PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

Décius Vinícius Mota Pereira

EXTRAÇÃO DE CLASSES ATRAVÉS DA ANÁLISE FORMAL DE CONCEITOS

Belo Horizonte

2014

Décius Vinícius Mota Pereira

EXTRAÇÃO DE CLASSES ATRAVÉS DA ANÁLISE FORMAL DE CONCEITOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Mark Alan Junho Song
Co-Orientador: Prof Dr. Luis Enrique Zárate

Belo Horizonte
2014

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Pereira, Decius Vinicius Mota
P436e Extração de classes através da análise formal de conceitos / Decius Vinicius Mota Pereira. Belo Horizonte, 2014.
89f.: il.

Orientador: Mark Alan Junho Song
Co-orientador: Luis Enrique Zárate
Dissertação (Mestrado) - Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática.

1. Engenharia de software. 2. Software - Desenvolvimento. 3. Programação orientada a objetos (Computação). 4. Estruturas de dados (Computação). I. Song, Mark Alan Junho. II. Zárate, Luis Enrique. III. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. IV. Título.

CDU: 681.3.03

Décius Vinícius Mota Pereira

EXTRAÇÃO DE CLASSES ATRAVÉS DA ANÁLISE FORMAL DE CONCEITOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Mark Alan Junho Song
Co-Orientador: Prof Dr. Luis Enrique Zárate

Prof. Dr. Mark Alan Junho Song

(Orientador) – PUC Minas

Prof. Dr. Luis Enrique Zárate

(Co-orientador) – PUC Minas

Prof. Dr. Humberto Torres Marques Neto -

PUC Minas

Prof. Dr. Anderson Almeida Ferreira -

UFOP

Belo Horizonte, 02 de abril de 2014

À minha esposa Payla, pelo incentivo na continuidade de meus estudos.

AGRADECIMENTOS

À Deus, pela minha existência, e pela certeza das coisas que se esperam e fatos que não se vêem.

À minha esposa Payla, pela ajuda e apoio constante em mais uma etapa de nossas vidas.

Aos meus pais, Jairo e Nilce, por terem me guiado desde criança na trilha do conhecimento e sabedoria.

Aos Professores Dr. Mark Alan e Dr. Luis Zárate pela orientação neste trabalho e pela paciência e confiança depositadas em mim.

“O começo é a parte mais difícil do trabalho”

Platão

RESUMO

A hierarquia de classes é uma das atividades mais importantes no processo de desenvolvimento de software orientado a objeto. O projeto de classes e sua hierarquização é uma tarefa difícil especialmente quando o que se pretende modelar é extenso e complexo. Alguns problemas são de difícil compreensão, mesmo quando modelados com o uso de uma metodologia. A construção precisa de uma hierarquia de classes requer entendimento profundo do problema, uma correta identificação dos atributos e métodos, suas similaridades, dependências e especializações. Uma hierarquização imprecisa ou incompleta das classes acarreta defeitos de fabricação dos softwares, tornando-os difíceis de manter ou efetuar correções. A Análise Formal de Conceitos fornece uma teoria que possibilita solucionar problemas de hierarquização ao realizar a máxima refatoração das classes enquanto se preservam as relações de especialização. Este trabalho apresenta uma abordagem da aplicação da teoria da Análise Formal de Conceitos na refatoração de classes com o objetivo de simplificar os estágios do projeto de novas classes. Com o uso da abordagem proposta, a redundância de classes é eliminada, a etapa de projeto é automatizada, novas abstrações do domínio são reveladas e o problema é melhor compreendido.

ABSTRACT

The class hierarchy is one of the most important activities of the object-oriented software development. The class design and their hierarchy is a hard task especially when what is sought is extensive and complex modeling. Some problems are difficult to understand even when modeled using a methodology. The precise construction of a hierarchy of classes requires deep understanding of the problem, a correct identification of attributes and methods, their similarities, dependencies and specializations. An inaccurate or incomplete hierarchy of classes entails manufacturing defects of the software, making it difficult to maintain or make corrections. The Formal Concept Analysis provides a theory which enables troubleshoot hierarchy of classes to accomplish the maximum refactoring of classes while preserving the relationships of specialization. This work presents an approach to the application of Formal Concept Analysis theory in class refactoring to simplify the design stages of new classes. Using the proposed approach, the redundancy classes are eliminated, the design stage is automated, new abstractions of the domain are revealed, and the problem is better understood.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AFC – *Análise Formal de Conceitos*

OO – *Orientação a Objeto*

DOO – *Projeto Orientado a Objeto*

AOO – *Análise Orientada a Objeto*

UML – *Unified Modeling Language*

RUP – *Rational Unified Process*

RH – *Recursos Humanos*

XMI – *XML Metadata Interchange*

SUMÁRIO

1 INTRODUÇÃO

A hierarquia de classes é uma etapa importante do desenvolvimento de softwares orientados a objeto. O projeto e manutenção de uma hierarquia são reconhecidos como um problema difícil (Rumbaugh et al., 1991), (Booch, 1994). Esta dificuldade aumenta com a quantidade de classes envolvidas e a possível evolução dos requisitos, os quais podem requerer a incorporação de mudanças no modelo hierárquico.

Alguns problemas são de difícil compreensão, mesmo quando modelados com o uso de uma metodologia. A construção precisa de uma hierarquia de classes requer entendimento profundo do problema, uma correta identificação dos atributos e métodos, suas similaridades, dependências e especializações. Uma hierarquização imprecisa ou incompleta das classes acarreta defeitos de fabricação dos softwares, tornando-os difíceis de manter ou efetuar correções.

A Engenharia de Software surgiu como uma abordagem sistemática e disciplinada para o desenvolvimento de software, estabelecendo um conjunto de atividades a serem seguidas por analistas, projetistas, desenvolvedores e colaboradores. A etapa do projeto de um sistema tornou-se então mais completa e precisa com a aplicação de metodologias de processo, como o RUP (*Rational Unified Process*), o uso de uma linguagem universal, como a UML (*Unified Modelling Language*), e teorias como a Orientação a Objeto (OO).

A aplicação da teoria de Orientação a Objeto nas etapas de construção de sistemas possibilitou o reuso de componentes dos softwares, tornou o desenvolvimento mais ágil e com maior qualidade, o que facilitou sobretudo a manutenção, adaptação e ampliação dos sistemas.

No entanto, mesmo com o uso da teoria de Orientação a Objeto, metodologias de processo e padrões de linguagem para o desenvolvimento de softwares ocorrida nas últimas décadas, é notória a necessidade de se dinamizar ainda mais as etapas de fabricação de software.

Uma das formas de se dinamizar a construção de softwares é valer-se de outras teorias que possam ser aplicadas a uma ou mais etapas do processo de desenvolvimento de software, e que realmente gere algum ganho significativo. Uma teoria abordada neste trabalho, que possui aplicação na etapa de projeto de

sistemas, cujos resultados são relevantes no processo de construção de softwares, é a Teoria da Análise Formal de Conceitos (AFC).

A AFC é um campo da matemática apresentada no início dos anos 80, cujo objetivo principal consiste na classificação dos objetos baseado em seus atributos. Na AFC comumente um domínio do problema é modelado como tabela denominada Contexto Formal, onde as linhas correspondem aos objetos, ou classes, e as colunas aos atributos. A AFC quando adequadamente aplicada na fase de um projeto OO resulta em uma revisão mais profunda do modelo, revelando uma nova estrutura das classes, ao passo que aumenta as características de qualidade desejáveis.

É comum ao projetista de software modelar o diagrama de classes de um sistema baseando-se exclusivamente nas especificações da etapa de Análise e em sua experiência em projetos anteriores. A ausência de um arcabouço matemático que o auxilie na modelagem e hierarquização das classes faz com que falhas sejam cometidas, principalmente no que se refere às características que se esperam de um bom projeto OO de software.

No processo de desenvolvimento de um software a etapa de projeto é crucial para o sucesso ou fracasso do produto final e o esforço gasto nesta etapa é elevado como indicado na Tabela 1.

TABELA 1

Distribuição de Esforço pelas Atividades do Processo de Software

Atividade	Esforço (%)
Fase de Definição	
Requisitos de Negócio	6%
Especificações Funcionais	10%
Fase de Entrega	
Projeto Detalhado	14%
Codificação e Teste de Unidade	40%
Teste do Sistema	20%
Teste de Aceitação do Usuário	10%
Esforço Total	**100%**

Fonte: Adaptado de Safavi et al., 2011

É notório que a fase de um Projeto de um Sistema demanda elevado esforço e tempo dos envolvidos nesta etapa e que possui significativa relevância no resultado do produto final, ou seja, o software.

Um aspecto fundamental no projeto de sistemas é a correta estruturação das classes e suas relações de hierarquia. De acordo com Guedes (2011), a hieraquia de classes torna possível o reuso de componentes, o que facilita o reuso de código e a modularidade. A ausência das relações de hierarquia torna o projeto deficiente, o que implica em um software de baixa qualidade.

1.1 Motivação

A hierarquia de classes e sua refatoração têm sido relatadas por outros autores em diversos cenários de desenvolvimento: na construção da hierarquia através de objetos e especificações das classes (Snelting; Tip, 2000), (Godin; Mili, 1993); na evolução da hierarquia de classes com a finalidade de acomodar novos requisitos através da adição ilimitada de classes (Rapicault; Napoli, 2001) ou através da adição limitada pela compatibilidade com uma hierarquia prévia ou objetos existentes (Godin; Mili, 1993); na reengenharia de uma hierarquia de classes já existente provenientes do relacionamento entre classes e seus atributos e métodos (Wille, 1982), utilizando ferramentas de análise de código (Rapicault; Napoli 2001), pela aplicação de refatoramentos (Godin; Mili, 1993), de modelos UML que incluem associações (Rapicault; Napoli, 2001), de padrões de acesso em aplicações (Grigoriev; Yevtushenko, 2000), pela detecção de defeitos utilizando métricas de software (Valtchev; Grosser; Roume; Hacene, 2003) e na reengenharia procedural de código em ambiente de objetos (Wille, 1982).

Estudos mais recentes realizados por Moha et al (2008), demonstram como identificar objetos em código procedural. Joshi (2009) utiliza informações do programa para realizar a reengenharia de classes. Arévalo (2010) especifica padrões de projeto para que seja realizado o agrupamento de classes e sua correta refatoração. Usman e Anquetil (2012) demonstra como o entendimento de sistemas orientados a objeto podem ser aprimorados com a hierarquização de classes e sua refatoração. Poelmans et al. (2012) fornecem uma visão geral mais facilmente compreensível de pesquisas baseadas em AFC na descoberta do conhecimento e mineração de dados. Kester (2013) aplica a AFC para avaliação de risco em projetos de engenharia de software, baseando-se na igualdade ou diferenciação dos atributos

dos riscos inerentes ao projeto.

Em muitos casos a abordagem proposta se baseia em algoritmos que não possuem resultados teóricos bem definidos. Dessa forma, os métodos correspondentes podem gerar resultados imprevisíveis.

A Análise Formal de Conceitos, em contrapartida, fornece um arcabouço teórico que pode ser aplicado ao projeto e manutenção de hierarquia de classes em ambientes orientados a objeto, cuja compreensão é mais natural.

Várias pesquisas têm adotado a Análise Formal de Conceitos na solução do problema de hierarquização de classes: (Tip; Snelting, 2002), (Wille, 2004), (Grigoriev; Yevtushenko, 2004), (Wolff; Yameogo, 2005), Usman e Anquetil (2012) e Kester (2013). Em muitos casos a abordagem proposta é baseada em técnicas que produzem hierarquias de difícil compreensão para analistas, projetistas e desenvolvedores, dos quais se exige elevado esforço para interpretá-las (Wille, 2004). Além disso, as hierarquias produzidas não decorrem da aplicação de heurísticas de simplificação, nem pelo tratamento da semântica dos atributos das classes.

Dessa forma critérios de qualidade desejáveis de um software, como simplicidade, reusabilidade, extensibilidade e manutenibilidade não são atendidos em sua totalidade, o que poderá acarretar futuros defeitos de fabricação.

1.2 Objetivos

O objetivo principal desta dissertação é a aplicação da Teoria da Análise Formal de Conceitos ao Projeto Orientado a Objeto de Classes através do uso de uma metodologia capaz de re-hierarquizar um modelo de classes definido inicialmente pelo projetista de sistema.

Para se atingir este objetivo as entidades e relacionamentos do modelo de classes inicialmente projetado são mapeados em diagramas de contexto e seus conceitos extraídos através da geração de reticulados e suas transformações. Os conceitos são interpretados como novas entidades do modelo, entidades vazias, superclasses ou subclasses, as quais são transformadas resultando em um novo modelo de classes.

O novo modelo de classes é uma refatoração do modelo inicial, cujas etapas deste processo estão descritas a seguir:

1. Mapeamento do Diagrama de Classes Inicial em uma Tabela de Contexto.
2. Geração do Reticulado de Conceitos.
3. Remoção de Classes Inconsistentes (ou Classes Vazias).
4. Eliminação da Herança Múltipla nos casos em que a linguagem alvo não a suporte.
5. Segmentação das classes onde atributos comuns têm diferente semântica
6. Geração do novo Diagrama de Classes

1.3 Principais contribuições e Resultados Esperados

A seguir são descritas as principais contribuições alcançadas com este trabalho:

- Desenvolvimento de um *framework*, cujo objetivo é o refatoramento de classes de um projeto, baseado na aplicação da AFC e OOD.
- Minimização de redundância de classes em Projetos Orientados a Objeto.
- Automatização e otimização da etapa de projeto de um sistema, mantendo as características inerentes aos modelos de classes orientados a objeto.
- Melhor compreensão do problema, uma vez que resulta em uma re-hierarquização do modelo aproximando-se das características desejáveis de um projeto orientado a objetos.
- Revelação de abstrações do domínio do problema antes não detectadas fornecendo um modelo de classes de fácil entendimento, reuso e manutenção.

1.4 Estrutura da dissertação

Esta dissertação está assim organizada. No Capítulo 2 são fornecidos os conceitos que proporcionam o embasamento teórico da Análise Formal de Conceitos e Orientação a Objeto, especificamente na fase de projeto de sistemas. O Capítulo 3 apresenta uma metodologia a que se propõe neste trabalho. O Capítulo 4 apresenta as características, funcionamento e resultados obtidos com um estudo de caso em análise. Por fim o Capítulo 5 apresenta as considerações finais e indica possíveis linhas de trabalho futuros.

2 REFERENCIAL TEÓRICO

As técnicas da AFC possibilitam que informações hierarquicamente estruturadas sejam manipuladas, organizadas e recuperadas para diversos fins, como na mineração de dados (Snelting e Tip, 2000), recuperação de informação (Godin; Chau, 2000), engenharia de software (Falleri; Huchard, 2008), redes sociais (Arévalo; Falleri, 2006), redes neurais (Joshi; Joshi, 2009), ontologias (Yevtushenko, 2000), entre outras.

Uma das vantagens de se empregar as técnicas de AFC para classificação hierárquica, é o fato de que por meio de um reticulado de conceitos, pode-se representar de forma generalizada informações organizadas hierarquicamente.

No contexto de desenvolvimento de software, a hierarquização dos dados é fundamental na fase de projeto de um sistema. A teoria da Orientação a Objeto (OO) neste caso fornece uma metodologia aplicável à classificação e estruturação de classes e objetos, atributos e/ou métodos, os quais participam da etapa do Projeto Orientado a Objeto (DOO) de softwares em geral.

É possível assim aplicar a AFC em DOO, contudo sendo necessário a priori um entendimento inicial dos conceitos e implicações do uso de ambas as teorias.

2.1 Análise Formal de Conceitos

A Análise Formal de Conceitos é uma técnica da matemática aplicada baseada na matematização da noção de conceito e na estruturação dos conceitos em uma hierarquia conceitual (Wille, 1982). Sua formalização teve início em 1982 com o trabalho apresentado por Wille, que propôs considerar cada elemento de um reticulado como um conceito formal e o reticulado como representando uma hierarquia entre os conceitos.

De acordo com Arévalo e Ducasse (2010) a AFC é uma técnica matemática para descoberta de agrupamentos significativos de objetos que possuem atributos em comum. A AFC pode ser aplicada a entidades de um programa, onde auxilia na geração de visões de alto nível, necessárias ao entendimento, à identificação e à correção de anomalias em softwares. A AFC pode ser utilizada para identificar objetos em código procedural através da pesquisa de funções que acessam variáveis globais (Ducasse; Arévalo, 2003), ou aplicada a sistemas orientados a objeto, através da análise dos componentes do software (classes, atributos ou métodos) e seus relacionamentos (pertence a, usa ou define) (Wille, 2004).

Alguns estudos da AFC objetivam o entendimento de sistemas orientados a objeto: (Wolff; Yameogo, 2005), (Moha et al, 2008), outros utilizam informação do programa para reengenharia de classes (Snelting; Tip, 2002) e (Joshi; Joshi, 2009). A informação extraída do programa é utilizada para construir reticulados de conceitos que são úteis para análise e descoberta de informações importantes.

O problema com as técnicas existentes na AFC é que elas produzem reticulados de conceitos que não são facilmente compreendidos por desenvolvedores, os quais muito se esforçam para interpretá-los. No entanto, quando o reticulado de conceitos é transformado em um modelo de classes, contendo seus atributos e métodos, a interpretação dos conceitos extraídos se torna mais simples.

Outro problema com as técnicas AFC é que se extraem muito mais conceitos do reticulado que o número de objetos tidos como entrada (Huchard; Hervé, 2000), tornando também mais complexa sua análise.

2.1.1 Conceito e Contexto Formal

A noção central da AFC, segundo Priss (2007), é a dualidade denominada Conexão de Galois. Esse tipo de dualidade é observada em situações nas quais coleções de dois tipos diferentes de itens são relacionados, tais que, se a coleção de um tipo é ampliada, a coleção do outro tipo diminui. Um exemplo clássico de Conexão de Galois que pode ser citado é a existente entre documentos e termos. De fato, a quantidade de documentos que podem ser recuperados diminui à medida que se amplia a quantidade de termos que cada documento deve conter simultaneamente.

Segundo ainda Priss (2007), na AFC, os itens de um tipo são denominados objetos formais enquanto os itens do outro tipo são chamados de atributos formais. O adjetivo “formal” é sempre usado para enfatizar que essas noções são formais e que, à rigor, objetos formais e atributos formais não necessitam ser objetos e atributos, respectivamente. O conjunto de objetos formais e atributos formais juntamente com suas relações entre si, forma o que é denominado um contexto formal.

De acordo com Wille (2004), a abordagem proposta em seu artigo tem o intuito de retornar à origem do conceito de reticulado ao século XIX como uma maneira de formalizar a Lógica, cujo passo primordial era a redução de um conceito à sua extensão. A proposta de Wille é fazer com que a redução seja menos abstrata pela retenção, sob alguma medida, da intensão de um conceito. Nesse sentido, Wolff e Yameogo (2005) afirma que, sob a ótica da filosofia, um conceito é uma unidade de pensamento que consiste de duas partes, a extensão e a intensão. A extensão abrange toda a variedade de objetos que pertencem a esse conceito, enquanto a intensão se refere aos atributos válidos de todos esses objetos. Dessa forma, os objetos e atributos desempenham um papel para inúmeras relações, como, por exemplo: na relação de hierarquia entre um subconceito e superconceito que se estabelecem entre os conceitos, na implicação entre atributos ou na relação de incidência “um objeto possui um atributo". O passo fundamental de Wille (2004) foi, portanto, agrupar na definição de um contexto formal, os objetos, seus atributos e na relação incidental entre eles.

Outro passo fundamental de Wille (2004), conforme Wolff e Yameogo (2005), foi a definição de conceito formal. Esse conceito pode ser interpretado, de maneira

informal, como o resultado de um determinado “fecho" das relações implicadas pelas conexões de Galois, ou seja, tendo como ponto de partida um conjunto de objetos formais, é possível a identificação de todos os atributos formais que eles possuem em comum. A partir desses atributos formais, todos os demais objetos formais que também compartilham destes atributos, e que não estavam originalmente no conjunto de objetos formais, podem ser identificados. Assim, é possível que seja compreendido o sentido em que foi usado o termo “fecho”. É fato, que neste ponto, o conjunto de atributos formais e objetos formais não podem ser mais ampliados, visto que todos os objetos que possuem estes atributos já foram identificados. O conceito formal é dessa forma um par de conjunto de objetos e conjunto de atributos formais “fechados".

O modelo matemático que possibilita representar objetos, atributos, e os relacionamentos que indicam se um objeto possui um ou mais atributos é denominado Contexto Formal. Segundo Nilander e Zárate (2011), na AFC comumente um domínio de problema é modelado como uma tabela cruzada, denominada Contexto Formal. Essa representação possibilita a obtenção das relações existentes entre o conjunto de objetos ou instâncias do domínio, a partir da lista de atributos que descrevem suas características, surgindo assim os conceitos formais.

Um Contexto Formal (G, M, I) consiste em dois conjuntos G e M, e uma relação binária I entre esses conjuntos. Os elementos de G são denominados objetos, enquanto que os de M são denominados atributos. Caso um objeto g possua uma relação I com um atributo m, essa relação é expressa por gIm ou (g, m) $\in$ I. Isso é interpretado como o “objeto g possui o atributo m”.

Para um conjunto A $\subseteq$ G de objetos (denominado extensão) é definido:

$$A' := \{m \in M \mid gIm \forall g \in A\}$$

como o conjunto de objetos comuns aos atributos em A. Em correspondência, o conjunto B $\subseteq$ M (denominado intensão) de atributos é definido

$$B' := \{g \in G \mid (gIm \forall g \in B\}$$

como o conjunto de objetos comuns aos atributos em B. Assim, um Conceito Formal de um contexto (G,M,I) consiste em um par ordenado (A,B) onde a seguinte propriedade se aplica:

$A \subseteq G$, $B \subseteq M$, $A' = B$ e $B' = A$

De forma simplificada, o conjunto dos objetos do conceito formal é denominado extensão, e dos atributos, intensão. Cada elemento da extensão possui todos da intensão e vice-versa.

Um Conceito Formal pode ser definido como um par (O, A) em um Contexto (G, M, I) onde $O \subseteq G$ se denomina extensão, $A \subseteq M$ se denomina intensão e há uma conexão de Galois entre G e M definida por:

- Para todo objeto o em G, não existente em O, há um atributo a em A, que o objeto o não possui.

- Para todo atributo a em M, não existente em A, há um objeto o em O que não possui o atributo a.

A representação de um contexto formal pode ser realizada através de uma tabela, cujas relações entre os objetos e os atributos são binárias, conforme pode ser observado no exemplo da Tabela 2. A Tabela 2, também denominada Tabela de Contexto representa uma estrutura que define objetos (linhas), atributos (colunas) e suas respectivas relações de incidência. Nesta representação, um conceito formal é um subvetor maximal tal que todas as células do subvetor estão assinaladas (Wolff; Yameogo, 2005). Nesta tabela a coleção de objetos, tentilhão e águia formam um conceito formal, juntamente com os atributos pássaro e voa, ou seja, existe um conceito formal “pássaro que voa", neste contexto, que possui as extensões tentilhão e águia.

TABELA 2

Um exemplo simplificado de contexto formal

Animais	Predador	Voa	Pássaro	Mamífero
Leão	X			X
Tentilhão		X	X	
Águia	X	X	X	
Lebre				X
Avestruz			X	

Fonte: Wolff, 1994

Assim como a teoria de conjuntos ou a álgebra relacional são fundamentais para o entendimento e modelagem dos bancos de dados relacionais, o conceito formal, conforme observado por Priss (2007), e definido na AFC, descreve uma propriedade natural da representação de informação, cuja aplicação é de grande valia para as hierarquias e estruturas de objeto e atributo. Segundo Priss (2007), as estruturas básicas da AFC foram descobertas por diferentes pesquisadores, em sua maioria de forma individual e em épocas e contextos diversos. No entanto, segundo esse autor, Wille (2004) é quem desenvolveu a AFC em uma teoria complexa, ampliada e com uma vasta gama de aplicações, enquanto que os demais autores até estabeleceram soluções estruturalmente elegantes, contendo contextos e reticulados, contudo não souberam explorar ou ampliar suas soluções.

2.1.2 Reticulado de Conceitos

Quando o conjunto de todos os conceitos formais de um Contexto Formal é hierarquicamente ordenado, recebe a denominação de reticulado de conceitos. Seus conceitos formais se relacionam como (E1, I1) ≤ (E2, I2), quando $E1 \subseteq E2$ e $I2 \subseteq I1$, sendo (E1, I1) chamado subconceito e (E2, I2) chamado superconceito (Wille, 2004).

O Reticulado de Conceitos é um grafo não dirigido, cujos nós representam os objetos ou entidades modeladas, ou apenas uma associação de conceitos. Atrelados aos nós encontram-se as propriedades ou atributos do modelo e/ou métodos.

Tão adequado quanto representar um contexto formal por meio de uma tabela cruzada, é representar um reticulado por meio de um diagrama. O diagrama representa o reticulado como um grafo, cujos vértices denotam conceitos formais e cujas arestas denotam suas relações (E1, I1) ≤ (E2, I2). Quando dois conceitos formais se relacionam como subconceito e superconceito, sem que haja qualquer outro conceito formal entre ambos, seus vértices devem ser conectados por uma aresta no diagrama. O vértice mais alto do diagrama representa o conceito formal cuja extensão contém todos os objetos, enquanto o mais baixo contém todos os atributos em sua intensão (Wille, 2004).

Os reticulados conceituais podem ser melhor visualizados com o uso dos Diagramas de Hasse, os quais possibilitam exibir as propriedades de cobertura de

objetos e atributos através de um diagrama hirerárquico. Do ponto de vista teórico seriam necessários dois Diagramas de Hasse para representar os objetos e atributos. Contudo, é possível combiná-los em um único diagrama, conforme mostrado em (Godin; Chau, 2000).

Para clarear e ilustrar as idéias anteriores, e exibir a visibilidade gráfica das conexões de Galois e dos conjuntos de conceitos formais, um Reticulado de Conceitos representado em Diagrama Hasse (veja Figura 1) é gerado a partir da Tabela 2.

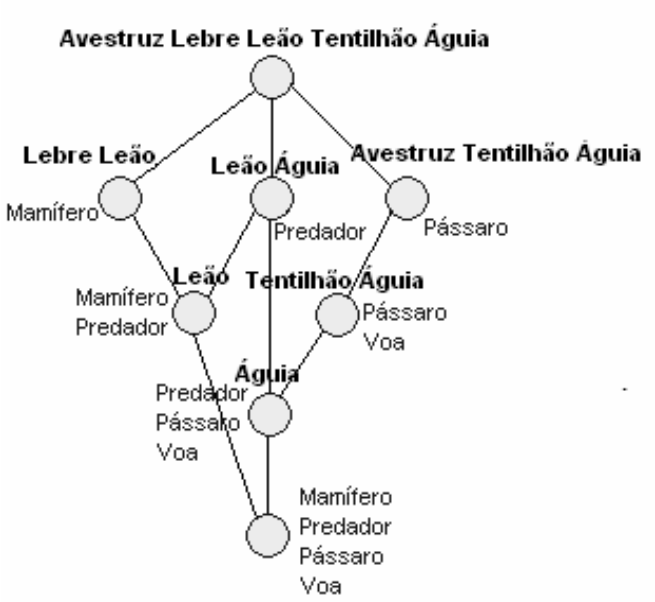


Figura 1: Exemplo de representação por reticulado de um contexto formal

Fonte: Adaptado de Wolff, 1994

Para que se compreenda um reticulado semelhante ao da Figura 1, o qual expressa um contexto formal, a leitura pode ser realizada da seguinte forma: As anotações acima de cada nó representam conjuntos de objetos, enquanto que as abaixo representam os atributos. Um objeto *a*, representado por um círculo (ou nó) no diagrama, possui um atributo *x* se e somente se houver uma trilha (ou aresta) de linhas ascendentes que conecte o objeto ao atributo. Dessa forma, as arestas ascendentes representam a intensão de um conceito, e as descendentes a sua extensão. A extensão do conceito mais alto, representado pelo nó situado no topo do grafo, é por sua vez o conjunto de todos os objetos. De forma geral, ela pode ou não possuir intensão, caso haja atributo. No exemplo da Figura 1, caso o atributo “animal" fosse acrescido, o conceito no topo teria este atributo, que também seria a

sua intensão. A intensão do conceito do nó mais inferior, localizado na base do grafo, é o conjunto de todos os atributos. A extensão dele será nula caso alguns desses atributos sejam contraditórios.

A ausência de anotações inferiores e superiores, por sua vez, pode significar que a classe poderá ser inferida através de hereditariedade, ou que não existe, ou seja, é uma classe vazia.

De forma genérica, se em um reticulado qualquer, A é um conceito acima de um conceito B, e os dois possuem uma ligação, o conceito A pode ser considerado um conceito mais geral do que B e, como tal, carrega parte dos atributos de B. Como consequência, é correto afirmar que se B acontece, A também está presente, sugerindo uma vinculação lógica. No reticulado, não se enxerga apenas uma hierarquia de conceitos, mas também todo o conjunto binário de relações presentes entre os conceitos. Isso faz com que a análise visual dos dados possa ser obtida olhando para uma hierarquia de classes.

Na Figura 1, cada nó no grafo é um conceito. Se dois objetos foram colocados no mesmo nó (conceito), eles possuem os mesmos atributos e são, portanto, instâncias da mesma classe de objetos que possuem aquele conjunto de atributos.

O número de atributos em comum pode ser ponderado em função do número de atributos que está presente em apenas um dos objetos, para medir a similaridade entre dois objetos. Contudo, muitas vezes atributos podem aparecer em pares ou trios, com a consequência de serem posicionados no mesmo nó do reticulado. Isso significa, que do ponto de vista estrutural, os atributos não estão adicionando informações relevantes para diferenciar objetos (Snelting; Tip, 2002).

O Reticulado de Conceitos pode ser construído a partir de algoritmos de associação ou classificação, como o GRAND (vide seção 2.1.3). Estas estruturas podem genericamente ser formadas através do algoritmo descrito abaixo. Este algoritmo constrói o reticulado de conceitos em níveis, onde se produz primeiro o conceito formal, seus subconceitos e as arestas necessárias. Depois, para cada conceito formal, o processo é repetido, produzindo-se mais subconceitos e mais arestas.

Algoritmo Pseudo-código para construção de reticulado de conceitos.

entradas: contexto formal (O', A, A', R).

resultados: reticulado de conceitos.

converter contexto para mono-valorado
inserir $(O, O^{\uparrow})$ no reticulado
conceitos $\leftarrow \{(O, O^{\uparrow})\}$
enquanto *conceitos* $\neq \varnothing$ **execute**
  seguintes $\leftarrow \varnothing$
  para todo $(E, I) \in$ *conceitos* **execute**
    subconceitos $\leftarrow$ subconceitos de (E, I)
    para todo $(E_1, I_1) \in$ *subconceitos* **execute**
      inserir (E_1, I_1) no reticulado
      conectar (E, I) e (E_1, I_1)
      seguintes $\leftarrow$ *seguintes* $\cup \{(E_1, I_1)\}$
    fim para
  fim para
  conceitos $\leftarrow$ *seguintes*
fim enquanto

A AFC fornece assim uma ferramenta formal para reconhecimento de grupos de elementos que compartilham de propriedades e métodos comuns, os quais revelam dependências implícitas e explícitas, possibilitando uma melhor compreensão dos conceitos.

2.1.3 Algoritmo GRAND

O Reticulado de Conceitos pode ser gerado através de diversos algoritmos, como: AClose, Titanic, Frequent Next Neighbours, Galicia, GRAND e Rulearner (Davey; Priestley, 2001), (Ganter, 1984) e (Ganter; Wille, 1999). Algoritmos como AClose, Titanic, Frequent Next Neighbours e Galicia se baseiam em regras probabilísticas de associação, enquanto o GRAND (Oosthuizen, 1988) e Rulearner (Oosthuizen; McGregor, 1988) se baseiam em regras de classificação. Estes dois últimos têm por finalidade gerar um pseudo-reticulado onde as regras encontradas possuem três tipos de elementos: nós-atributos, nós-objetos e nós-intermediários, o que os torna bastante úteis na classificação de entidades ou classes na etapa de projeto de softwares orientados a objeto.

De maneira conceitual, para um contexto formal (G,M,I), os nós-atributos

representam cada um dos atributos de M, nós-objetos representam os objetos do contexto formal e nós-intermediários servem para manter a estrutura do reticulado, ou seja, para garantir que os supremos de quaisquer nós pertençam ao reticulado. O pseudo-reticulado é construído incrementalmente.

Inicialmente, cada atributo é representado por um nó-atributo no pseudo-reticulado. Em seguida, inclui-se um novo objeto representado por um nó-objeto. Se já existe no reticulado um nó-objeto com os mesmos atributos que o novo objeto, então a atualização do pseudo-reticulado é feita, incluindo-se o novo objeto à extensão do nó-objeto correspondente. Caso contrário, cria-se um nó-objeto para o novo objeto. Depois, associa-se o novo nó-objeto a cada um dos nós-atributos representando a intensão do objeto. Em seguida, verifica-se a estrutura do reticulado; se o supremo (interseção) do novo nó e cada um dos nós presentes no pseudo-reticulado está presente no pseudoreticulado.

Então, atualizam-se as listas de sucessores dos conceitos envolvidos na atualização do pseudo-reticulado e remove-se as associações redundantes.

O algoritmo para a construção do pseudo-reticulado é apresentado em pseudo-código mais adiante. Inicialmente, criam-se os nós-atributos (linha 1). Depois, o algoritmo entra em um ciclo para incluir todos os objetos no reticulado (linhas 2 a 29). Se existe um nó-objeto pertencente ao pseudo-reticulado com a mesma intensão do objeto a ser incluído, então a atualização do reticulado consiste em incluir o novo objeto à extensão do nó-objeto correspondente (linhas 4 e 5). Caso contrário, cria-se um nó-objeto para o novo objeto (linha 7). O novo nó-objeto é associado a todos os nós-atributos que estão na intensão do objeto (linhas 8 a 10). Depois, verifica-se se a estrutura do pseudo-reticulado foi preservada após a inclusão do nó-objeto (linhas 11 a 27), ou seja, se os supremos do nó-objeto e os demais nós do pseudo-reticulado também pertencem ao pseudo-reticulado. Para verificar a estrutura do reticulado, todos os nós já pertencentes ao reticulado, exceto os nós atributos, são avaliados (linha 3). Os nós são avaliados em ordem de tamanho da interseção entre a intensão do nó e a do novo objeto. A cada iteração, escolhe-se o nó (X,Y) que possui o maior número de atributos em comum com o novo objeto (linha 12).

Concluídos os passos descritos anteriormente, cria-se um novo nó, cuja intensão é a interseção da intensão do objeto e a do nó que está sendo avaliado, e

cuja extensão é a extensão do nó avaliado acrescida do novo objeto (linha 13). Este novo nó é incluído no pseudo-reticulado caso ele já não pertença ao mesmo (linha 14). Depois, o novo nó é associado aos demais nós do pseudo-reticulado (linhas 15 a 23). Para evitar que associações redundantes sejam mantidas, ou seja, para garantir que apenas a relação de cobertura seja mantida pelas listas de sucessores, utilizaram-se os conjuntos conjA e conjB.

O conjA contém os antecessores do novo nó cujas listas de sucessores devem ser atualizadas removendo-se os sucessores do novo nó (linha 25). O conjB contém os sucessores do novo nó e serve para atualizar a lista de sucessores desse novo nó. Os sucessores de nós que estão em conjB são removidos da lista de sucessores do novo nó (linha 26). Após avaliar o nó (X,Y), ele é retirado da lista de nós para que os supremos sejam avaliados (linha 24). O processo é repetido até que não existam mais nós a serem avaliados.

Algoritmo construirPseudoReticulado
Entrada: Um contexto formal (G,M,I)
Saída: O pseudo-reticulado L associado ao contexto formal (G,M,I)
início

1. $L := \{(m', \{m\}) | m \in M\}$
2. **para cada** $g \in G$ **faça**
3. $C := \{(X,Y) \in L - ||Y| > 1\}$ /* *C contém os nós-objetos e nós-intermediários* */
4. **se** $(W,g') \in L$ **então**
5. $W := W \cup \{g\}$
6. **senão**
7. $L := L \cup \{(\{g\}, g')\}$
8. **para cada** $m \in g'$ **faça**
9. $(m', \{m\}).sucessores := (m', \{m\}).sucessores \cup \{(\{g\}, g')\}$
10. **fim para**
11. **enquanto** $C \neq \emptyset$ **faça**
12. $(X,Y) := (X_i, Y_i) \in C$ tal que $|Y_i \cap g'| = max\{|Z \cap g'| | (W,Z) \in C\}$
13. $(X_2, Y_2) := (\{g\} \cup X, g' \cap Y)$
14. $L := L \cup \{(X_2, Y_2)\}$
15. **para cada** $(X_1, Y_1) \in L$ **faça**
16. **se** $(X_1, Y_1) > (X_2, Y_2)$ **então**
17. $(X_1, Y_1).sucessores := (X_1, Y_1).sucessores \cup \{(X_2, Y_2)\}$
18. $conjA \leftarrow conjA \cup (X_1, Y_1)$
19. **senão se** $(X_1, Y_1) < (X_2, Y_2)$ **então**
20. $(X_2, Y_2).sucessores := (X_2, Y_2).sucessores \cup \{(X_1, Y_1)\}$
21. $conjB := conjB \cup (X_1, Y_1)$
22. **fim se**
23. **fim para**
24. $C := C - \{(X,Y)\}$
25. $\forall (W,Z) \in conjA \quad (W,Z).sucessores := (W,Z).sucessores - (X_2, Y_2).sucessores$
26. $\forall (W,Z) \in conjB \quad (X_2, Y_2).sucessores := (X_2, Y_2).sucessores - (W,Z).sucessores$
27. **fim enquanto**
28. **fim se**
29. **fim para**

fim

Após a construção do pseudo-reticulado, as regras de classificação podem ser extraídas. As regras são geradas para cada um dos atributos-classe. Ao gerar as regras para um atributo-classe, o algoritmo faz uma busca pelo pseudo-reticulado tentando identificar o nó mais geral (com menos atributos) a partir do qual pode-se deduzir a classe. A busca inicia-se pelo nó-atributo corresponde à classe que se está extraindo as regras. Visitam-se os sucessores desse nó até encontrar um nó (X,Y) que atenda às seguintes propriedades:

1 o atributo-classe *c* pertence à intensão *Y* do nó;

2 c ∈ (Y – {c})";

3 *Y* é mínimo em relação às regras já geradas.

Quando as propriedades forem atendidas, gera-se a regra Y- {c} → c. Os sucessores de (X,Y) não são avaliados pois eles não serão mínimos.

O algoritmo para encontrar as regras, *GerarRegras*, apresentado mais adiante executa uma busca em largura nos sucessores do nó-atributo da classe cujas regras serão extraídas (linha 1). Já a função *gerarRegrasNos,* executa efetivamente a geração de regras a partir dos nós do pseudoreticulado. O algoritmo verifica, para cada nó (X,Y) que ainda deve ser avaliado (*nosVisitar*), seus sucessores na tentativa de extrair as regras (linhas 1 a 9). Se um nó (X1,Y1) sucessor de (X,Y) atende às propriedades descritas acima, então uma nova regra é criada (linhas 3 e 4). Caso contrário, os sucessores do nó (X1,Y1) deverão ser avaliados na próxima iteração e, com isso, o nó é inserido no conjunto dos próximos nós a serem avaliados (linha 6). Após avaliar todos os nós de um nível, a função *gerarRegrasNos* é chamada recursivamente para os nós do próximo nível (linhas 10 a 12). O algoritmo termina quando não existem mais nós a serem avaliados, ou seja, quando *proximosNos* está vazio.

Algoritmo GerarRegras
Entrada: Um pseudo-reticulado L e um conjunto de atributos-classe C
Saída: Um conjunto R de regras de classificação válidas
início
1. **para cada** $c \in C$ **faça**
2. $(X,Y) := (W,\{c\}) \in L$ /* (X,Y) *é o nó-atributo da classe* c */
3. $R := R \cup gerarRegrasNos((X,Y).sucessores, L, c)$
4. **fim para**
fim

Algoritmo gerarRegrasNos
Entrada: Uma lista de nós a visitar $nosVisitar$, um pseudo-reticulado L e um atributo-classe c
Saída: Um conjunto R de regras de classificação válidas
início

```
1.  para cada (X,Y) ∈ nosVisitar faça
2.    para cada (X1,Y1) ∈ (X,Y).sucessores faça
3.      se c ∈ (Y1 − {c})'' então
4.        R := R ∪ {Y1 − {c} → c}
5.      senão
6.        proximosNos := proximosNos ∪ {(X1,Y1)}
7.      fim se
8.    fim para
9.  fim para
10. se proximosNos ≠ ∅ então
11.   R := R ∪ gerarRegrasNos(proximosNos,L,c)
12. fim se
13. retorne R
```

fim

2.2 Trabalhos Relacionados - Aplicação da AFC em DOO

Várias pesquisas têm adotado a AFC na solução de problemas ou otimização do desenvolvimento de software, seja a nível de projeto, codificação, manutenção ou engenharia reversa.

No que se refere à aplicação da AFC na identificação de módulos e objetos, Arévalo e Ducasse (2003) apresentam um método que extrai objetos do código procedural. Outra abordagem compara a capacidade de identificação de objetos dos módulos e a técnica AFC (Moha et al, 2008).

Algumas heurísticas são descritas para interpretar os conceitos do reticulado. Uma abordagem que transforma um sistema legado em COBOL em um sistema de componentes distribuídos é proposta em (Usman; Anquetil, 2012).

O relacionamento entre as variáveis de programa e procedimentos através da AFC, para reestruturação de programas desenvolvidos em linguagem C, é apresentada em (Huchard; Hervé, 2000).

No que se refere à reengenharia de sistemas orientados a objeto e a AFC, Godin e Mili (1993) propuseram analisar classes através de seus métodos, para avaliar e refatorar hierarquia de classes OO utilizando a AFC. Rapicault e Napoli (2001) definem um algoritmo incremental, e estuda a especialização do método em um framework AFC. Glinz (2007) compara várias configurações da AFC. Grigoriev e

Yevtushenko (2004) aplicam a AFC para extrair hierarquia de interfaces Java a partir das classes Java.

Snelting e Tip (2000) apresentam um framework para detectar e reparar problemas de projeto em uma hierarquia de classes. O framework é também utilizado por Ganter (2003) para refatorar hierarquia de classes em Java. Arévalo e Ducasse (2003) utilizam a AFC para compreender como métodos e classes em uma hierarquia OO são combinadas, através dos significados da herança e dos relacionamentos das interfaces.

Falleri e Huchard (2008) utilizam a AFC para identificar características de herança nas hierarquias. Moha et al (2008) por sua vez utilizam a AFC para visualizar a estrutura de classes Java, e selecionar um modo efetivo para leitura de métodos. Em (Joshi; Joshi, 2009) uma técnica apoiada por ferramenta é apresentada, com o intuito de identificar abstrações úteis e hierarquia de classes em códigos OO.

A AFC é utilizada para detectar a presença de falhas e padrões de projeto em (Wille, 2004), e refatoramentos para corrigir estas falhas. É também utilizada para compreensão do código fonte de softwares através da busca de conceitos relevantes (Wollf; Yameogo, 2005).

Segundo Huchard e Hervé (2000), a AFC é útil para examinar informação gerada pela execução de um programa, com o objetivo de reconstruir os grafos de fluxo. É útil ainda na mineração de dados (Usman; Anquetil, 2012), na detecção de atributos de classes a partir do código fonte de sistemas, e na procura por padrões de codificação e suas violações (Godin; Chau, 2000).

No que se refere à filtragem obtida pela AFC, alguns estudos como (Glinz, 2007), (Falleri; Huchard, 2008) e (Moha et al., 2008) a aplicam na engenharia reversa, em que técnicas de filtragem de conceitos solucionam o problema da complexidade em reticulado de conceitos. A noção de particionamento dos conceitos é utilizada para filtrar os conceitos que não são interessantes para a criação dos módulos. Ganter (2003) filtra os conceitos conforme suas propriedades em reticulados construídos em sua abordagem. Rapicault e Napoli (2001) descrevem algumas características para redução da pesquisa, por conceitos que descrevem importantes esquemas em hierarquia de classes. Grigoriev e Yevtushenko (2004) fornecem alguns padrões que podem emergir quando o contexto é baseado em métodos e atributos das classes.

Em (Moha et al., 2008) um catálogo de padrões para reticulados de conceitos foi gerado, com o propósito de permitir a automatização da tarefa de interpretação do reticulado, auxiliando o projetista a concentrar na tarefa de reengenharia mais que o entendimento da complexidade inerente ao reticulado. Não é objetivo de Moha et al. (2008) a hierarquização de classes a partir da geração do reticulado de conceitos.

A abstração de conceitos e relacionamentos para um domínio específico foram automatizadas por técnicas baseadas na aplicação da AFC, em contexto dirigido a modelo como proposto por Arévalo e Ducasse (2003). Diagramas de classes em UML foram utilizadas como linguagem de modelagem para o modelo de classes, com o objetivo de mostrar como a abordagem dirigida ao modelo possibilita parametrização e generalização para qualquer meta-modelo expressar modelos de classe. Contudo o trabalho apresentado por Arévalo e Ducasse (2003) não trata a semântica dos atributos, ou simplifica o reticulado de conceitos através de sua poda.

Em (Arévalo; Ducasse, 2010), algoritmos foram desenvolvidos para a construção de hierarquias de classes através de diferentes frameworks, mostrando as vantagens e desvantagens de uso do Reticulado de Galois e sub-hierarquia como modelos de hierarquias de classes. Um incoveniente de Arévalo e Ducasse (2010) consiste na geração de herança múltipla, requerendo ajustes para linguagens que possuem somente herança simples.

Em (Huchard; Hacene 2007) foi apresentada uma abordagem genérica implementada em uma ferramenta capaz de lidar com qualquer linguagem descrita por um meta-modelo, a qual auxilia arquitetos de software a projetar e melhorar seus modelos de classe. Moha et al. (2008) também apresentou a técnica Análise Relacional de Conceitos (RCA), como uma extensão da AFC. Embora Huchard e Hacene (2007) tenham contribuído com uma teoria capaz de normalizar modelos de classe baseado em diferentes meta-modelos, a semântica dos atributos não é tratada tal como Moha et al. (2008).

Este trabalho propõe aperfeiçoar as pesquisas relatadas anteriormente no que se refere à simplificação do reticulado de conceitos, através da aplicação da AFC, tal como (Arévalo, 2003), (Wille, 2004), (Falleri; Huchard, 2011) e (Moha et al., 2008), mas também com o uso de heurísticas de poda do Reticulado de Conceitos, e tratamento da semântica dos atributos das classes. As diferenças existentes entre as linguagens de programação, no que se refere ao conceito de herança simples ou

múltipla, também é suportado por este trabalho, o que não ocorre nas pesquisas relatadas em (Huchard; Hacene, 2007), (Arévalo et al., 2010), (Yevtushenko, 2000) e (Rapicault; Napoli, 2001).

3 ABORDAGEM PROPOSTA

Quando se utiliza Análise Formal de Conceitos para o projeto de hierarquia de classes, o conjunto dos objetos formais *G* é um conjunto de artefatos de software, ou seja, classes, objetos ou variáveis do programa, os quais são utilizados como ponto de partida na busca por uma adequada hierarquia de classes.

O conjunto dos atributos formais *M* corresponde às propriedades de classes ou objetos. As propriedades que são relevantes incluem os atributos (variáveis de instância) e métodos (corpo e/ou assinatura do método). Neste trabalho considera-se como ponto de partida um conjunto de especificações de classes, isto é, *G* é um conjunto de classes. Considera-se ainda somente a refatoração de atributos das classes, cuja aplicação é estendida aos métodos.

Um aspecto importante neste trabalho é a minimização de redundância, e a criação de subclasses via especializações. Em relação à minimização de redundância, a idéia consiste na refatoração de classes reduzindo inconsistências e minimizando futuramente a redundância de código.

Quanto às subclasses utiliza-se também esta refatoração das classes, como uma forma de identificar uma hierarquia, estabelecendo uma identificação de tipo e subtipo.

Quando se utiliza Análise Formal de Conceitos para o projeto de hierarquia de classes, o conjunto dos objetos formais G é um conjunto de artefatos de software, ou seja, classes, objetos ou variáveis do programa, os quais são utilizados como ponto de partida na busca por uma adequada hierarquia de classes.

O conjunto dos atributos formais M corresponde às propriedades de classes ou objetos. As propriedades que são relevantes incluem os atributos (variáveis de instância) e métodos (corpo e/ou assinatura do método).

O Contexto Formal é obtido através do mapeamento das entidades ou objetos e os atributos e métodos em uma Tabela de Contexto, como a ilustrada à esquerda da Figura 2.

O Reticulado de Conceitos é por sua vez gerado a partir do Contexto Formal, e possui todas as classes ou objetos, atributos e métodos, do modelo de classes do DOO em análise. Além das entidades e propriedades existentes na tabela de

contexto, novas entidades podem surgir no reticulado, visto que uma nova hierarquização do modelo inicialmente proposto é revelada através da aplicação da AFC. A Figura 2 à direita ilustra o Reticulado de Conceitos gerado a partir da Tabela de Contexto da mesma figura.

	Atributos					
Objetos	s1	s2	s3	s4	s5	s6
d1	x			x		
d2	x		x	x		
d3	x		x			
d4	x					x
d5		x	x	x		
d6		x	x			
d7		x			x	
d8				x	x	x
d9		x			x	

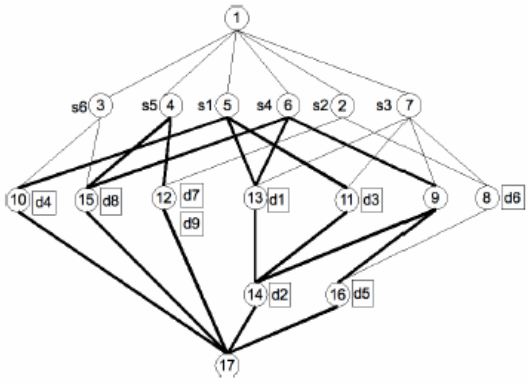


Figura 2: Exemplo de Contexto Formal à esquerda, e Reticulado de Conceitos à direita, ambos gerados a partir de um Projeto Orientado a Objetos

Fonte: Carreiro, 2010

É possível perceber através da Figura 2 (à direita) que os nós do reticulado gerado se referem aos objetos ou classes iniciais do modelo (d1 a d9), acrescido de novas classes em potencial (1, 2, 3, 4, 5, 6 e 7) e classes inconsistentes, ou vazias (9). Assim como os nós, as arestas também são classificadas no modelo e podem ser interpretadas como generalizações, cujo relacionamento entre as classes ou objetos se faz através de herança múltipla ou herança simples.

A aplicação da AFC em DOO assim revela ao projetista de software uma re-hierarquização do modelo de classes inicial, coerente com o que se está modelando, e passível de ser utilizada como modelo de classes final do projeto.

3.1 Arquitetura

Para obtenção da máxima refatoração das classes, e uma nova hierarquização do modelo, é necessária a execução dos seguintes passos:

a. Mapeamento do Diagrama de Classes em uma Tabela de Contexto;

b. Geração do Reticulado de Conceitos;

c. Eliminação da Herança Múltipla nos casos em que a linguagem alvo não a suporte.

d. Remoção de Classes Inconsistentes (ou Classes Vazias);

e. Segmentação das classes onde atributos comuns têm semântica diferente;

A fim de automatizar o processo de refatoração do modelo inicial do projeto de classes, um framework foi desenvolvido para experimentação dos aspectos teóricos explanados neste trabalho. Inicialmente duas premissas básicas foram estabelecidas para o framework:

1. O modelo inicial da classe é descrito no padrão UML e,
2. A integração com uma ferramenta AFC que interprete o formato XMI (XML Metadata Interchange), como Galicia (Yevtushenko, 2000) ou Conexp (Valtchev et al, 2003).

A arquitetura proposta segue dessa forma o fluxo de entrada, processamento e saída de dados mostrado na Figura 3, que foi denominado como Framework AFC.

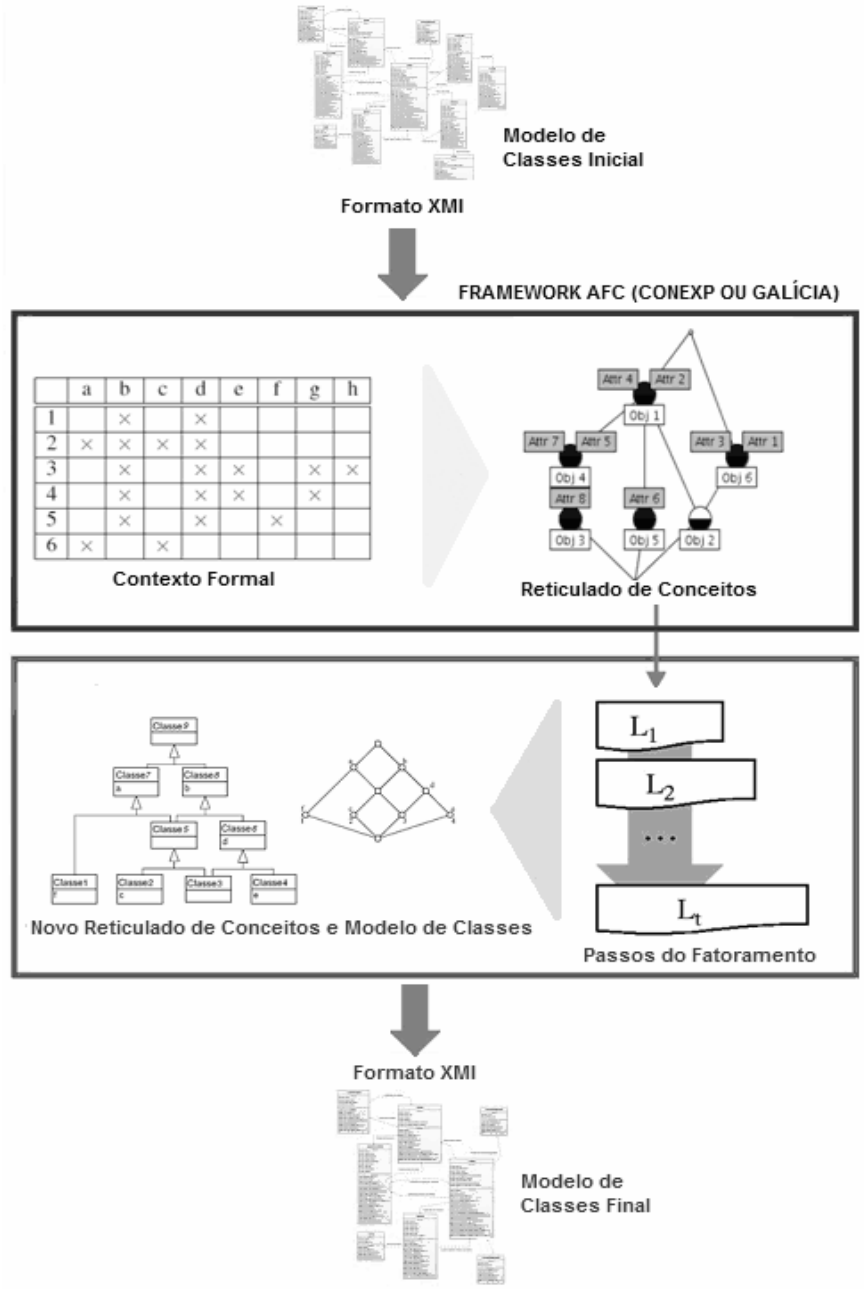


Figura 3: Fluxo de Processamento do Framework AFC

3.2 Refatoração de Classes

Para que as classes sejam refatoradas o framework mostrado na Figura 3 opera da seguinte forma: a leitura de um arquivo XMI obtido através da exportação do diagrama de classes em padrão UML é realizado pelo framework, que através de

uma ferramenta AFC, como o Conexp ou Galicia é transformado em um Diagrama de Contexto, e por sua vez em um Reticulado de Conceitos. Outra sequência de passos é então executada através de um algoritmo, que tem por função eliminar os nós vazios e tratar as classes redundantes do reticulado, por exemplo, quando não se deseja ter herança múltipla no modelo. Quando a herança múltipla é permitida no modelo, o framework é parametrizado para que ignore esta etapa.

Como última etapa do framework, uma relação de atributos comuns das classes do projeto são apresentadas ao projetista, para que sejam analisadas suas propriedades, de forma que não havendo correspondência semântica entre elas, haja uma segmentação automática das classes.

3.2.1 Mapeamento do Diagrama de Classes em uma Tabela de Contexto

Primeiramente considera-se que o projetista de software opte em escolher as entidades e objetos do modelo de classes em sua hierarquia original, ou no mesmo nível hierárquico, sem seus relacionamentos de especialização ou associação. A forma como o modelo inicial de classes é estabelecida pelo projetista difere na presença ou ausência das relações de hierarquia entre as classes, bem como da unicidade ou replicação dos atributos e métodos existentes nas classes.

Há duas formas do projetista de software modelar as entidades e objetos: através do desenho das classes concretas, sem qualquer hierarquia prévia existente, ou por meio de uma hierarquia pré-existente, onde as propriedades e métodos são replicados das classes ascendentes para as classes descendentes. A arquitetura proposta neste trabalho pode ser aplicada igualmente a qualquer um dos modelos, sem prejuízo dos resultados obtidos, conforme exemplificado a seguir:

a. Modelagem inicial através de classes concretas

Considere que o projetista de software tenha estabelecido as entidades/objetos do modelo de classes sem suas propriedades e inter-relacionamentos. Considere ainda que o projetista tenha especificado o conjunto de quatro classes concretas e respectivos atributos como ilustrado na Figura 3. A especificação poderia ser interpretada como o conjunto exato de classes concretas

que o modelo deveria conter, em outras palavras, estas classes são as únicas a produzir objetos em uma aplicação.

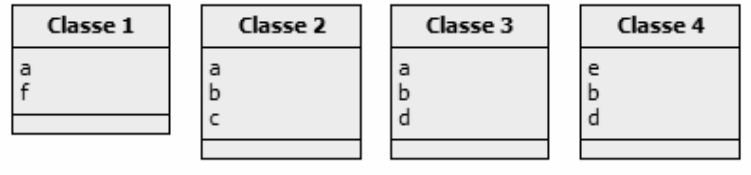


Figura 4: Modelo inicial de classes concretas

A relação de incidência *I* do contexto formal *K,* representando o conjunto das quatro classes ilustradas na Figura 4, e suas variáveis de instância é mostrada na Tabela 3. O contexto é desenhado como uma tabela de linhas e colunas, com as classes identificadas por números inteiros e as variáveis por letras.

TABELA 3

Contexto Formal

	a	b	c	d	e	f
1	x					x
2	x	x	x			
3	x	x		x		
4		x		x	x	

b. Modelagem inicial através de uma hierarquia prévia de classes

Considere que o projetista de software tenha no entanto estabelecido as entidades/objetos do modelo de classes com suas propriedades e inter-relacionamentos, conforme mostrado na Figura 5.

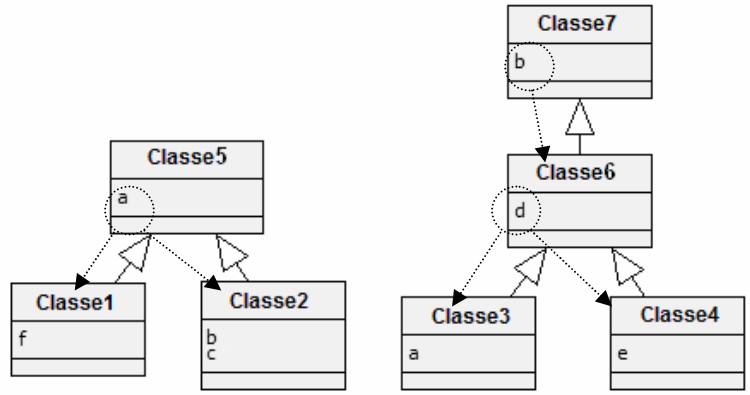


Figura 5: Modelo inicial de classes hierarquizado

Caso o novo modelo de classes possua os mesmos atributos que os ilustrados na Figura 4, e que dentro das relações de hierarquia existentes a transposição dos atributos das classes ascendentes para as classes descendentes (como mostrado na Figura 5) acarrete na definição do mesmo modelo de classes da Figura 4, conclui-se que o contexto formal gerado é idêntico. Dessa forma a tabela de contexto formal gerado a partir do modelo hierarquizado de classes da Figura 5 é o mesmo que o anteriormente apresentado na Tabela 3.

Suponha no entanto, que o projetista de software tenha estabelecido seu modelo de classes inicial divergente do que foi ilustrado nas Figuras 4 e 5. O novo modelo desenhado difere nas relações de hierarquia, como ilustrado na Figura 6.

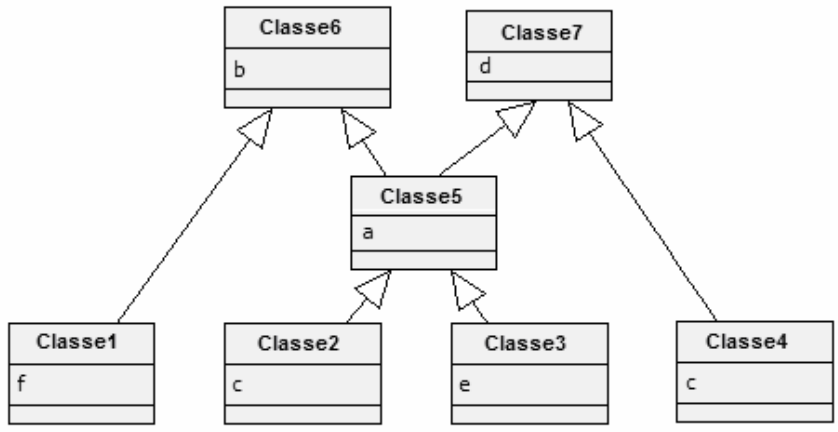


Figura 6: Modelo de classes contendo nova hierarquia

Neste caso infere-se que no processo de transposição dos atributos das classes ascendentes para as classes descendentes, o modelo de classes resultante, denominado aqui de classes concretas, tenha propriedades divergentes para uma ou mais classes do modelo. Sendo assim é de se afirmar que o contexto formal gerado também difere do foi apresentado na Tabela 3. O novo contexto formal gerado a partir da Figura 6 é ilustrado na Tabela 4.

TABELA 4

Contexto Formal obtido a partir de um novo modelo de classes

	a	b	c	d	e	f
1		x				x
2	x	x	x	x		
3	x	x		x	x	
4			x	x		

O contexto formal gerado a partir do modelo de classes definido inicialmente pelo projetista de software é a base para construção do reticulado de conceitos. A partir do reticulado de conceitos, um conjunto de transformações são realizadas para se chegar a um novo modelo de classes. Dessa forma conclui-se desde já que os resultados finais obtidos são distintos conforme se projeta o modelo de classes inicial.

3.2.2 Geração do Reticulado de Conceitos

Como o problema é organizar estas classes em uma hierarquia, um reticulado de conceitos é utilizado como um guia para o projeto. Para isso, cada conceito formal é interpretado como uma classe na hierarquia, e as ligações entre as classes são vistas como relações de especialização.

Na Figura 7 é apresentado um diagrama do Reticulado de Conceitos. Os rótulos atribuídos aos conceitos indicam em que classe um atributo em específico deve ser declarado, por exemplo, os atributos *a* e *b* são declarados em duas classes gerais que estão localizadas imediatamente abaixo da classe raiz da hierarquia. Para a hierarquia de classes, o conceito definido pelo nó *bottom*, localizado no ponto mais baixo do reticulado, é ignorado, uma vez que não tem utilidade, pois não representa informações de classes.

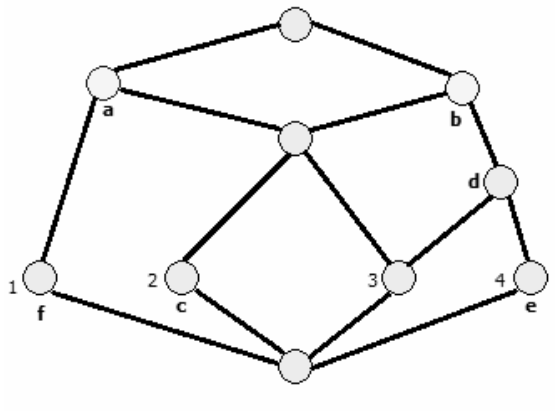


Figura 7: Reticulado de Conceitos para o Contexto Formal da tabela 3

A Figura 8 mostra a hierarquia na forma do reticulado de atributos fatorados, que corresponde à sua interpretação do reticulado de conceitos. As quatro classes iniciais permanecem na hierarquia, mas há menos atributos declarados nestas classes devido ao refatoramento produzido pelo reticulado de conceitos.

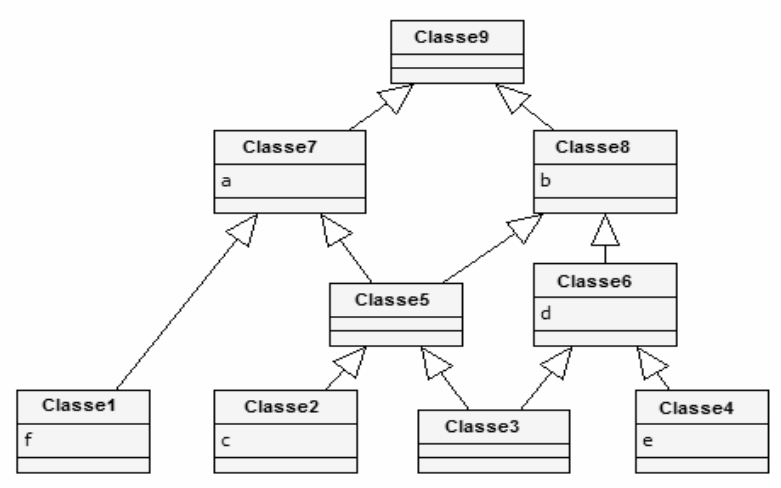


Figura 8: Hierarquia de Classes do Reticulado de Conceitos da Figura 4

Novas classes (classes 5 a 9) são adicionadas por possuírem atributos comuns. Estas classes são vazias porque instâncias são criadas somente para as quatro classes iniciais. A natureza da redução do rotulamento do reticulado de conceitos garante que cada atributo apareça exatamente uma vez na hierarquia. Os atributos de objeto nas classes concretas iniciais permanecem imutáveis. Contudo, parte delas são, agora, herdadas de algumas novas classes. De modo geral, todas as subclasses são especializações, desde que herdem os atributos de classes ancestrais sem qualquer exceção.

Do ponto de vista de um projetista de sistemas, utilizar esta hierarquia produzirá o mesmo efeito que se utilizadas as quatro classes iniciais. Portanto, a hierarquia gerada, pode ser interpretada como um refatoramento das especificações das quatro classes iniciais.

3.2.3 Eliminação de Herança Múltipla

Há um grande número de projetos que possibilitam minimizar redundância. O reticulado de conceitos alcança este objetivo minimizando o número de classes e a herança múltipla (para as linguagens alvo que não suportam herança múltipla). Isso é alcançado através do agrupamento de classes sempre que possível, como ilustrado na Figura 7, em que são mostradas duas classes de entrada.

O Reticulado de Conceitos da Figura 7 possibilita a refatoração dos atributos comuns *a* e *b* utilizando apenas a *Classe1* e *Classe2*. O modelo da Figura 9(a) se torna mais complexo, uma vez que contêm duas classes, uma para cada atributo, capturando assim as classes 1 e 2 em um modelo de herança múltipla. Em contrapartida, o projeto mostrado na Figura 9(b) é mais simples e fornece o mesmo critério de qualidade, ao evitar redundância e conformidade com a especialização, em um modelo de herança simples.

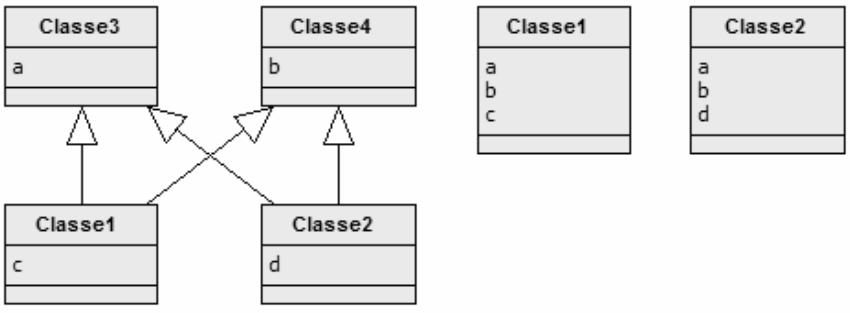


Figura 9(a): Modelo de classes contendo herança múltipla

Figura 9(b): Modelo de classes eliminada a herança múltipla

Figura 9: Eliminação da Herança Múltipla

A transformação de um modelo que contém herança múltipla em um modelo que contém herança simples consiste somente na cópia dos atributos de suas classes ascendentes em suas classes descendentes, onde há o relacionamento de herança múltipla. As classes ascendentes então deixam de existir no modelo depois de concluída as cópias, caso não haja outras ligações com outras classes.

3.2.4 Remoção de Classes Inconsistentes (ou Classes Vazias)

O reticulado de conceitos é uma representação de igualdades entre um conjunto de classes concretas. Como seu tamanho cresce rapidamente pode-se pensar em ignorar alguns de seus nós, de forma a manter sua estrutura gerenciável. Assim uma primeira idéia poderia ser a remoção de classes vazias que não declaram nenhuma propriedade ou método. Estas classes, comumente chamadas de classes vazias, poderiam ser removidas sem violar critérios de qualidade, ou seja, sem redundância e especialização. No exemplo da Figura 8, as classes vazias Classe5 e Classe9 poderiam ser omitidas. Embora a *Classe3* não declare nenhum atributo ela é mantida porque ela é uma classe *bottom*.

A estrutura resultante da remoção de todas as classes vazias denomina-se sub-hierarquia Galois (Wille, 1982), e corresponde ao conjunto simplificado de todos os conceitos de atributos e objetos do Reticulado de Conceitos. A figura 10 ilustra este novo reticulado.

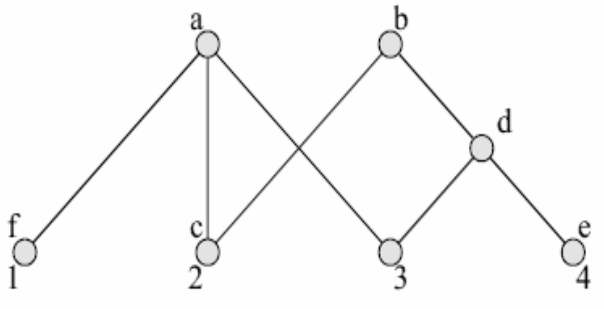


Figura 10: Reticulado de Conceitos resultante da eliminação de classes vazias

Considerando que a linguagem de programação suporte o conceito de herança múltipla, o novo modelo de classe resultante é mostrado na Figura 11.

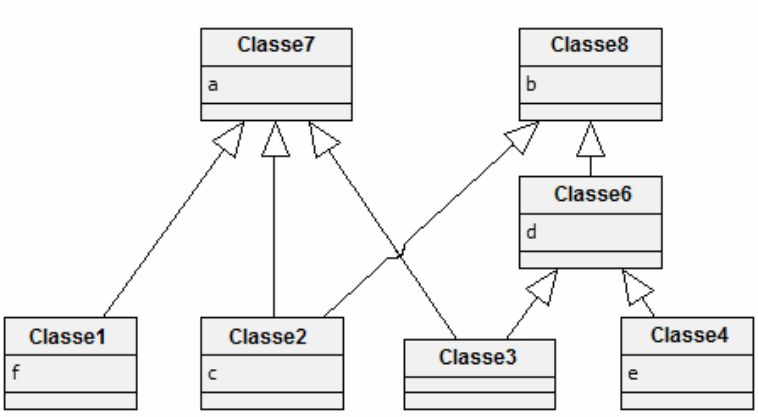


Figura 11: Modelo de classe resultante após a aplicação dos passos de refatoração

O modelo de classes mostrado na Figura 11 é o resultado do mapeamento das classes originais da Figura 4 em uma Tabela de Contexto, a qual por sua vez foi convertida em um Reticulado de Conceitos, e eliminadas as classes vazias. As classes vazias eliminadas, *Classe5* e *Classe9*, podem ser melhor identificadas através da Figura 8.

Contudo, se a definição inicial do projeto previsse que um dos pré-requisitos fosse a modelagem de classes sem suporte à herança múltipla, o diagrama de classe resultante seria modelado conforme a Figura 12 (vide transformação no item 3.2.3).

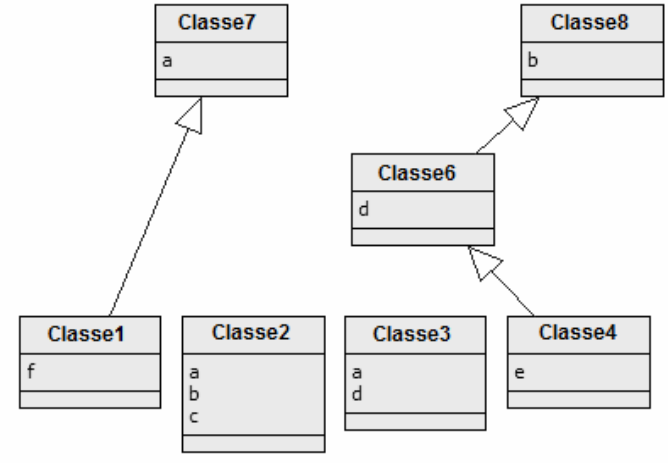


Figura 12: Modelo de classe resultante sem suporte à herança múltipla

3.2.5 Segmentação de classes onde atributos comuns têm semânticas diferentes

Os atributos de classe definidos pelo projetista de software podem possuir nomes idênticos, contudo suas propriedades podem ser diferentes, tornando-os assim semanticamente distintos. Este fato implica que, embora eles tenham o mesmo nome, eles não compartilham das mesmas características. Dessa forma, eles não deveriam ser resumidos a um simples atributo pertencente a uma nova classe ancestral, gerada no Reticulado de Conceitos.

A Figura 13 ilustra duas tabelas que exemplificam o mapeamento dos atributos de classes existentes. A Figura 13(a) descreve os atributos de cada classe, e a Figura 13(b) os atributos comuns, os quais estão identificados com os símbolos ● e ■. Neste exemplo foi considerado o modelo de classes da Figura 11, o qual suporte o conceito de herança múltipla.

Classe1	**a,f**
Classe2	**a,b,c**
Classe3	**a,b,d**
Classe4	**b,d,e**
Classe6	**b,d**
Classe7	**a**
Classe8	**b**

Figura 13(a): Mapeamento dos atributos às classes

	a	*b*	*c*	*d*	*e*	*f*
Classe1	x					x
Classe2	x	■	x			
Classe3	x	●		x		
Classe4		●		x	x	
Classe6		●		x		
Classe7	x					
Classe8		■				

Figura 13(b): Identificação das classes com os atributos comuns

No estágio do processo de refatoração das classes, o projetista de software deve intervir de forma a identificar os atributos que possuem o mesmo nome, e cujas propriedades são distintas. O resultado é mostrado na Figura 14.

Classe1	**a,f**
Classe2	**a,b1,c**
Classe3	**a,b2,d**
Classe4	**b2,d,e**
Classe6	**b2,d**
Classe7	**a**
Classe8	**b1**

Figura 14(a): Mapeamento dos atributos às classes

	a	*b1*	*b2*	*c*	*d*	*e*	*f*
Classe1	x						x
Classe2	x	x		x			
Classe3	x		x		x		
Classe4			x		x	x	
Classe6			x		x		
Classe7	x						
Classe8		x					

Figura 14(b): Segmentação dos atributos comuns e novos nomes

A identificação de atributos comuns demanda que seus nomes sejam alterados, tal como mostrado na Figura 14(a). Classes segmentadas são criadas, como mostrado na Figura 14(b).

Um novo modelo de classes é então obtido, onde *Classe2* herda o atributo renomeado *b1*, pertencente à *Classe8*, e uma nova classe é criada: *Classe9*, a qual contém o atributo renomeado *b2*. Ambos os atributos têm como origem o atributo *b*, oriundo do modelo de classes anterior. A Figura 15 apresenta a nova hierarquia de classes gerada, enquanto a Figura 16 apresenta o mesmo modelo contendo os atributos originais. Os dois modelos atingem igualmente o propósito deste passo de refatoração.

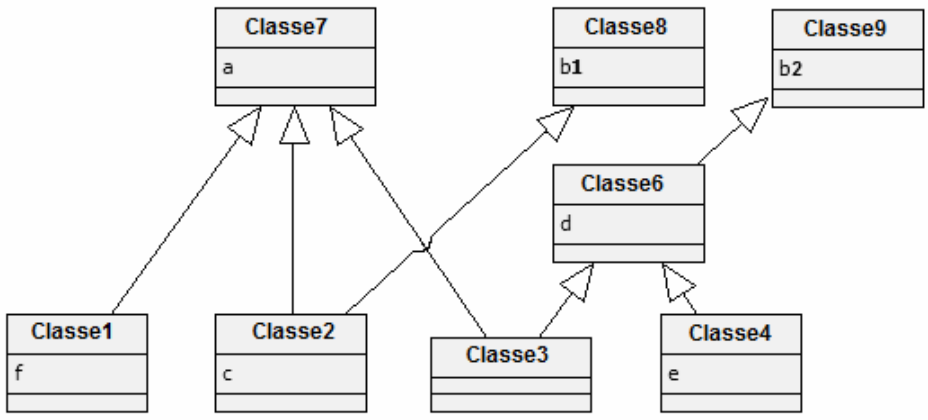


Figura 15: Novo modelo de classes resultante da segmentação dos atributos comuns

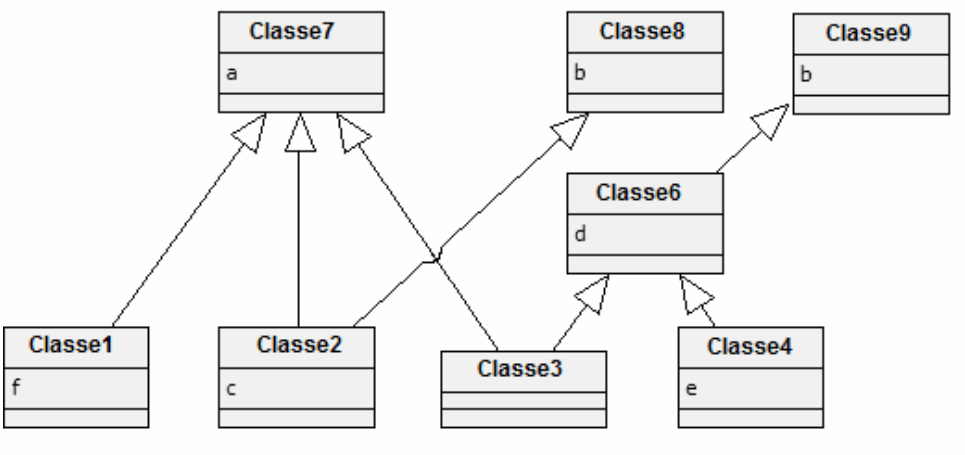


Figura 16: Modelo de Classes final contendo os atributos originais

4 ESTUDO DE CASO: SIGCRF – SISTEMA DE GERENCIAMENTO INTEGRADO DO CONSELHO REGIONAL DE FARMÁCIA DO ESTADO DE MINAS GERAIS

O estudo de caso proposto neste trabalho exemplifica a aplicação da metodologia em um sistema real, utilizado e mantido por uma empresa pública, que tem por finalidade a fiscalização das empresas e profissionais farmacêuticos no âmbito do Estado de Minas Gerais. O sistema SIGCRF – Sistema de Gerenciamento Integrado do Conselho Regional de Farmácia do Estado de Minas Gerais foi escolhido como apoio para aplicação da abordagem proposta neste trabalho, em virtude de ser um sistema de gestão mais amplo, concebido em OO e mantido há mais de dez anos.

4.1 SIGCRF – Visão Geral

O SIGCRF é o software de Gestão do Conselho Regional de Farmácia do Estado de Minas Gerais, concebido e mantido desde o ano de 2001, cujas funcionalidades em nível macro abrangem:

- Registro das Empresas e Profissionais Farmacêuticos
- Fiscalização das Atividades de Assistência do Responsável Técnico Farmacêutico
- Processos Fiscais e Éticos
- Processos Judiciais
- Gestão Financeira e Contábil
- Gestão de Compras, Contratos e Licitações
- Recursos Humanos
- Auditoria Interna

O SIGCRF é um software desenvolvido em Java, cujo Gerenciador de Banco de Dados utilizado é o SQL Server. Alguns indicadores da versão atual de seu projeto são listados a seguir, para maior compreensão da complexidade e amplitude do software (vide Anexo I):

- 23 (vinte e três) módulos
- 87 (oitenta e sete) classes
- 1736 (mil setecentos e trinta e seis) procedimentos
- 863 (oitocentas e sessenta e três) funções
- 118 (cento e dezoito) tabelas de bancos de dados
- 2.143.263 (dois milhões, cento e quarenta e três mil duzentos e sessenta e três) linhas de código (loc)

4.2 Requisitos

Para uma melhor aplicação da metodologia proposta neste trabalho, foi escolhido um sistema, cujo projeto foi concebido sob a abordagem de Orientação a Objeto. Um requisito inicial é a existência de um projeto de classes do software, estruturado por meio de uma hierarquia prévia, ou por meio de classes relacionadas em um mesmo nível.

Para obtenção de resultados mais significativos, os quais refatoram e otimizam o projeto inicial das classes, sugere-se como segundo requisito que o projeto do sistema seja de leitura mais complexa para o projetista. Apesar deste requisito exigir uma análise mais subjetiva, que depende de fatores como conhecimento e experiência prévia do projetista, alguns indicadores mensuráveis devem ser levados em consideração, como os relatados anteriormente para o SIGCRF.

4.3 Solução Proposta para uma Hierarquia de Classes Prévia

Aplicando-se a metodologia proposta com o objetivo de se obter a máxima refatoração das classes do SIGCRF, e uma nova hierarquização do modelo inicial, os passos abaixo foