|L_i|,|L_j|))})\in[0,1]$$

Nesta pesquisa foi adotado um fator de 0.8 para considerar que as strings são equivalentes. Dessa forma, se duas strings forem comparadas, e o resultado produzido pela função de distância por edição for maior ou igual a 0.8, essas strings serão consideradas equivalentes.

Faz-se necessária, a seguinte observação sobre os resultados da última linha da tabela 1. As strings “Professor” e “Professar” produziram um resultado acima do esperado (0.888). Assim, essas strings serão consideradas equivalentes apesar de não existir nenhuma relação entre si. Por conseguinte, pode-se comprometer a eficácia do método proposto por este trabalho. Desse modo, adota-se o uso de ontologia para completar as etapas deste método, porém isto será explicado nas seções que seguem.

ETAPA 3: Definir o conceito de igualdade entre duas strings

A partir das etapas 1 e 2, foi definido o conceito de igualdade entre duas strings. Assim, se duas string forem iguais, independente da capitalização, ou se a DE entre as duas, for maior ou igual a 0.8 (explicado na etapa 2), as strings serão consideradas equivalentes. Sugere-se um estudo futuro sobre o valor ideal para este fator.

ETAPA 4: Recuperar Árvore de Termo na Ontologia

Dado um termo da ontologia (um conceito) é preciso recuperar todos os termos que são sub termos (ou filhos) deste. Essa operação será necessária para responder a pergunta da 6ª etapa.

ETAPA 5: Como a ontologia deverá ser escrita

Esta etapa é importante e deve-se tomar um cuidado especial pois sua implementação depende exclusivamente do especialista criador da ontologia (ser humano). Para permitir o funcionamento do CompogeMatch faz-se necessário definir a ontologia com a hierarquia correta, ou seja, todos os conceitos descendentes de Pessoa deverão estar abaixo deste

conceito. Assim como os conceitos descendentes de Objeto e Lugar. Se essa etapa/regra não for seguida, o funcionamento do método pode ser comprometido.

ETAPA 6: Verificar se um conceito é um lugar

O conceito Cidade é um lugar? Com as etapas já apresentadas, torna-se fácil responder essa pergunta.

Dado um conceito, o primeiro passo é verificar se este está presente na ontologia - considerando as etapas de normalização e similaridade entre strings apresentadas até o momento (etapa 3). Se o conceito não for encontrado na ontologia, nenhuma afirmação poderá ser feita. Caso contrário, o segundo passo é verificar se o conceito encontrado ou um conceito superior (observar relações de herança *é um)* é um conceito filho do conceito Lugar. Se uma dessas condições for satisfeita, pode-se responder que sim, o conceito Cidade é um lugar.

Esta etapa deve ser aplicada aos conceitos Pessoa e Objeto.

Apresentadas as etapas e pretensões do método, cabe agora validá-las através de uma linguagem forma de programação. As seções seguintes apresentam os algoritmos que implementam o método CompogeMatch.

3.6 ALGORITMOS DO MÉTODO

Nesta seção são apresentados os algoritmos que implementam o método CompogeMatch. Para o entendimento do método proposto, faz-se necessário apresentar algumas funções básicas do framework. O código-fonte (escrito em Java) completo encontra-se na seção 3 dos anexos deste trabalho. Aqui são apresentadas somente as principais funções:

a) Função para normalizar as strings como nomes das classes, termos da ontologia, implementação da etapa 1:

```
String PTBR <= "çÇáéíóúÁÉÍÓÚãõÃÕàèìòùÀÈÌÒÙüÜâêîôûÂÊÎÔÛ"
String  NORMALIZE <= "cCaeiouAEIOUaoAOaeiouAEIOUuUaeiouAEIOU"
String TOKENS <= " _0123456789!#$@¬§°¢£³²¹%¨&*()_-+=?/;:.>,<~^]}[{´`'\""
Função String normalize(String value)
Inicio
   String result <= ""
   Para i = 0 ate tamanho(value) Faça
      Caracter str <= Cadeia(value,i,i+1)
      Inteiro pos <= Posição(PTBR,str)
      Se (pos != -1) Então
         result <= result + SubString(NORMALIZE,pos, pos+1)
      Senão
         Inteiro pos_tokens <= Posição(TOKENS,str)
         Se (pos_tokens != -1) Então
            result <= result + ""
         Senão
            result <= result + str
         Fim Se
      Fim Se
   Fim Para
   Retorno result
Fim
```

Exemplos: normalize(“médico”) -> medico

normalize(“Ciência da Computação”) -> CienciadaComputacao

Essa função elimina os caracteres e símbolos que possam atrapalhar uma comparação entre duas strings.

b) Função que define a distância de edição entre duas strings LEVENSHTEIN (1966) (GILLELAND, 2007). A função Distance.LD utilizada por esta função está no anexo 3, Implementação da etapa 2:

```
Função Double similarity(String a, String b) {
Inicio
   a <= TransformaEmMaiusculo(a)
   b <= TransformaEmMaiusculo(b)
   Double a_len <= Tamanho(a)
   Double b_len <= Tamanho(b)
   Double min_len <= Menor(a_len, b_len)
   Inteiro ed_value <= DistanciaEdicao.calcular(a, b)
   Double calc <= ((min_len - ed_value) / min_len)
   Retorno Maior(0, calc)
Fim
```

Exemplos: similarity(“professor”,”professores”) -> 0,777

similarity(“médico”,”médicos”) -> 0,833

similarity(“população”,”POPULAÇÃO”) -> 1,000

Essa função tem a estrita responsabilidade de calcular a distância de edição entre duas strings normalizadas ou não.

c) A função *isEqualsStrings*(String a, String b) compara se as strings passadas como parâmetro (a e b) são iguais. O critério utilizado para considerar que duas strings são iguais é o resultado do cálculo da função *similarity (etapa 2).* Se este for superior a 0,8, será considerado que as strings são equivalentes, caso contrário a função *isEqualsStrings* retornará que as strings são diferentes. Um ponto que merece atenção é que este método normaliza as strings antes de calcular a similaridade entre elas.

Esta função é a implementação da etapa 3:

```
Função Lógico isEqualsStrings(String a, String b)
Inicio
   Se (Maiusculo(a) == Maiusculo(b)) Então
     Retorno Verdadeiro
   Fim Se
   String normalized_a <= normalize(a)
   String normalized_b <= normalize(b)
   Se (Maiusculo(a) == Maiusculo(b)) Então
      Retorno Verdadeiro
   Fim Se
   Double sm <= similarity(normalized_a, normalized_b)
   Se (sm >= 0.8) Então
      Retorno Verdadeiro
   Fim Se
   Retorno False
Fim
```

Exemplos: isEqualsStrings(“médicos”,”médico”) -> true

isEqualsStrings(“professor”,”Professores”) -> true

isEqualsStrings(“professor”,”educador”) -> false

Os exemplos acima mostram os resultados dessa função. Um ponto a ser ressaltado é que, no caso das strings Professor e Educador, a função retornou que as strings não são iguais, uma vez que esta função faz uma análise sintática das palavras. Apesar de não ser objeto deste trabalho discutir o significado de ‘Professor‘ e ‘Educador’, no contexto de sistemas de informação e modelos conceituais esses termos poderiam ser considerados

equivalentes. Esse é mais um motivo para este método adotar uma ontologia como artefato que auxilia na definição semântica dos termos e conceitos.

d) A função *getSubClassByParent(String className, boolean too)* tem o propósito de recuperar todos os termos da ontologia que possuem um mesmo conceito pai (passado como parâmetro para a função), implementação da etapa 4**:**

```
Função OntologyClass[] getSubClassByParent(String className)
Inicio
   Inteiro k <= 1
   OntologyClass[] result = new OntologyClass[]
   Para i = 1 Até Tamanho(classMap) Faça
      OntologyClass element = classMap[i]
      Se Pai(classMap, element) == className
         Result[k] <= getSubClassByParent(element))
      Fim Se
   Frim Para
   Retorno result
Fim
```

Exemplos: getSubClassByParent(“Pessoa”, true) -> [Pessoa, Cliente, Paciente,...]
getSubClassByParent(“Paciente”, true) -> [Paciente]

Duas observações para facilitar a leitura dessa função: primeira: A função *Pai* chamada nesta função retorna o nome do pai do conceito passado como parâmetro; segunda: a variável *classMap* é uma coleção contendo todos os conceitos da ontologia.

e) A função *getPlace(String className)* listada no quadro que segue recebe como parâmetro uma string que é o nome do conceito do modelo conceitual onde espera-se saber se este é um conceito lugar ou não. A partir daí, o método percorre todos os conceitos da ontologia que são descendentes do conceito Lugar. Nesta iteração, caso o nome de algum conceito for equivalente (considerando o critério de similaridade entre strings explicado na etapa 3) ao nome do conceito, a função retornará esse conceito da ontologia. Caso contrário, um objeto nulo será retornado nesta função.

Implementação das etapas 4, 5 e 6**:**

```
Função OntologyClass getPlace(String className)
Inicio
   OntologyClass result <= null;
   Para I = 1 Até Tamanho(placeMap) Faça
      OntologyClass klass <= placeMap[i];
      Se isEqualsStrings(klass.getName(), className)) Então
         result = klass;
      Fim Se
      Se (result == null) Então
        OntologyClass[] list <= getSubClassByParent(klass.getName())
        Para j = 1 Até Tamanho(list) Faça
           OntologyClass child <= list[j]
           Se isEqualsStrings(child.getName(), className) Então
              result = klass;
           Fim Se
           Se result != null Pare
        Fim Para
     Fim Se
     Se result != null Pare
   Fim Para
   Retorno result
Fim
```

Exemplos: *getPlace*(“Paciente”) -> null

getPlace(“município”) -> Municipio

getPlace(“empresas”) -> Empresa

Apesar da coleção *placeMap* não estar declarada neste método, ela contém todos os conceitos descendentes do conceito Lugar.

Agora que os métodos essenciais do CompogeMatch foram apresentados, na seção que segue serão mostrados os roteiros (um passo-a-passo) e algoritmos de detecção de PAs em modelos conceituais.

3.6.1 Detecção do Padrão Organization Hierarquies

A detecção do PA *Organization Hierarquies* pelo método CompogeMatch acontece da seguinte forma: o método cria dois vetores de strings, o primeiro (*Mc*) contendo o nome de cada conceito do modelo conceitual; o segundo (*Og*) com todos os termos da ontologia. A partir desses vetores, o método itera o vetor *Mc* e procura um elemento sintaticamente equivalente no vetor *Og* (utiliza a função *isEqualsStrings* explicada na seção anterior). Quando essa condição for satisfeita, o método testa se o conceito é descendente de Lugar,

se for, o conceito poderá ser considerado candidato ao PA *Organization* Hierarquies. Esse processo de detecção pode ser visualizado na fig. 18:

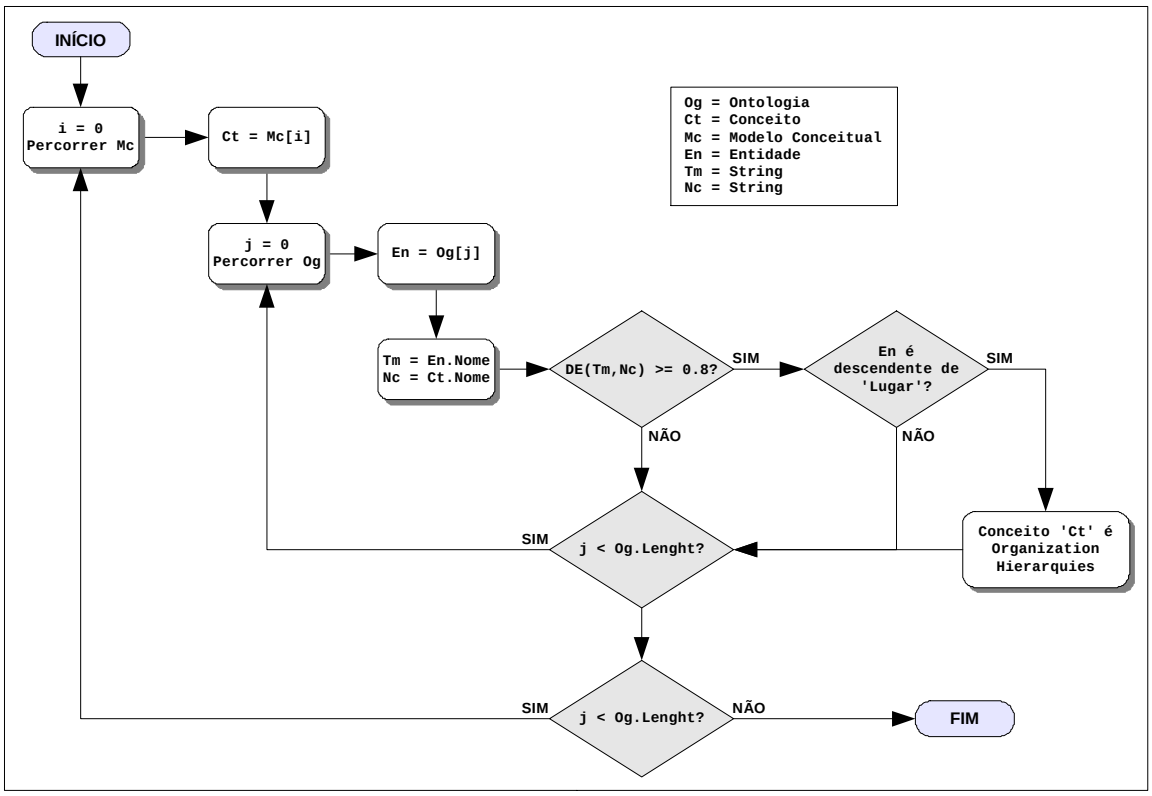


Figura 18: Processo de detecção do padrão Organization Hierarquies

A seguir, detalha-se o fluxo descrito acima, passo-a-passo:

Passo 1: Percorrer por todos os conceitos do modelo.

Passo 2: Aplicar a lógica a seguir que determina a ocorrência do padrão:

Um conceito do modelo é considerado como o *Organization Hierarquies*, quando as seguintes condições forem realizadas:

i) Existir um termo correspondente na ontologia.

ii) Esse termo deve ser subconceito do conceito Lugar, para isso, o método leva em consideração as relações de generalização e especialização.

Passo 3: Retorna ao Passo 1.

Esses passos e condições vislumbram-se no trecho de código da listagem que segue. Este código contém funções apresentadas na seção anterior.

```
Função OrganizationPattern[] detect()
Inicio
   OrganizationPattern[] patterns <= new OrganizationPattern[]
   Para i = 1 Até Tamanho(concepts) Faça
      OntologyClass classPlace <= null
      Concept conceptElement <= concepts[i]
      classmodel <= normalize(conceptElement.getName())
      classPlace <= getPlace(classmodel)
      Se classPlace == null Então
         OntologyClass[] classes = getClassBySimilarity(classmodel)
         Para j = 1 Até Tamanho(classes) Faça
            OntologyClass classSimilarElement <= classes[j]
            classPlace = getPlace(classSimilarElement.getName())
            Se classPlace != null) Então
               Pare
            Fim Se
         Fim Para
      Fim Se
      Se classPlace != null Então
        Patterns[k] <= new OrganizationPattern(conceptElement)
      Fim Se
    Fim Para
    Retorno patterns
Fim
```

A demonstração de como o padrão *Party* é detectado pelo método se desenvolverá na próxima seção.

3.6.2 Detecção do Padrão *Party*

O processo de detecção do padrão *Party* acontece de forma semelhante a do padrão *Organization Hierarquies*. A diferença é que nesta detecção o conceito ascendente será o conceito Pessoa. Abaixo seguem as tarefas, passo-a-passo, para detecção do padrão *Party*:

Passo 1: Percorrer cada conceito do modelo conceitual.

Passo 2: Verificar se é um conceito *Party*, a partir do nome de cada conceito.

Para um conceito do modelo ser considerado *Party,* deve-se satisfazer as seguintes condições:

i) Existir um termo correspondente na ontologia

ii) Esse termo deve se descendente do conceito Pessoa, para isso, o método leva em consideração as relações de generalização e especialização.

Passo 3: Retornar ao Passo 1.

Satisfazendo, conjuntamente, as condições do passo 2, o método considera o conceito como padrão *Party*. Para isso, é imprescindível que o nome do conceito esteja presente na ontologia com conceito descendente do conceito Pessoa, do contrário nenhuma conclusão sobre o PA correspondente poderá ser feita.

A próxima seção apresenta o processo de detecção do PA Contract, que envolve uma quantidade maior de passos para detecção dos PAs *Party* e *Organization Hierarquies*.

3.6.3 Detecção do Padrão *Contract*

A detecção do padrão *Contract*, se comparada com a detecção dos padrões *Party* e *Organization Hierarquies*, se mostra mais complexa.

A seguir, colocam-se em evidência os passos para realizar a detecção do padrão *Contract*:

Passo 1: Para realizar a detecção do padrão *Contract*, o método percorre os conceitos do modelo conceitual combinando-os de três em três. Essa combinação leva em consideração as associações, ou seja, o método percorre os conceitos que estão diretamente associados. Dado um modelo conceitual que segue a forma do modelo apresentado na fig. 21, o método irá iterar da seguinte forma (A,B,C) (B,C,F) (B,C,D) (C,F,D) (C,D,E). Repara-se que, os conceitos precisam ter uma associação direta, pois as associações por transitividade como (A,C,F) ou (A,C,D) não serão combinadas e consequentemente iteradas. As seguir uma lista com as iterações do modelo da fig. 19:

Primeira Iteração: (A,B,C)

Segunda Iteração: (B,C,F)

Terceira Iteração: (B,C,D)

Quarta Iteração: (C,F,D)

Quinta Iteração: (C,D,E)

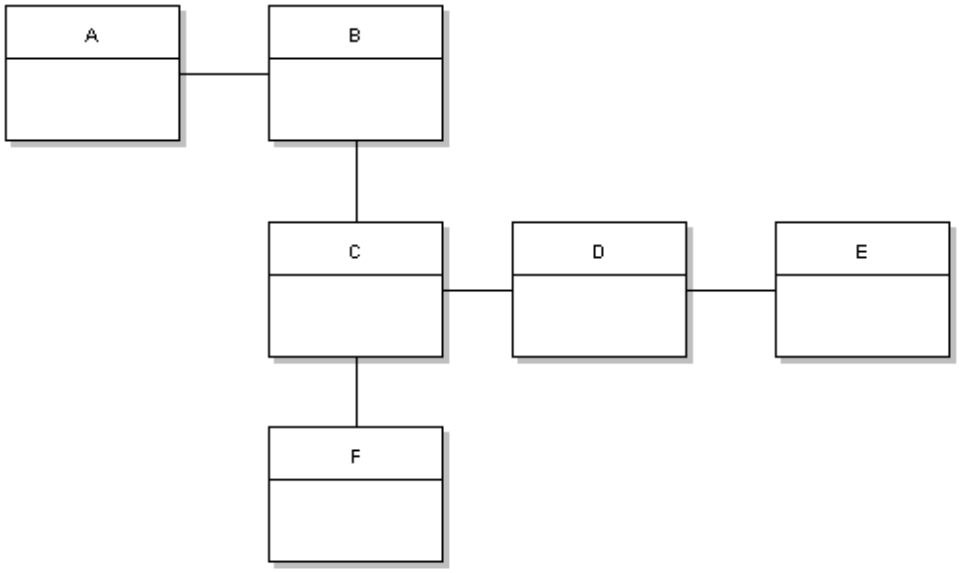


Figura 19: Como o método CompogeMatch permuta os conceitos

A seguir são apresentados os passos no processo de detecção do padrão *Contract*. Os passos apresentados usam como base os conceitos da primeira iteração (A,B,C), no entanto, estes passos podem ser generalizados para todas as cinco iterações do modelo.

Passo 2: Em cada tríplice de conceitos permutados, verificar se o primeiro conceito ou o último (‘A‘ ou ‘C‘) é *Party*. Assume-se então que, para detectar o padrão *Contract*, uma ocorrência do *Party* deverá ocorrer primeiramente. Se essa condição for verdadeira, o método segue para o passo 3, caso contrário, segue para a próxima iteração no “passo 1”;

Passo 3: Se o conceito ‘A‘ for *Party*, o método verifica se o conceito ‘C‘ tem um conceito correspondente na ontologia descendente de Objeto. Porém, se o conceito ‘A‘ não for *Party*, mas sim o conceito ‘C‘, o método fará uma inversão. Confere se existe um conceito correspondente na ontologia descendente de Objeto para o conceito ‘A‘. Ocorrendo a hipótese de nenhum dos dois conceitos, ‘A‘ ou ‘C‘, forem *Party*, o método vai

para a próxima iteração no passo 1 e não aponta a tríplice (A,B,C) como candidata ao PA *Contract*.

Passo 4: Se o Passo 3 for verdadeiro, o método verifica se existe uma associação para muitos (associação N) de “A -> B”;

Passo 5: Se o passo 4 for verdadeiro, o método verifica se existe uma associação para 1 (associação 1) de “B -> A”;

Passo 6: Se o passo 5 for verdadeiro, o método verifica se existe uma associação para N de “B -> C”;

Passo 7: Se o passo 6 for verdadeiro, o método verifica se existe uma associação para 1 de “C -> B”;

Passo 8: Se o passo 7 for verdadeiro, o método considera a tríplice “A, B e C” como conceitos candidatos ao padrão *Contract*.

Passo 9: O método vai para a próxima iteração.

A fig. 20 mostra o padrão *Contract* e suas associações.

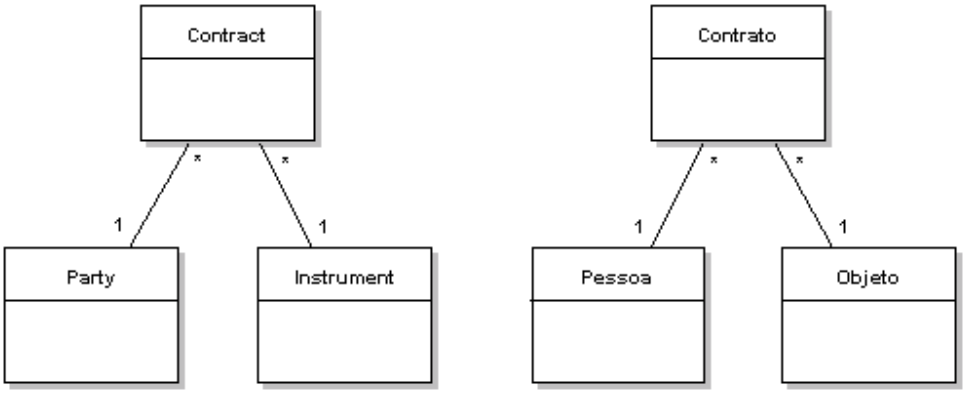


Figura 20: Associações para caracterizar o padrão Contract

Os passos acima podem ser visualizados no código em linguagem Java na seção de anexos.

As seções anteriores tiveram como propósito apresentar as etapas e funções do framework do CompogeMatch bem como os algoritmos de sua implementação. A seção seguinte apresenta como utilizá-lo em um processo de recuperação.

3.7 MODELO DE RECUPERAÇÃO DE COMPONENTES ATRAVÉS DOS PAs

Tendo sido visto como o método detecta a ocorrência de PAs em modelos conceituais, esta seção apresenta um modelo de recuperação de modelos conceituais e componentes softwares.

Um ponto que precisa ser observado aqui, é a possibilidade de utilizar o método CompogeMatch para procurar modelos conceituais e componentes de software em bibliotecas digitais utilizando os PAs. A idéia central disso é a partir de um PA encontrar todos os modelos conceituais que contenham este padrão. Isso poderá trazer mais resultados que uma busca direta utilizando com o filtro uma palavra-chave, o que será apresentado na seção de experimentos.

O modelo de recuperação proposto poderá ser utilizá-lo de duas formas: na primeira o usuário submete um PA e método recupera todos os modelos conceituais da biblioteca que tenha este PA. Uma observação a esta forma é que o usuário deverá conhecer os padrões de análise, o que torna essa forma um pouco difícil de ser utilizada; a segunda forma é submeter um modelo conceitual onde o método deverá executar a recuperação em dois passos: 1) detectar quais os PAs existentes no modelo submetido como filtro; 2) a partir dos padrões de análise detectados, recuperar da biblioteca digital os modelos conceituais que contenham esses PAs. Observe que a segunda forma abstrai a necessidade de o usuário conhecer os padrões de análise para a realização da busca.

A fig. 21 apresenta um fragmento de modelo conceitual típico de um sistema de locadora de veículos. Este fragmento poderia ser um registro de uma biblioteca digital de modelos e componentes.

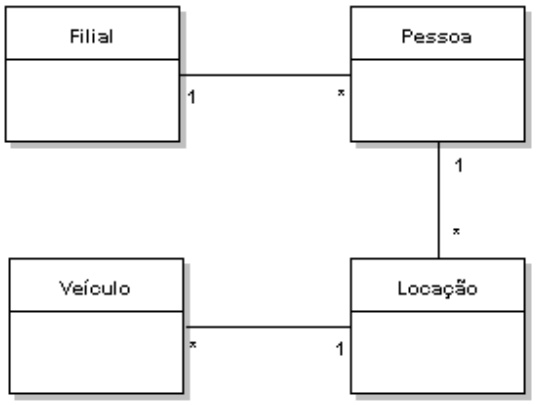


Figura 21: Modelo Conceitual de locadora de veículos (exemplo)

No processo de recuperação por palavra-chave, uma busca bem sucedida somente ocorreria se as palavras especificadas no filtro fossem idênticas aos nomes das classes, por exemplo: “filial”, “pessoa”, “locação”, “veículo”. Uma busca utilizando uma palavra sinônima, como “automóvel”, não produziria um resultado bem sucedido ao modelo da fig. 21, a não ser que houvesse um mapeamento entre os termos equivalentes através de um léxico.

Dados os modelos da fig. 21 o método aponta a ocorrência dos PAs *Party* e *Organization Hierarquies*, como mostra a fig. 22:

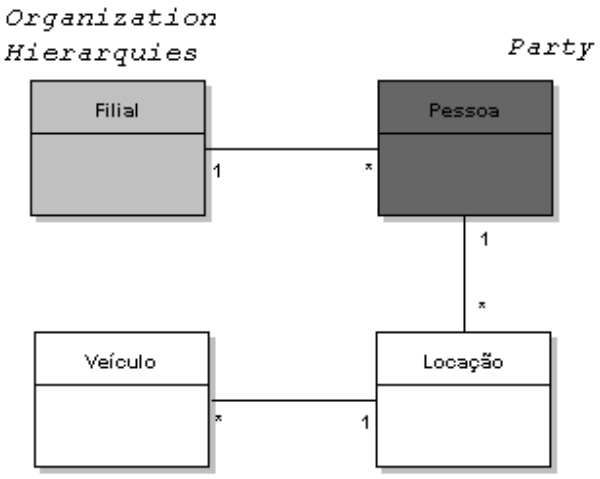


Figura 22: PAs detectados pelo método

Essa detecção de ocorrência foi possível em função da existência de termos correspondentes na ontologia e suas relações de herança definidas de forma correta, conforme explicado nas seções 3.5.1 e 3.5.2.

3.8 OUTROS MODELOS CONCEITUAIS E PADRÕES DETECTADOS

Para melhor ilustrar o método, a seguir são apresentados alguns modelos conceituais que foram submetidos ao processo de detecção:

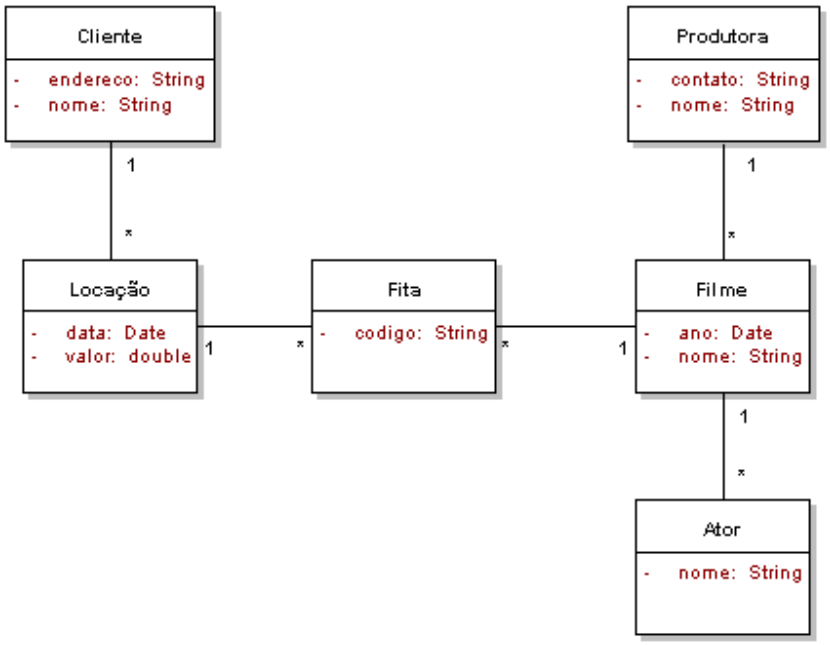


Figura 23: Modelo Conceitual de um sistema de vídeo locadora

Padrões detectados na figura 23:

Party: Cliente e Ator

Organization Hierarquies: nenhum

Contract: (Cliente, Locação, Fita)

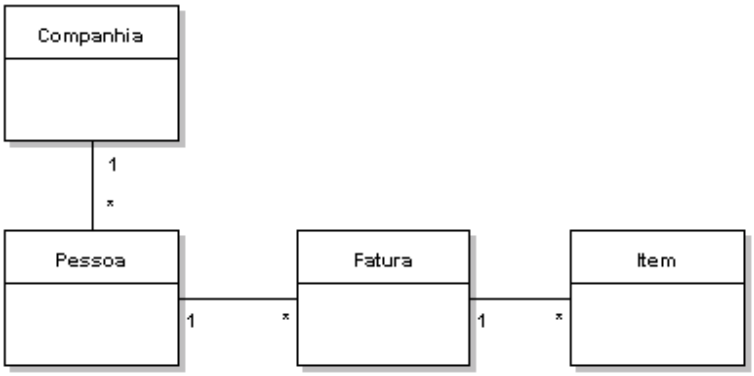


Figura 24: Modelo Conceitual de um sistema de fatura

Padrões detectados na figura 24:

Party: Pessoa

Organization Hierarquies: Companhia

Contract: (Pessoa, Fatura, Item)

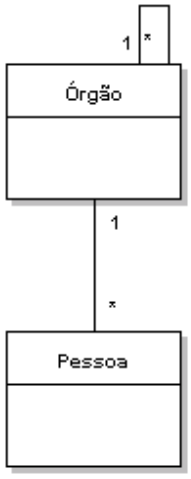


Figura 25: Parte de um modelo conceitual de um sistema de controle de processos

Padrões detectados na figura 25:

Party: Pessoa

Organization Hierarquies: Órgão

Contract: nenhum

A partir dessa detecção, é possível realizar uma recuperação de modelos conceituais que contenham padrões de análise.

No modelo de recuperação proposto uma busca utilizando como índice o padrão *Contract* apresentaria os modelos das fig. 23 e 24, já que, os modelos das fig. 22 e 25 não apresentaram a ocorrência do padrão *Contract.*

Os modelos conceituais apresentados nesta seção não estão completos e foram simplificados para facilitar a compreensão das idéias deste trabalho. Além disso, os atributos foram suprimidos, pois não influenciam no método de detecção dos PAs correntemente considerados.

Este capítulo tratou de como o método CompogeMatch realiza a detecção de padrões de análise em modelos conceituais utilizando ontologias. No capítulo seguinte mostra-se um estudo de caso e os resultados obtidos com a aplicação deste método em modelos conceituais reais.

4 EXPERIMENTOS E RESULTADOS

Este capítulo aponta os experimentos realizados, a fim de mostrar os resultados do método aplicado em modelos conceituais de sistemas em produção. Esses artefatos (modelos conceituais e ontologia) utilizados durante esta fase do trabalho, são de sistemas de informação em produção e não fictícios. Tomou-se o devido cuidado, para garantir a eficácia do método sob um ponto de vista prático.

4.1 A ONTOLOGIA

A ontologia utilizada nos experimentos foi confeccionada por especialistas[2] (OntoGovBR, 2007). Esses profissionais não tinham conhecimento dos modelos conceituais que seriam submetidos nestes experimentos, por isso, pode-se dizer que a ontologia utilizada não adquiriu vícios durante o processo de confecção. A ontologia utilizada será chamada de OntoGovBr que é produto do projeto COMPOGE (2006). A OntoGovBr pode ser visualizada completamente no anexo 2. A OntoGovBr foi criada a partir do levantamento do conceitos encontrados nos web-sites dos portais dos governos estaduais.

4.2 OS MODELOS CONCEITUAIS

Os modelos conceituais utilizados nestes experimentos estão organizados nos seguintes pacotes:

a) **erp** – Sistema de Gestão Financeira. Esses modelos fazem parte de um sistema de gestão corporativa.

b) **idesti** – Módulo de Segurança. Link para acesso ao código fonte: avatar.inf.ufsc.br;

c) **open** – Sistemas diversos de código fonte aberto. Link: http://directory.fsf.org/;

d) **ro** – Sistemas de Governo Eletrônico do Estado de Rondônia.

[2] Ontologia utilizada pelo projeto Compoge (http://www.inf.ufsc.br/compoge/) para e-Gov.

e) **sbc** – Sistema de Inscrição nos eventos SBES-SBBD (Simpósio Brasileiro de Engenharia de Software e Banco de Dados), evento da SBC (Sociedade Brasileira da Computação). Código fonte disponível no cvs avatar.inf.ufsc.br;

Os modelos conceituais utilizados neste capítulo são considerados válidos para os experimentos, pois atendem as seguintes condições:

a) São sistemas reais em produção.

b) Os modelos conceituais são inerentes a negócios e governo eletrônico.

c) Como os sistemas não continham uma documentação completa, foi necessário realizar uma engenharia reversa a partir do modelo de dados para chegar ao modelo conceitual.

Um dos objetivos deste estudo de caso era ter acesso aos modelos conceituais das empresas fornecedoras de software para governo eletrônico espalhadas por todo o Brasil, e disponibilizar esses componentes de software (modelos conceituais) no portal Compoge (http://www.inf.ufsc.br/compoge) como meio de divulgar os trabalhos já desenvolvidos nesta área. Assim, quando uma entidade governamental desejasse adquirir componentes de software, utilizaria este método como ferramenta de busca. Apesar das inúmeras vantagens, nenhuma fornecedora teve interesse em divulgar seus trabalhos. Uma suspeita do movito desse desinteresse é a tímida política de compartilhamento e de reuso praticada pelos órgãos governamentais brasileiros. Acredita-se que, com a política incentivadora do uso de software livre adotada pelo governo nos últimos anos, essa situação deverá mudar.

4.3 RESULTADOS DO PROCESSO DE DETECÇÃO

Segue a lista de modelos conceituais submetidos ao método, bem como os resultados atingidos:

Experimento 01 - Modelo **idesti.Seguranca.** PAs encontrados:

Party: Usuário

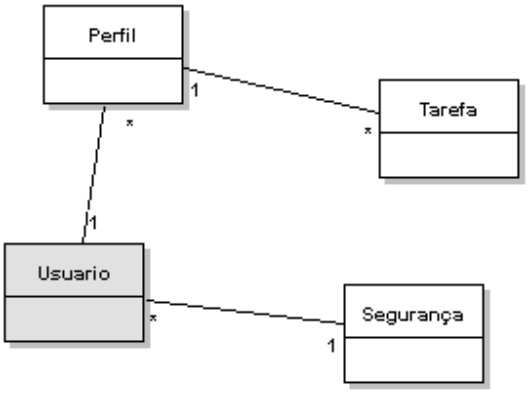


Figura 26: Modelo Conceitual idesti.Segurança

Experimento 02 - Modelo **open.JurisRBAC** e PAs encontrados:

Party: [Advogado]

Contract: [Advogado, Processo, Documento]

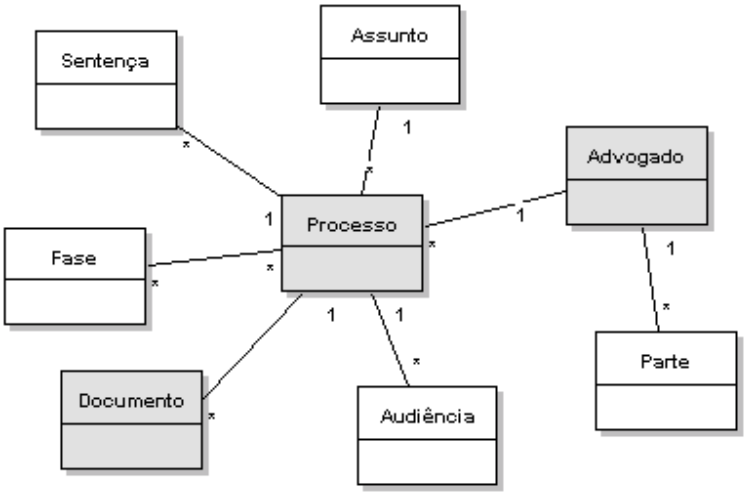


Figura 27: Modelo Conceitual open.JurisRBAC

Experimento 03 - Modelo **ro.Procon** e PAs encontrados:

Party: [Usuário]

Organization Hierarchies: [Unidade, Local]

Contract: [Usuário, Queixa, Objeto]

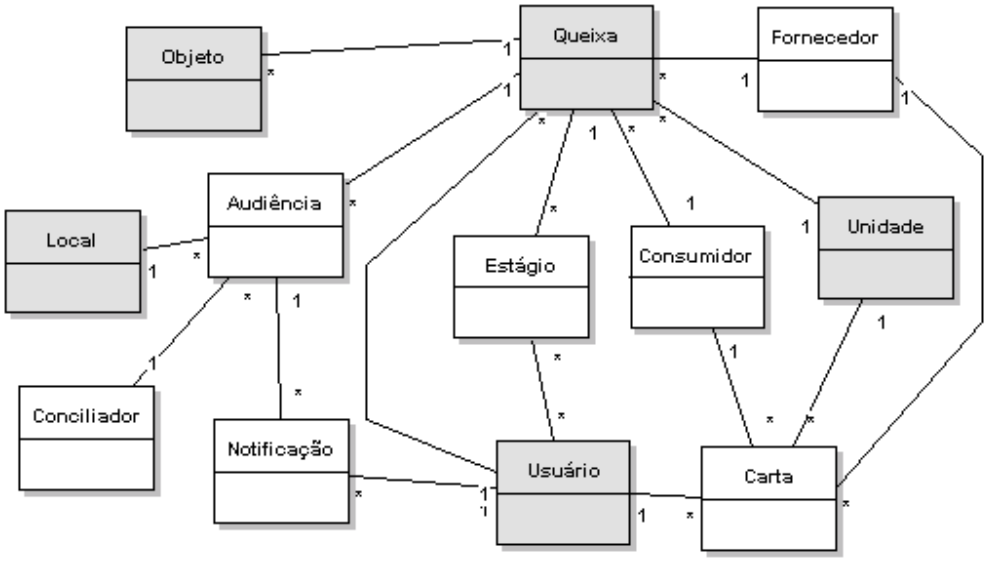


Figura 28: Modelo Conceitual ro.Procon

Pela definição do padrão *Party* observa-se que, o conceito Conciliador, nesse modelo, também deveria ser considerado *Party*. No entanto, isso não ocorreu, por que na ontologia OntoGovBr não foi encontrado o termo Conciliador.

A tabela 02 apresenta um resumo de todos os modelos conceituais submetidos ao método:

Tabela 02: Modelos Conceituais submetido ao CompogeMatch

Modelo Conceitual	No. Conceitos	Party	Organization Hierarchies	Contract
erp.Compra	15	1		
erp.Fidelidade	13	2		
erp.Financeiro	37	2		
erp.Venda	27	2	1	
idest.Seguranca	4	1		
open.AllComerce	15			
open.ClosedShop	11	1		1

open.DotProject	11	1		
open.FreeMRP	21		2	
open.JurisRBAC	8	1		1
open.Mantis	13	1		
open.OsFinancial	33	1	2	
open.WebErp	42		1	
open.Xplanner	9	1		
ro.Cerimonial	11	2	1	
ro.Frota	20	2	1	
ro.Material	13		1	
ro.Ouvidoria	7	1		
ro.Patrimonio	13		1	
ro.Procon	12	1	2	1
ro.Protocolo	10	1		
sbc.InscricaoSbc	11			
Total	**356**	**21**	**12**	**3**

A tabela 03 apresenta os resultados agrupados por pacotes:

Tabela 03: Modelos Conceituais submetidos ao método organizados por pacotes

Pacote	No. Conceitos	Party	Organization Hierarchies	Contract	Total
erp	92	7	1		**8**
idesti	4	1			**1**
open	163	6	5	2	**13**
ro	86	7	6	1	**14**
sbc	11				**0**
Total	**356**	**21**	**12**	**3**	**36**

A tabela mostra que o método detectou um maior número de PÁS para os modelos conceituais do pacote “ro”. Esse pacote contém modelos conceituais ligados à área de e-gov. De todos os PAs detectados, cerca de 39%, fazem parte deste pacote. Essa performance se dá ao fato da OntoGovBR ter sido criada para o domínio de governo

eletrônico, assim, os termos presentes na ontologia relacionam-se mais com termos de e-gov.

4.4 COMPARAÇÃO ENTRE MÉTODOS

Esta seção apresenta uma comparação argumentativa entre os métodos CompogeMatch (apresentado por este trabalho) e GURU (GIRARDI & IBRAHIM, 1995) ROSA (MAAREK, BERRY e KAISER, 1994). Os métodos GURU e ROSA foram utilizados nesta comparação, pois são os mais relevantes encontrados durante a fase de revisão bibliográfica. Todos os métodos de comparação por palavra-chave apontam para estes métodos. Embora já tenham algum tempo de concepção, eles foram utilizados como base comparativa com o CompogeMatch. O quadro 01 apresenta os modelos conceituais utilizados nesta comparação:

Quadro 01: Modelos Conceituais utilizados nas comparações entre métodos

Modelo Conceitual	**Conceitos**
erp.Compra	[Custo, Estoque, Filial, Item, Item de Movimentação, Linha, Marca, Moeda, Perfil, Pessoa, Preço, Produto, Suporte, Movimentação, Forma de Pagamento]
erp.Fidelidade	[Cartão, Classe, Filial, Funcionário, Item, Movimentação, Movimentação Fidelidade, Pessoa, Pontuação, Prêmio, Título, Usuário]
erp.Financeiro	[Agente, Avalista, Baixa de Título, Classe de Comissão, Cobrador, Comissão, Conta Financeira, Conta Pessoal, Correntista, Cotação, Credor, Custo, Fator de Correção, Funcionário, Função, Histórico Bancário, Histórico de Lançamento, Histórico de Titulo, Lançamento Financeiro, Lançamento em Conta Financeira, Lançamento em Livro Caixa, Livro Caixa, Moeda, Movimentação, Ordem de Serviço, Pessoa, Previsão, Remessa, Situação, Tipo de

	Documento, Tipo de Remessa, Titulo, Titulo em Orçamento, Usuário, Vendedor, Vinculo]
erp.Venda	[Avalista, Cobrador, Comissão, Correntista, Credor, Custos, Entrega, Filial, Forma de Pagamento, Funcionário, Grade, Item, Item de Movimentação, Linha, Marca, Moeda, Movimentação, Pessoa, Preço, Produto, Quantidade, Região, Transportador, Titulo, Unidade, Usuário, Vendedor]
idest.Seguranca	[Perfil, Segurança, Tarefa, Usuário]
open.AllComerce	[Carrinho, Click, Cliente, Contatos, Endereço, Evento, Inventário, Mapa, Pedido, Peça, Preço, Preço de Venda, Sessão, Taxa, Zona]
open.ClosedShop	[Administrador, Afiliado, Atributo, Certificado de Presente, Grupo, Maximização, Pedido, Produto, Usuário, Valor, Venda]
open.DotProject	[Arquivo, Companhia, Contato, Departamento, Evento, Fila, Fórum, Permissão, Projeto, Tarefa, Usuário]
open.FreeMRP	[Produto, Componente, Embalagem, Projeto, Processo, Classe, Cor, Unidade, Configuração de Estoque, Material, Colaborador, Acesso, Permissão, Movimento, Estoque, Local, Ordem de Produção, Pedido de Entrada, Fornecedor, Cliente, Pedido de Saída]
open.JurisRBAC	[Advogado, Assunto, Audiência, Processo, Documento, Fase, Parte, Sentença]
open.Mantis	[Erro, Histórico, Monitor, Arquivo, Texto, Anotação, Noticias, Categoria, Projeto, Financiador, Perfil, Usuário, Versão]
open.OsFinancial	[Recurso, País, Conta, Contato, Bem, Pedido, Banco, Credor, Moeda, Devedor, Banco DNL, Fonte, Funcionário, Evento, Fluxo, Grupo, Projeto, Local, Fabricante, Pagamento, Período, Perfil, Estoque, Ano, Transação de Estoque, Tarefa, Ação, Imposto, Total, Transação, Tipo, Unidade, Usuário]

open.WebErp	[Grupo de Conta, Seção de Conta, Área, Conta, Transação Bancária, Gráfico, Contato, Moeda, Devedor, Imposto, Tipo de Cliente, Local, Estoque, Transferência, Período, Preço, Compra, Pedido de Compra, Relatório, Análise, Vendedor, Entregador, Movimentação, Fornecedor, Contrato, Tipo, Balanço, Centro, Bucket, Categoria, Tipo de Venda, Razão de Débito, Transação de Débito, Taxas de Débito, Recolhedor, Desconto, Custo de Frota, Província, Pedido de Venda, Carregamento, Custo de Entrega, Termos de pagamento]
open.Xplanner	[Projeto, Estoria, Iteração, Tarefa, Pessoa, Perfil, Permissão, Nota, Tempo]
ro.Cerimonial	[Usuário, Convite, Unidade, Servidor, Sistema, Tarefa, Papel, Endereço, Pessoa, Promovedor, Órgão]
ro.Frota	[Tarefa, Ocorrência, Papel, Servidor, Unidade, Sistema, Combustível, Item de Movimento, Peça, Movimento, Marca, Item de Serviço, Estoque, Serviço, Fornecedor, Usuário, Pessoa, Hodômetro, Depositário, Maquinário]
ro.Material	[Doença, Movimento, Medicamento, Papel, Paciente Tratamento, Pedido, Médico, Tarefa, Material, Unidade, Leito, Item]
ro.Ouvidoria	[Classe, Providência, Denunciante, Status, Relato, Tipo, Fonte]
ro.Patrimonio	[Componente, Depreciação, Item de Movimentação, Imóvel, Veículo, Servidor, Bem, Grupo, Unidade, Categoria, Compra, Item de Compra, Movimentação]
ro.Procon	[Fornecedor, Estágio, Carta, Usuário, Conciliador, Notificação, Consumidor, Audiência, Local, Objeto, Queixa, Unidade]
ro.Protocolo	[Usuário, Movimentação, Departamento, Secretaria, Remessa, Documento, Processo, Anexo, Arquivo Morto, Assunto]
sbc.InscricaoSbc	[Participante, Inscrição, Boleto, Evento, Produto, Preço,

	Horário, Categoria, Receptivo, Voo, Hospedagem]

Considerando que os modelos apresentados no quadro 01 compõem uma biblioteca digital de componentes e modelos conceituais de software, a seguir são realizadas comparações entre os métodos GURU e ROSA e CompogeMatch.

Antes de apresentar os resultados das comparações, se apresentam como os métodos executam suas buscas particularmente. Nestas comparações todos os métodos recebem como filtro uma palavra-chave e executam as seguintes atividades:

Métodos GURU e ROSA: esses métodos fazem uma busca sintática e procuram no banco de dados por conceitos que tenham equivalência com o filtro passado. Por exemplo, se o filtro for uma palavra como Professor, ambos os métodos tratão como resultados todos os conceitos que tenham em seu nome a string “Professor”.

Método CompogeMatch: o modelo de recuperação utilizado nas comparações que seguem se dá em dois passos:

Passo 1: Determina o PA que a palavra-chave (filtro) corresponde. Para isso, procura na ontologia qual conceito corresponde à palavra-chave. Se esta for correspondente ao padrão *Party*, por exemplo, sugere-se que o filtro da busca seja pelo padrão *Party.* Outra hipótese é se a palavra-chave corresponde ao conceito Lugar, sugere-se neste caso, que a busca seja feita utilizando como filtro o padrão *Organization Hierarquies.*

Passo 2: Encontrado o PA (passo 1) que corresponda à palavra-chave, o método recupera todos os modelos conceituais que o contenha.

Portanto, o comportamento do método CompogeMatch em um processo de recuperação se dá em duas fases: primeiro define qual é o padrão de análise do filtro especificado, e segundo encontra todos os modelos que contém este padrão de análise.

Apresentadas as formas de como os métodos (GURU, ROSA e CompogeMatch) realizam suas buscas, são apresentadas a seguir algumas comparações**.**

Primeira comparação: Passando como filtro a palavra-chave “Cliente” qual seria o resultado desta busca?

Quadro 02: Resultado da comparação, palavra utilizada: “Cliente”

Método	**Modelo Conceitual e Conceitos encontrados**
GURU	open.AllComerce [Cliente, Tipo de Cliente] open.FreeMRP [Cliente, Tipo de Cliente] open.WebErp [Cliente, Tipo de Cliente]
ROSA	open.AllComerce [Cliente, Tipo de Cliente] open.FreeMRP [Cliente, Tipo de Cliente] open.WebErp[Cliente, Tipo de Cliente]
CompogeMatch	erp.Compra [Pessoa] erp.Fidelidade [Funcionário, Pessoa, Usuário] erp.Financeiro [Agente, Avalista, Cobrador, Correntista, Credor, Funcionário, Pessoa, Usuário, Vendedor] erp.Venda [Avalista, Cobrador, Correntista, Credor, Pessoa, Transportador, Usuário, Vendedor] idest.Seguranca [Usuário] open.AllComerce [Cliente] open.ClosedShop [Administrador, Afiliado, Usuário] open.DotProject[Usuário] open.FreeMRP [Colaborador, Fornecedor, Cliente] open.JurisRBAC [Advogado, Parte] open.Mantis [Financiador, Usuário] open.OsFinancial [Credor, Devedor, Funcionário, Fabricante, Usuário] open.WebErp [Devedor, Vendedor, Entregador, Fornecedor] open.Xplanner [Pessoa] ro.Cerimonial [Usuário, Servidor, Pessoa, Promovedor] ro.Frota [Servidor, Fornecedor, Usuário, Pessoa, Depositário] ro.Material [Paciente, Médico] ro.Ouvidoria [Denunciante]

	ro.Patrimonio [Servidor] ro.Procon [Fornecedor, Usuário, Conciliador, Consumidor] ro.Protocolo [Usuário] sbc.InscricaoSbc [Participante]

Os resultados obtidos nos métodos GURU e ROSA são bem previsíveis, isso porque a busca retorna conceitos onde o nome corresponda ao filtro especificado, neste caso Cliente. Como já mencionado, esses métodos não saberiam distinguir palavras sinônimas, como por exemplo, Cliente e Consumidor, conceitos estes que poderiam ser considerados equivalentes. Uma hipótese para melhorar a eficácia destes métodos (GURU e ROSA) é adotar um Thesaurus (PICARD, 1995) indicando a relação de palavras fortemente associadas. Ainda assim, suspeita-se que o comportamento não chegará aos resultados obtidos com o uso de ontologias. Uma sugestão de trabalho futuro é apresentada no capítulo de conclusão deste.

Já o método CompogeMatch encontrou um maior número de modelos e conceitos, porque com a ajuda da ontologia e os padrões de análise foi possível considerar que os conceitos: Cliente, Consumidor, Pessoa, Usuário, bem como os demais apresentados no quadro 02 são conceitos *Party*. Assim, o método trouxe todos os modelos conceituais que continham o padrão *Party*. Uma questão que pode ser contestada é a relevância dos resultados apresentados pelo CompogeMatch. Uma proposta de investigação desta relevância é apresentada como trabalho futuro. O que se pode afirmar até aqui é que a busca por padrões pode trazer resultados mais genéricos do que o que está se procurando.

Segunda comparação: Passando como filtro a palavra chave “Venda” qual seria o resultado dessa busca?

Quadro 03: Resultado da comparação, palavra utilizada: “Venda”

Método	Modelo Conceitual
GURU	open.ClosedShop [Venda, Preço de Venda] open.AllComerce [Venda, Preço de Venda]
ROSA	open.ClosedShop [Venda, Preço de Venda] open.AllComerce [Venda, Preço de Venda]
CompogeMatch	erp.Compra [Pessoa -> Movimentação -> Item de Movimentação] erp.Venda [Pessoa -> Movimentação -> Item de Movimentação] open.ClosedShop [Usuário -> Venda -> Produto] open.FreeMRP [[Cliente -> Pedido de Saída -> Produto], [Fornecedor -> Pedido de Entrada -> Produto]] ro.Frota [Pessoa -> Movimento -> Item de Movimento] ro.Material [Paciente -> Tratamento -> Medicamento] ro.Patrimonio [Servidor -> Compra -> Item de Compra] ro.Procon [Consumidor -> Queixa -> Objeto]

Novamente o método CompogeMatch localizou mais resultados na busca, em função do seu modelo de recuperação. Primeiro detectou que a palavra “Venda” é integrante do padrão *Contract*, e logo após, apresentou como resultado todos os modelos conceituais que continham o padrão *Contract.*

5 CONCLUSÃO

Por fim, passa-se a apresentar as considerações deste trabalho, bem como uma síntese dos objetivos propostos, que foram ou não alcançados, no decorrer desta pesquisa. Também são apresentadas algumas sugestões de trabalhos futuros que não foram contemplados nesta pesquisa.

Busca-se com este trabalho apresentar um método (CompogeMatch) de detecção de padrões de análise (PAs) (FOWLER, 1997) em modelos conceituais, utilizando ontologias. Espera-se que esse método, após ser aprimorado, possa ser utilizado em bibliotecas de componentes (principalmente na área de governo eletrônico) para recuperação de modelos conceituais a partir de PAs.

A hipótese de melhorar os resultados das buscas em bibliotecas digitais utilizando como índices os PAs encontrados dentro do modelo conceitual foi definida com a intenção de reduzir o tempo de busca, além de apresentar uma alternativa às buscas por palavras-chave.

A partir disso, foram definidos os elementos necessários para possibilitar essa detecção: o modelo conceitual, os padrões de análise (nem todos são detectados neste trabalho), e, um artefato que define o significado das coisas. Daí surgiu a necessidade de um léxico ou outra ferramenta de mapeamento de significados, sendo que para esse trabalho foi utilizada uma ontologia.

A ontologia desempenha no método o papel fundamental de representação de conhecimento. Como em redes semânticas, a ontologia OntoGovBR (OntoGovBR, 2007) resolve impasses como: a) definição de sinônimos dentro de um contexto; b) estabelecer relações entre conceitos (“é um”, “tem um”). Esses recursos foram essenciais e suficientes para a elaboração do CompogeMatch

Logo após define-se, como objetivo específico deste trabalho, a linguagem de representação desses elementos, XML/XMI, juntamente com os critérios de detecção dos PA’s, para assim, escrever os algoritmos necessários para tornar real e verificar todas essas idéias.

O capítulo de experimentos e estudo de caso, tinha como objetivo inicial a execução deste método com base em modelos conceituais de todas as empresas que fornecem software para governo eletrônico e que estejam filiadas à ABEP (Associação Brasileira de Entidades Estaduais de Tecnologia de Informação e Comunicação). No entanto, esses modelos não foram disponibilizados pelas referidas empresas, sendo assim, durante a elaboração deste trabalho, aconteceu o I WBCSGE (Workshop Brasileiro de Compartilhamento de Componentes para Governo Eletrônico) do projeto Compoge, reunindo profissionais e acadêmicos de todo o Brasil. Discutiram-se os motivos do baixo reaproveitamento de componentes de software entre as entidades governamentais, bem como se apresentaram alternativas de como resolver esse problema.

A deficiência no reuso de componentes gera gastos astronômicos (FISL, 2006) (GOVERNO, 2007) para os cofres públicos. Dentre outras pesquisas, apresentou-se durante o evento, o método CompogeMatch (NASCIMENTO & WAZLAWICK, 2007), como uma ferramenta que auxilia as entidades govenamentais na redução de custos. Uma vez que permite auxiliar no processo de recuperação de componentes de software (modelos conceituais) em bibliotecas digitais.

Para executar o CompogeMatch e verificar o resultado desta pesquisa, basta acessar portal http://www.inf.ufsc.br/compoge e submeter um modelo conceitual em formato XMI que o método listará os padrões encontrados.

5.1 LIMITAÇÕES DO TRABALHO / TRABALHOS FUTUROS

Como parte de um processo metodológico esse trabalho apresenta algumas limitações, algumas identificadas na fase de projeto e outras descobertas durante o desenvolvimento e experimentos. Por isso, espera-se dar continuidade nesta pesquisa.

Todo o código fonte das idéias aqui apresentadas está disponível no portal Compoge (http://www.inf.ufsc.br/compoge) além da seção de anexos.

Segue uma lista de idéias que podem complementar essa pesquisa:

a) Outros critérios para detecção dos PA's que não foram abordados nessa pesquisa, vez que FOWLER (1997) traz aproximadamente cem padrões, enquanto essa pesquisa contemplou apenas três (*Party, Organization Hierarquies e Contract)*;

b) Com freqüência se encontra o modelo físico dos dados, dessa forma uma engenharia reversa se fez necessário para execução do CompogeMatch. Sabe-se que, não é novidade os trabalhos de engenharia reversa. O que se sugere é uma simplificação dos modelos de engenharia específicos para este método, sem a necessidade de reverter atributos, pois como foi mostrado durante a explanação do método, os atributos não são relevantes para a detecção dos PAs;

c) Verificar a relevância dos modelos conceituais obtidos no processo de recuperação, uma vez que a busca por PAs é mais genérica genérica do que uma busca direta por termos correspondentes.

d) No processo de recuperação desta pesquisa, o método CompogeMatch foi comparado com GURU (GIRARDI & IBRAHIM, 1995) ROSA (MAAREK, BERRY e KAISER, 1994), uma hipótese que surgiu nesta fase foi a possibilidade de combinar estes métodos com um Thesaurus (PICARD, 1995) para melhorar a eficácia no processo de recuperação.

e) Adaptar o CompogeMatch para aceitar outros formatos de representação de modelos conceituais, pois na versão atual admitem-se somente modelos que estejam em formato XMI.

f) Minerar modelos conceituais e montar uma biblioteca digital de componentes de software de governo eletrônico ou áreas afins e permitir a recuperação através dos métodos apresentados nessa pesquisa;.

g) Levantar a redução efetiva de custo para com as entidades governamentais na utilização de uma biblioteca digital de componentes;

As idéias sugeridas acima procuram incrementar ainda mais esta pesquisa com o objetivo de amadurecer o método CompogeMatch, tornando-o uma ferramenta útil no processo de busca e recuperação de componentes de software.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AMIN Rushikesh, CINNÉIDE Ó Mel, e VEALE Tony (2004): LASER: A Lexical Approach to Analogy in Software Reuse, 26TH Int. Conf. on Software Engineering, Scotland.

BRAGA M. M. Regina, MATTOSO Marta e WERNER M. L. Cláudia (2001): The Use of Mediation and Ontology Technologies for Software Component Information Retrieval ACM Symposium on Software Reusability, SSR2001, Toronto, Canada, pp.19-28.

BUCHLI Frank e NIERSTRASZ Oscar (2003): Detecting Software Patterns using Formal Concept Analysis, Diploma thesis, University of Bern. pp. 1-31

CHANDRESEKARAN B. JOSEPHSON J.R. e Benjamins V.R. (1999): What are ontologies, and why do we need them?, IEEE Intelligent Systems, pp. 20-26.

COTA I. Renata, MENEZES S. Crediné e FALBO A. Rircado (2004): Modelagem Organizacional Utilizando Ontologias e Padrões de Análise. VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software - IDEAS'2004, pp. 56-67, Arequipa, Perú, Maio 2004.

FERNANDEZ B. Eduardo e YUAN Xiaohong (2000): Semantic analysis patterns, 19th Int. Conf. on Conceptual Modeling ER2000, pp. 183-195.

FISL (2006): Software Público Brasileiro vai compartilhar soluções desenvolvidas pelos estados e governo federal. 7º. Fórum Internacional de Software Livre, Porto Alegre. Disponível em: http://fisl.softwarelivre.org/7.0/www/indexd5dd.html?q=node/195 Acesso em: 15/12/2007.

FOWLER Martin (1997): Analysis Patterns: Reusable Object Models, Addison Wesley.

GAMMA Erich, HELM Richard, JOHNSON Ralph, VLISSIDES John (1993): Designer Patterns: Abstraction and Reuse of Object-Oriented Design. In European Conference on Object-Oriented Programming Proceedings (ECOOP'93), volume 707 of Lecture Notes in Computer Science. Springer-Verlag, July 1993. pp-406-431

GILLELAND Michael (2007): Levenshtein Distance, in Three Flavors. Disponível em: http://www.merriampark.com/ld.htm Acesso em 15/06/2007.

GIRARDI M. R. e IBRAHIM B (1995): Using English to retrieve software. The Journal of Systems and Software, 30(3):249-270, September 1995.

GÓMEZ-PÉREZ Asunción, RAMÍREZ Jaime e VILLAZÓN-TERRAZAS Boris (2007): Reusing Human Resources Management Standards for Employment Services. Methodology for Reusing Human Resources Management Standards (SEKE), 2007. pp-280-285.

GOVERNO (2007): Governo fala sobre software livre: Disponível em: https://www.governoeletronico.gov.br/noticias-e-eventos/noticias/serpropgenoticia.2007-04-15.5228678428/?searchterm=cacic Acesso em: 15/12/2007.

GUARINO Nicola, GIARETTA Pierdaniele (1995): Ontologies and Knowledge Bases Towards a Terminological Clarification. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam: pp. 25-32

GUARINO Nicola (1997): Understanding, building, and using ontologies: a commentary to using explicit ontologies. Iin KBS development. International Journal of Human and Computer Studies, v. 46, p. 293-310.

LARMAN Craig (2001): Appying UML and Patterns: An introdution to Object-Oriented Analysis and Design and the Unified Process, 2nd Edition. Prentice Hall

LEVENSHTEIN I. Vladimir (1966): Binary Codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, Vol. 10, No. 8. (1966), pp. 707-710.

MAAREK S. Yoëlle, BERRY M. Daniel e KAISER E. Gail (1994): Guru: Information retrieval for reuse. In Landmark Contribuitions in Software Reuse and Reverse Engineering. Prentice Hall, 1994.

MISHNE Gilad e RIJKE Maarten (2004): Source Code Retrieval using Conceptual Similarity, RIAO 2004, Pittsburgh.

MONTES-Y-GÓMEZ Manuel, LÓPEZ-LÓPEZ Aurélio e GELBUKH Alexander (2000): Information Retrieval with Conceptual Graph Matching, Database and Expert Systems Applications, pp.312-321

NASCIMENTO Kuesley, WAZLAWICK S Raul (2007): Compoge: Detecção de padrões de análise em modelos conceituais para recuperação de componentes. Porto Alegre: SECOP, 2007.

OntoGovBR (2007): OntoGovBR: Ontologia para compartilhamento de componentes para governo eletrônico. Porto Alegre, 2007.

PICARD W. Rosalind (1995): Toward a visual thesaurus, in Springer-Verlag Workshops in Computing, 1995. Pág. 358. http://citeseer.ist.psu.edu/picard95toward.html

REVURI Sandhya, UPADHYAYA Sujatha R e KUMAR P Sreenivasa (2006): Using Domain Ontologies for Efficient Information Retrieval. 13th International Conference on Management of Data, Delhi, India Dezembro, 2006.

RUSSELL Stuart, NORVIG Peter (2004): Inteligência Artificial. Rio de Janeiro: Elsevier, 2004, 2ª. Edição, pp-187-357

SUN (1995) Object-Oriented Programming Concepts, Disponível em: http://java.sun.com/docs/books/tutorial/java/concepts/ Acesso em: 10/01/2006

SCHREIBER Guus, WIELINGA Bob , JANSWEIJER Wouter (1995): The KACTUS View on the 'O' Word. Technical Report, ESPRIT Project 8145 KACTUS, University of Amsterdam, 1995.

WAZLAWICK S. Raul (2004): Análise e projeto de sistemas de informação orientados a objetos Rio de Janeiro: Elsevier, 2004. pp. 32-141

ANEXO 1 – GLOSSÁRIO

COMPOGEMATCH – Método apresentado neste trabalho para detectar padrões de análise em modelos conceituais utilizando ontologias que pode ser usado em buscas por componentes de software em bibliotecas digitais;

CONTRACT – Padrão de análise que representa uma transação envolvendo três elementos: Uma pessoa, uma transação (operação) e um objeto. Exemplo: Uma venda de mercadoria ou um aluguel de um carro. Quando um software precisa implementar uma solução para este tipo de problema, certamente será necessário implementar o padrão de análise *Contract.*

FRAMEWORK – Conjunto de classes de um domínio para auxiliar no desenvolvimento de software e tornar a confecção do software mais eficiente. Frameworks normalmente são limitados a um assunto, por exemplo, framework para gerenciamento da camada de persistência de uma aplicação (Hibernate, JDO);

JAVA – Linguagem de programação orientada a objetos especificada pela SUN na qual este método está implementado;

MODELO CONCEITUAL – Diagrama que representa as informações e suas associações que o sistema/software manipula ou gerencial;

ONTOLOGIA – Artefato utilizado na área da inteligência artificial para representar conhecimento em um domínio específico. É utilizada na ciência da computação para definir conceitos e suas relações. Com uma ontologia é possível, por exemplo, saber que motocicleta e carro de passeio são meios de transporte através de uma relação com um conceito que os defina.

ORGANIZATION HIERARCHIES – Padrão de análise que representa uma unidade organizacional em modo hierárquico (ex. Secretaria de estado, autarquias);

PADRÃO DE ANÁLISE (ANALYSIS PATTERNS) – Um padrão de análise é um conceito adotado quando se tem um problema recorrente onde se pode adotar uma solução padrão ainda na área de análise do projeto. A idéia é a mesma dos padrões de projeto.

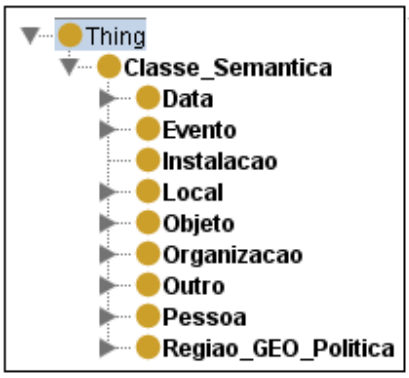
PADRÕES DE PROJETO – Tríade composta por um problema recorrente, uma solução padrão e um nome para facilitar disseminação. No dia-a-dia do desenvolvimento de software os profissionais se deparam com problemas recorrentes, um padrão de projeto é uma solução para este tipo de problema. Com um nome específico para facilitar a divulgação dos conceitos. Exemplos: Factory, Singleton, MVC entre outros;

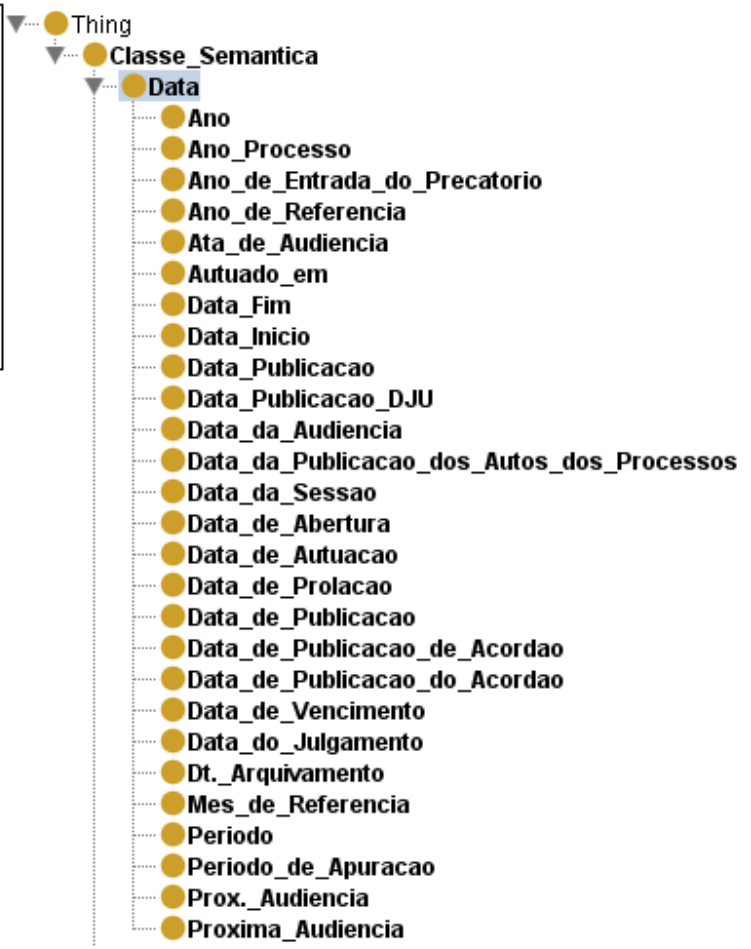
PARTY – Padrão de análise (PA) que representa uma entidade como uma Pessoa. Este padrão está contemplado nesta pesquisa;

SMALLTALK – Uma das primeiras linguagens de programação orientada a objetos da década de 60;

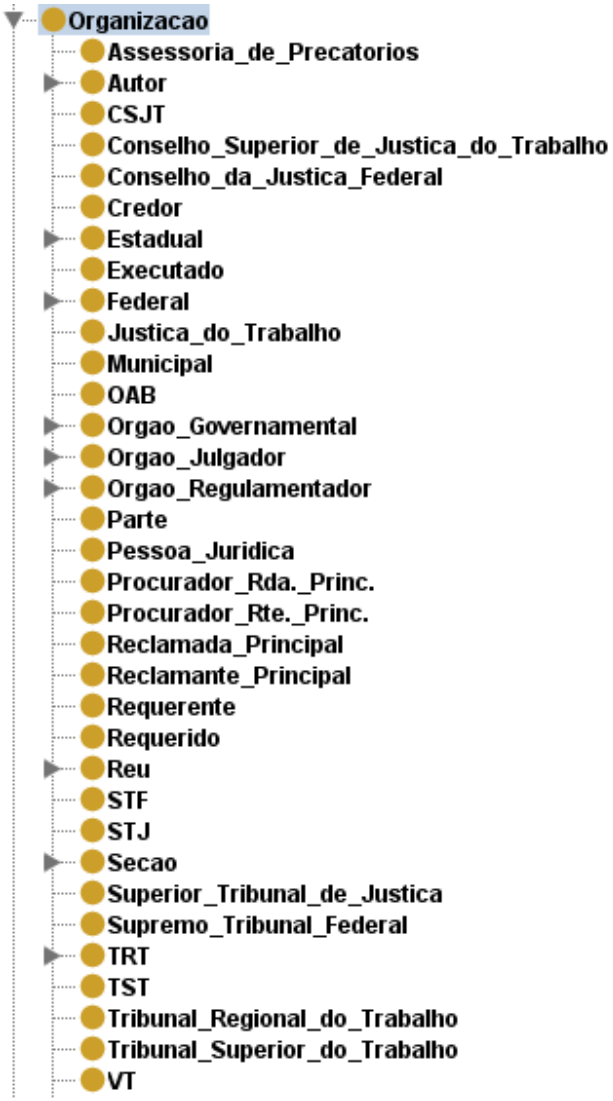
UML – Linguagem de Modelagem Unificada utilizada para descrever processos e artefatos nas fases de análise e projeto de sistemas;

ANEXO 2 - ONTOLOGIA






Evento
Acao
Acidente_do_Trabalho
Acordao
Acordaos
Agravo_Regimental
Alienacao_Fiduciaria
Alteracao_Contratual
Aposentadoria
Arquivamento
Ato_do_TST_Nº
Audiencia
Audiencias_do_Tribunal
Autuacao
Conciliacoes
Concorrencia_Publica
Concurso_Publico
Convite
Decisao
Depositos_Judiciais
Desistencia
Dissidios_Coletivos
Dissidios_Individuais
Embargos_de_Declaracao
Execucao_de_Atos_de_Secretaria
Pracas_e_Leiloes
Pregao_Eletronico
Pregao_Presencial
Provimento_da_Corregedoria
Publicacao
Publicacao_Edital
Publicacao_de_Acordaos
Remetido_ao_Arquivo_Geral
Rito
Sentenca
Objeto
Objeto_Abstrato
Ata
Atas
Classe_Processual
Decreto
Despachos_da_Presidencia
Ementa
Inteiro_Teor
Jurisprudencia
Legislacao
Lei
Licitacao
Nome_da_Parte
Partes_do_Processo
Pauta
Processo
RR/RE/AR/Embargos
Recurso
Recurso_Ordinario
Registro
Tipo_de_Acao_Trabalhista
Tipo_de_Processo
Tipo_de_Sustentacao_Oral


Organizacao
Assessoria_de_Precatorios
Autor
CSJT
Conselho_Superior_de_Justica_do_Trabalho
Conselho_da_Justica_Federal
Credor
Estadual
Executado
Federal
Justica_do_Trabalho
Municipal
OAB
Orgao_Governamental
Orgao_Julgador
Orgao_Regulamentador
Parte
Pessoa_Juridica
Procurador_Rda._Princ.
Procurador_Rte._Princ.
Reclamada_Principal
Reclamante_Principal
Requerente
Requerido
Reu
STF
STJ
Secao
Superior_Tribunal_de_Justica
Supremo_Tribunal_Federal
TRT
TST
Tribunal_Regional_do_Trabalho
Tribunal_Superior_do_Trabalho
VT


Pessoa
Advogado
Autor
Credor
Estagiario
Executado
Idoso
Indio
Juiz
Magistrado
Oficiais_de_Justica
Parte
Perito
Procurador_Rda._Princ.
Procurador_Rte._Princ.
Reclamada_Principal
Reclamado
Reclamante
Reclamante_Principal
Redator
Relator
Representante
Requerente
Requerido
Reu
Revisor
Usuario

ANEXO 3 – FRAMEWORK COMPOGEMATCH

```
package util;
public class NormalizeString {
	static String PTBR = "çÇáéíóúÁÉÍÓÚãõÃÕàèìòùÀÈÌÒÙüÜâêîôûÂÊÎÔÛ";
	static String NORMALIZE = "cCaeiouAEIOUaoAOaeiouAEIOUuUaeiouAEIOU";
	static String TOKENS = " _0123456789!#$@¬§°¢£³²¹%¨&*()_-+=?/;:.>,<~^]}[{´`'\"";
	public static String normalize(String value) {
		String result = "";
		for (int i = 0; i < value.length(); i++) {
			String str = value.substring(i, i+1);
			int pos = PTBR.indexOf(str);
			if (pos != -1) {
				result += NORMALIZE.substring(pos, pos+1);
			} else {
				int pos_tokens = TOKENS.indexOf(str);
				if (pos_tokens != -1) {
					result += "";
				} else {
					result += str;
				}
			}
		}
		return result;
	}
}

package util;
public class LexicalSimilarity {
	public static double similarity(String a, String b) {
		a = a.toUpperCase();
		b = b.toUpperCase();
		double a_len = a.length();
		double b_len = b.length();
		double min_len = Math.min(a_len, b_len);
		Distance ed = new Distance();
		int ed_value = ed.LD(a, b);
		double calc = ((min_len - ed_value) / min_len);
		return Math.max(0, calc);
	}
}
package util;
/*
 * Este código foi retirado no seguinte link:
 *
 * http://www.merriampark.com/ld.htm#JAVA
 *
 * Autores:
 *
 * by Michael Gilleland, Merriam Park Software
 */
public class Distance {
  //****************************
  // Get minimum of three values
  //****************************
  private int Minimum (int a, int b, int c) {
    int mi;
    mi = a;
    if (b < mi) {
      mi = b;
    }
    if (c < mi) {
      mi = c;
    }
    return mi;
  }
```

```
  //****************************
  // Compute Levenshtein distance
  //****************************
  public int LD (String s, String t) {
    int d[][]; // matrix
    int n; // length of s
    int m; // length of t
    int i; // iterates through s
    int j; // iterates through t
    char s_i; // ith character of s
    char t_j; // jth character of t
    int cost; // cost

    // Step 1
    n = s.length ();
    m = t.length ();
    if (n == 0) {
      return m;
    }
    if (m == 0) {
      return n;
    }
    d = new int[n+1][m+1];

    // Step 2
    for (i = 0; i <= n; i++) {
      d[i][0] = i;
    }
    for (j = 0; j <= m; j++) {
      d[0][j] = j;
    }

    // Step 3
    for (i = 1; i <= n; i++) {
      s_i = s.charAt (i - 1);
      // Step 4
      for (j = 1; j <= m; j++) {
        t_j = t.charAt (j - 1);

        // Step 5
        if (s_i == t_j) {
          cost = 0;
        }
        else {
          cost = 1;
        }

        // Step 6
        d[i][j] = Minimum (d[i-1][j]+1, d[i][j-1]+1, d[i-1][j-1] + cost);
      }
    }
    // Step 7
    return d[n][m];
  }
}

package pattern;
public class Pattern {
	private String Modelo;
	public String getModelo() {
		return Modelo;
	}
	public void setModelo(String Modelo) {
		this.Modelo = Modelo;
	}
}
```

```
package pattern;
import model.Concept;
public class PartyPattern extends Pattern {
	private Concept concept;
	public PartyPattern(Concept value) {
		concept = value;
	}
	public Concept getConcept() {
		return concept;
	}
	public void setConcept(Concept concept) {
		this.concept = concept;
	}
}
```

```
package pattern;
import model.Concept;
public class OrganizationPattern extends Pattern {
	private Concept concept;
	public OrganizationPattern(Concept value) {
		this.concept = value;
	}
	public Concept getConcept() {
		return concept;
	}
	public void setConcept(Concept concept) {
		this.concept = concept;
	}
}
```

```
package pattern;
import model.Concept;
public class ContractPattern extends Pattern {
	private Concept person;
	private Concept object;
	private Concept relation;
	public Concept getObject() {
		return object;
	}
	public void setObject(Concept object) {
		this.object = object;
	}
	public Concept getPerson() {
		return person;
	}
	public void setPerson(Concept person) {
		this.person = person;
	}
	public Concept getRelation() {
		return relation;
	}
	public void setRelation(Concept relation) {
		this.relation = relation;
	}
}
```

```
package model;
public class Association {
	private String id;
	private Concept concept;
	private String multiplicity;
	private String roleName;
	public Concept getConcept() {
		return concept;
	}
	public void setConcept(Concept concept) {
		this.concept = concept;
	}
```

```
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getMultiplicity() {
		return multiplicity;
	}
	public void setMultiplicity(String multiplicity) {
		this.multiplicity = multiplicity;
	}
	public String getRoleName() {
		return roleName;
	}
	public void setRoleName(String roleName) {
		this.roleName = roleName;
	}
}
```

```
package model;
import java.util.ArrayList;
public class Concept {
	private String id;
	private String name;
	private ArrayList<Association> associations = new ArrayList<Association>();
	public Concept(String id, String name) {
		super();
		this.id = id;
		this.name = name;
	}
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public ArrayList<Association> getAssociations() {
		return associations;
	}
	public void setAssociations(ArrayList<Association> associations) {
		this.associations = associations;
	}
}
```

```
package model;
import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Namespace;
import org.jdom.input.SAXBuilder;
public class Model {
 private ArrayList<Package> packages = new ArrayList<Package>();
 public Model(String fileName) throws Exception {
 try {
  File file = new File(fileName);
  SAXBuilder sb = new SAXBuilder();
  Document doc = sb.build(file);
  Element rootElement = doc.getRootElement();
```

```
 Namespace ns = rootElement.getNamespace("XML");
 Element contentElement = rootElement.getChild("XMI.content");
 Element UML_ModelElement = contentElement.getChild("Model",  ns);
 Element ownerElement0 = UML_ModelElement.getChild("Namespace.ownedElement",  ns);
 Element PackageElement = ownerElement0.getChild("Package",  ns);
 Element ownerElement1 = PackageElement.getChild("Namespace.ownedElement",  ns);
 List packageList = ownerElement1.getChildren("Package", ns);
 for (Iterator<Element> i = packageList.iterator(); i.hasNext();) {
  Element packageElement = i.next();
  Package bean = new Package(packageElement.getAttributeValue("xmi.id"),
                             packageElement.getAttributeValue("name"));
  packages.add(bean);
  Element ownerElement2 = packageElement.getChild("Namespace.ownedElement",  ns);
  List classList = ownerElement2.getChildren("Class", ns);
  for (Iterator<Element> j = classList.iterator(); j.hasNext();) {
   Element classElement = j.next();
   Concept concept = new Concept(classElement.getAttributeValue("xmi.id"),
                                 classElement.getAttributeValue("name"));
   bean.getConcepts().add(concept);
  }
  List associationList = ownerElement2.getChildren("Association", ns);
  for (Iterator<Element> j = associationList.iterator(); j.hasNext();) {
   Element associationElement = j.next();
   String id = associationElement.getAttributeValue("xmi.id");
   String conceptSourceName = "";
   String conceptTargetName = "";
   String multiplicitySource = "";
   String multiplicityTarget = "";
   List tagList0 = associationElement.getChild("ModelElement.taggedValue",ns)
                                     .getChildren("TaggedValue", ns);
   for (Iterator<Element> k = tagList0.iterator(); k.hasNext();) {
    Element tagElement = k.next();
    String tag = tagElement.getAttributeValue("tag");
    String value = tagElement.getAttributeValue("value");
    if (tag.equalsIgnoreCase("ea_sourceName")) conceptSourceName = value;
    if (tag.equalsIgnoreCase("ea_targetName")) conceptTargetName = value;
   }
   List associationEndList = associationElement.getChild("Association.connection", ns)
                                               .getChildren("AssociationEnd", ns);
   for (Iterator<Element> k = associationEndList.iterator(); k.hasNext();) {
    Element AssociationEndElement = k.next();
    List tagList1 = AssociationEndElement.getChild("ModelElement.taggedValue", ns)
                                         .getChildren("TaggedValue", ns);
    for (Iterator<Element> m = tagList1.iterator(); m.hasNext();) {
     Element tagElement = m.next();
     String value = tagElement.getAttributeValue("value");
     if (value.equalsIgnoreCase("source"))
      multiplicitySource=AssociationEndElement.getAttributeValue("multiplicity");
     if (value.equalsIgnoreCase("target"))
      multiplicityTarget = AssociationEndElement.getAttributeValue("multiplicity");
    }
   }
   Association association1 = new Association();

   association1.setMultiplicity(multiplicityTarget);
   Concept source1 = bean.getConceptByName(conceptSourceName);
   Concept target1 = bean.getConceptByName(conceptTargetName);
   association1.setConcept(target1);
   Association association2 = new Association();
   association2.setMultiplicity(multiplicitySource);
   association2.setConcept(source1);
   source1.getAssociations().add(association1);
   target1.getAssociations().add(association2);
  }
 }
}
catch (Exception e) {
 throw e;
```

```
 }
}
public ArrayList<Package> getPackages() {
 return packages;
}
public String toString() {
 StringBuffer sb = new StringBuffer();
 for (Iterator<Package> i = packages.iterator(); i.hasNext();) {
  Package p = i.next();
   sb.append("Package: ");
   sb.append(p.getName());
   sb.append("[");
   String temp = "";
   for (Iterator j = p.getConcepts().iterator(); j.hasNext();) {
    Concept c = (Concept) j.next();
    sb.append(temp);
    sb.append(c.getName());
    temp = ",";
   }
   sb.append("]");
   sb.append("\n");
   sb.append("Associations: ");
   for (Iterator j = p.getConcepts().iterator(); j.hasNext();) {
    Concept c = (Concept) j.next();
    for (Iterator<Association> k = c.getAssociations().iterator(); k.hasNext();) {
     Association association = k.next();
     sb.append("[");
     sb.append(c.getName() + " -> <"+association.getMultiplicity()+">" +
               association.getConcept().getName());
     sb.append("]");
    }
   }
   sb.append("]");
   sb.append("\n");
  }
  return sb.toString();
 }
}
```

```
package model;
public class OntologyClass {
        private String name;
        private String parentName;
        public OntologyClass(String name, String parentName) {
                super();
                this.name = name;
                this.parentName = parentName;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getParentName() {
                return parentName;
        }
        public void setParentName(String parentName) {
                this.parentName = parentName;
        }
}
```

```
package model;
import java.util.ArrayList;
import java.util.Iterator;
public class Package {
        private String id;
        private String name;
```

```
        private ArrayList<Concept> concepts = new ArrayList<Concept>();
        public Package(String id, String name) {
                super();
                this.id = id;
                this.name = name;
        }
        public String getId() {
                return id;
        }
        public void setId(String id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public ArrayList<Concept> getConcepts() {
                return concepts;
        }
        public Concept getConceptByName(String name) throws Exception {
                for (Iterator<Concept> i = concepts.iterator(); i.hasNext();) {
                        Concept element = i.next();
                        if (element.getName().equalsIgnoreCase(name)) {
                                return element;
                        }
                }
                throw new Exception("Concept not found "+name);
        }
        public void setConcepts(ArrayList<Concept> concepts) {
                this.concepts = concepts;
        }
}

package model;
import java.util.Collection;
import java.util.HashMap;
public class PersonAttribute {
  private static HashMap<String, OntologyClass> classMap =
                    new HashMap<String, OntologyClass>();
  public void add(String key, OntologyClass value) {
  classMap.put(key, value);
 }
 public Collection<OntologyClass> getList() {
        return classMap.values();
 }
}

package control;
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import model.Model;
import model.OntologyClass;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Namespace;
import org.jdom.input.SAXBuilder;
import util.LexicalSimilarity;
import util.NormalizeString;
public class APDetect {
 private static HashMap<String, OntologyClass> classMap
                 = new HashMap<String, OntologyClass>();
```

```
 private static HashMap<String, OntologyClass> personMap
                  = new HashMap<String, OntologyClass>();
 private static HashMap<String, OntologyClass> objectMap
                  = new HashMap<String, OntologyClass>();
 private static HashMap<String, OntologyClass> placeMap
                  = new HashMap<String, OntologyClass>();
 private Model model = null;
 static private APDetect me;

 public static APDetect getInstance(String fileontology, String filemodel)
                        throws Exception {
  if (me == null) me = new APDetect(fileontology, filemodel);
  return me;
}
public static APDetect getInstance() throws Exception {
 if (me == null) throw new Exception("Informe o caminho da ontologia e do model.");
 return me;
}
public void reload(String fileontology, String filemodel) throws Exception {
 me = new APDetect(fileontology, filemodel);
}
private APDetect(String fileontology, String filemodel) throws Exception {
 try {
  loadOntology(fileontology);
  loadModel(filemodel);
 }
 catch (Exception e) {
  throw e;
 }
}
public List<OntologyClass> getClassBySimilarity(String className) {
 ArrayList<OntologyClass> result = new ArrayList<OntologyClass>();
 String s0 = NormalizeString.normalize(className);
 for (Iterator<OntologyClass> i = classMap.values().iterator(); i.hasNext();) {
  OntologyClass klass = i.next();
  String s1 = NormalizeString.normalize(klass.getName());
  if (LexicalSimilarity.similarity(s0, s1) >= 0.8) {
   result.add(klass);
  }
 }
 return result;
}
public List<OntologyClass> getSubClassByParent(String className, boolean inclusive) {
 ArrayList<OntologyClass> result  = new ArrayList<OntologyClass>();
 if (inclusive) result.add(this.getClassByName(className));
 for (Iterator i = classMap.values().iterator(); i.hasNext();) {
  OntologyClass element = (OntologyClass) i.next();
  if (element.getParentName().equalsIgnoreCase(className)) {
   result.addAll(getSubClassByParent(element.getName(), true));
  }
 }
 return result;
}
public OntologyClass getPerson(String className) {
 OntologyClass result = null;
 for (Iterator i = personMap.values().iterator(); i.hasNext();) {
  OntologyClass klass = (OntologyClass) i.next();
  if (isEqualsStrings(klass.getName(), className)) result = klass;
  if (result == null) {
   List<OntologyClass> list = this.getSubClassByParent(klass.getName(), false);
   for (Iterator j = list.iterator(); j.hasNext();) {
    OntologyClass child = (OntologyClass) j.next();
    // se o nome da classe for exatamente igual
    if (isEqualsStrings(child.getName(), className)) result = klass;
    if (result != null) break;
   }
  }
  if (result != null) break;
```

```
 }
 return result;
}
public OntologyClass getPlace(String className) {
 OntologyClass result = null;
 for (Iterator i = placeMap.values().iterator(); i.hasNext();) {
  OntologyClass klass = (OntologyClass) i.next();
  if (isEqualsStrings(klass.getName(), className)) result = klass;
  if (result == null) {
   List<OntologyClass> list = this.getSubClassByParent(klass.getName(), false);
   for (Iterator j = list.iterator(); j.hasNext();) {
    OntologyClass child = (OntologyClass) j.next();
    // se o nome da classe for exatamente igual
    if (isEqualsStrings(child.getName(), className)) result = klass;
    if (result != null) break;
   }
  }
  if (result != null) break;
 }
 return result;
}
public boolean isPerson(String className) {
 boolean result = false;
 for (Iterator i = personMap.values().iterator(); i.hasNext();) {
  OntologyClass klass = (OntologyClass) i.next();
  if (isEqualsStrings(klass.getName(), className)) result = true;
  if (result == false) {
   List<OntologyClass> list = this.getSubClassByParent(klass.getName(), false);
   for (Iterator j = list.iterator(); j.hasNext();) {
    OntologyClass child = (OntologyClass) j.next();
    // se o nome da classe for exatamente igual
    if (isEqualsStrings(child.getName(), className)) result = true;
    if (result == true) break;
   }
  }
  if (result == true) break;
 }
 return result;
}
public boolean isObject(String className) {
 boolean result = false;
 for (Iterator i = objectMap.values().iterator(); i.hasNext();) {
  OntologyClass klass = (OntologyClass) i.next();
  if (isEqualsStrings(klass.getName(), className)) result = true;
  if (result == false) {
   List<OntologyClass> list = this.getSubClassByParent(klass.getName(), false);
   for (Iterator j = list.iterator(); j.hasNext();) {
    OntologyClass child = (OntologyClass) j.next();
    // se o nome da classe for exatamente igual
    if (isEqualsStrings(child.getName(), className)) result = true;
    if (result == true) break;
   }
  }
  if (result == true) break;
 }
 return result;
}
private boolean isEqualsStrings(String a, String b) {
 if (a.equalsIgnoreCase(b)) return true;
 String normalized_a = NormalizeString.normalize(a);
 String normalized_b = NormalizeString.normalize(b);
 if (a.equalsIgnoreCase(b)) return true;
 double sm = LexicalSimilarity.similarity(normalized_a, normalized_b);
 if (sm >= 0.8) return true;
 return false;
}
private void loadOntology(String fileName) throws Exception {
 try {
```

```
File file = new File(fileName);
SAXBuilder sb = new SAXBuilder();
Document doc = sb.build(file);
Element rootElement = doc.getRootElement();
Namespace nsowl = rootElement.getNamespace("owl");
Namespace nsrdf = rootElement.getNamespace("rdf");
Namespace nsrdfs = rootElement.getNamespace("rdfs");
List<Element> list = rootElement.getChildren("Class", nsowl);
for (Iterator<Element> i = list.iterator(); i.hasNext();) {
 Element element = i.next();
 String name = element.getAttributeValue("ID", nsrdf);
 if (name == null) {
  name = element.getAttributeValue("about", nsrdf);
  name = formatName(name);
 }
 if (name != null) {
  String parentName = getThingName();
  Element subClassOfElement = element.getChild("subClassOf", nsrdfs);
  if (subClassOfElement != null)
   parentName = subClassOfElement.getAttributeValue("resource", nsrdf);
  if (parentName == null) {
   Element classElement = subClassOfElement.getChild("Class", nsowl);
   parentName = classElement.getAttributeValue("ID", nsrdf);
   if (parentName == null) {
    parentName = classElement.getAttributeValue("about", nsrdf);
    parentName = formatName(parentName);
   }
   if (parentName == null) {
    parentName = getThingName();
   }
  }
  parentName = formatName(parentName);
  OntologyClass classnew = new OntologyClass(name, parentName);
  classMap.put(name, classnew);
 }
}
List<OntologyClass> copy = new ArrayList<OntologyClass>();
copy.addAll(classMap.values());
for (Iterator<OntologyClass> i = copy.iterator(); i.hasNext();) {
 OntologyClass klass = i.next();
 if (!klass.getName().equalsIgnoreCase(getThingName()) ) {
  OntologyClass parentClass = this.getClassByName(klass.getParentName());
  if (parentClass == null) {
   classMap.put(klass.getParentName(),
               new OntologyClass(klass.getParentName(), getThingName()));
  }
 }
}
// percorrer as classes e criar as classes do tipo PersonAttribute
List<Element> listObject = rootElement.getChildren("ObjectProperty", nsowl);
for (Iterator<Element> i = listObject.iterator(); i.hasNext();) {
 Element objectElement = i.next();
 String ID = objectElement.getAttributeValue("about", nsrdf);
 if (ID.equalsIgnoreCase("#isPerson")) {
  String key = "";
  Element domainElement = objectElement.getChild("domain", nsrdfs);
  String resource = domainElement.getAttributeValue("resource", nsrdf);
  if (resource == null) {
   Element classElement = domainElement.getChild("Class", nsowl);
   Element unionElement = classElement.getChild("unionOf", nsowl);
   List<Element> classCollectionElement = unionElement.getChildren("Class", nsowl);
   for (Iterator<Element> j = classCollectionElement.iterator(); j.hasNext();) {
    Element element = j.next();
    resource = element.getAttributeValue("about", nsrdf);
    key = resource.substring(1); // elimina o #
    OntologyClass value = this.getClassByName(key);
    personMap.put(key, value);
   }
```

```
    } else {
     key = resource.substring(1); // elimina o #
     OntologyClass value = this.getClassByName(key);
     personMap.put(key, value);
    }
   } else if (ID.equalsIgnoreCase("#isPlace")) {
    String key = "";
    Element domainElement = objectElement.getChild("domain", nsrdfs);
    String resource = domainElement.getAttributeValue("resource", nsrdf);
    if (resource == null) {
     Element classElement = domainElement.getChild("Class", nsowl);
     Element unionElement = classElement.getChild("unionOf", nsowl);
     List<Element> classCollectionElement = unionElement.getChildren("Class", nsowl);
     for (Iterator<Element> j = classCollectionElement.iterator(); j.hasNext();) {
      Element element = j.next();
      resource = element.getAttributeValue("about", nsrdf);
      key = resource.substring(1); // elimina o #
      OntologyClass value = this.getClassByName(key);
      placeMap.put(key, value);
     }
    } else {
     key = resource.substring(1); // elimina o #
     OntologyClass value = this.getClassByName(key);
     placeMap.put(key, value);
    }
   } else if (ID.equalsIgnoreCase("#isObject")) {
    String key = "";
    Element domainElement = objectElement.getChild("domain", nsrdfs);
    String resource = domainElement.getAttributeValue("resource", nsrdf);
    if (resource == null) {
     Element classElement = domainElement.getChild("Class", nsowl);
     Element unionElement = classElement.getChild("unionOf", nsowl);
     List<Element> classCollectionElement = unionElement.getChildren("Class", nsowl);
     for (Iterator<Element> j = classCollectionElement.iterator(); j.hasNext();) {
      Element element = j.next();
      resource = element.getAttributeValue("about", nsrdf);
      key = resource.substring(1); // elimina o #
      OntologyClass value = this.getClassByName(key);
      objectMap.put(key, value);
     }
    } else {
     key = resource.substring(1); // elimina o #
     OntologyClass value = this.getClassByName(key);
     objectMap.put(key, value);
    }
   }
  }
 }
 catch (Exception e) {
  throw e;
 }
}
private void loadModel(String fileName) throws Exception {
 model = new Model(fileName);
}
private String formatName(String value) {
 if (value.substring(0,1).equals("#")) {
  value = value.substring(1);
 }
 return value;
}
public Map<String, OntologyClass> getClassMap() {
 return classMap;
}
public Map<String, OntologyClass> getPersonMap() {
 return personMap;
}
public Map<String, OntologyClass> getObjectMap() {
```

```
 return objectMap;
}
public Map<String, OntologyClass> getPlaceMap() {
 return placeMap;
}
public Collection<OntologyClass> getList() {
 return classMap.values();
}
public Model getModel() {
 return model;
}
public static OntologyClass getClassByName(String name) {
 if (name.equalsIgnoreCase(getThingName())) return getThingClass();
 if (classMap.containsKey(name)) {
  return classMap.get(name);
 }
 return null;
}
public static OntologyClass getThingClass() {
 return new OntologyClass(getThingName(), null);
}
public static String getThingName() {
 return "Thing";
}
}

package control;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import model.Concept;
import model.OntologyClass;
import pattern.PartyPattern;
import util.NormalizeString;
public class PartyDetect {
 private ArrayList<PartyPattern> patterns = new ArrayList<PartyPattern>();
 public PartyDetect() {}
 public void detect() throws Exception {
  APDetect control = APDetect.getInstance();
  // percorrer todos os pacotes
  for (Iterator<model.Package> i = control.getModel().getPackages().iterator();
                               i.hasNext();){
   model.Package packageElement = i.next();
   String classmodel;
   for (Iterator<Concept> j = packageElement.getConcepts().iterator(); j.hasNext();) {
    OntologyClass classPerson = null;
    Concept conceptElement = j.next();
    classmodel = NormalizeString.normalize(conceptElement.getName());
    classPerson = control.getPerson(classmodel);
    if (classPerson == null) {
     List<OntologyClass> classes = control.getClassBySimilarity(classmodel);
     // percorrer as classes similares e normalizadas
     for (Iterator<OntologyClass> k = classes.iterator(); k.hasNext();) {
      OntologyClass classSimilarElement = k.next();
      classPerson = control.getPerson(classSimilarElement.getName());
      if (classPerson != null) break;
     }
    }
    if (classPerson != null){
     PartyPattern Party = new PartyPattern(conceptElement);
     Party.setModelo(packageElement.getName());
     patterns.add(Party);
    }
   }
  }
 }
 public ArrayList<PartyPattern> getPatterns() {
  return patterns;
```

```
 }
 public void setPatterns(ArrayList<PartyPattern> patterns) {
  this.patterns = patterns;
 }
}

package control;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import model.Concept;
import model.OntologyClass;
import pattern.OrganizationPattern;
import util.NormalizeString;
public class OrganizationDetect {
 private ArrayList<OrganizationPattern> patterns = new ArrayList<OrganizationPattern>();
 public OrganizationDetect() {}
 public void detect() throws Exception {
  APDetect control = APDetect.getInstance();
  // percorrer todos os pacotes
  for (Iterator<model.Package> i = control.getModel().getPackages().iterator();
                               i.hasNext();){
   model.Package packageElement = i.next();
   String classmodel;
   for (Iterator<Concept> j = packageElement.getConcepts().iterator(); j.hasNext();) {
    OntologyClass classPlace = null;
    Concept conceptElement = j.next();
    classmodel = NormalizeString.normalize(conceptElement.getName());
    classPlace = control.getPlace(classmodel);
    if (classPlace == null) {
     List<OntologyClass> classes = control.getClassBySimilarity(classmodel);
     // percorrer as classes similares e normalizadas
     for (Iterator<OntologyClass> k = classes.iterator(); k.hasNext();) {
      OntologyClass classSimilarElement = k.next();
      classPlace = control.getPlace(classSimilarElement.getName());
      if (classPlace != null) break;
     }
    }
    if (classPlace != null){
     OrganizationPattern Organization = new OrganizationPattern(conceptElement);
     Organization.setModelo(packageElement.getName());
     patterns.add(Organization);
    }
   }
  }
 }
 public ArrayList<OrganizationPattern> getPatterns() {
  return patterns;
 }
 public void setPatterns(ArrayList<OrganizationPattern> patterns) {
  this.patterns = patterns;
 }
}

package control;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import model.Association;
import model.Concept;
import model.Model;
import model.OntologyClass;
import pattern.ContractPattern;
import util.NormalizeString;
public class ContractDetect {
 private ArrayList<ContractPattern> patterns = new ArrayList<ContractPattern>();
 public ContractDetect() {}
 public void detect() throws Exception {
```

```
  APDetect control = APDetect.getInstance();
  for (Iterator<model.Package> i = control.getModel().getPackages().iterator();
                                  i.hasNext();;) {
   model.Package packageElement = i.next();
   String classmodel;
   for (Iterator<Concept> j = packageElement.getConcepts().iterator(); j.hasNext();;) {
    OntologyClass classPerson = null;
    Concept conceptElement = j.next();
    classmodel = NormalizeString.normalize(conceptElement.getName());
    classPerson = control.getPerson(classmodel);
    if (classPerson == null) {
     List<OntologyClass> classes = control.getClassBySimilarity(classmodel);
     // percorrer as classes similares e normalizadas
     for (Iterator<OntologyClass> k = classes.iterator(); k.hasNext();;) {
      OntologyClass classSimilarElement = k.next();
      classPerson = control.getPerson(classSimilarElement.getName());
      if (classPerson != null) break;
     }
    }
    if (classPerson != null) {
     for (Iterator<Association> k = conceptElement.getAssociations().iterator();
                                 k.hasNext();;) {
      Association associationElement = k.next();
      if (associationElement.getMultiplicity() == null)
             throw new Exception("As associacoes devem estar definidas suas cadinalidades!"+
                                  conceptElement.getName());
      // somente se for associacao para N
      if (associationElement.getMultiplicity().equalsIgnoreCase("*") ||
          associationElement.getMultiplicity().equalsIgnoreCase("N")) {
       Concept classRelation = associationElement.getConcept();
       for (Iterator<Association> m = classRelation.getAssociations().iterator();
                                   m.hasNext();;) {
        Association associationRelationElement = m.next();
        if (associationRelationElement.getMultiplicity() == null)
           throw new Exception("As associacoes devem estar definidas suas cadinalidades!");
        // verificar se é um objeto
        if (associationRelationElement.getMultiplicity().equalsIgnoreCase("*") ||
            associationRelationElement.getMultiplicity().equalsIgnoreCase("N")) {
         if (control.isObject(associationRelationElement.getConcept().getName())) {
          ContractPattern contract = new ContractPattern();
          contract.setPerson(conceptElement);
          contract.setObject(associationRelationElement.getConcept());
          contract.setRelation(classRelation);
          contract.setModelo(packageElement.getName());
          patterns.add(contract);
         }
        }
       }
      }
     }
    }
   }
  }
 }
}
public ArrayList<ContractPattern> getPatterns() {
 return patterns;
}
public void setPatterns(ArrayList<ContractPattern> patterns) {
 this.patterns = patterns;
}
}
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo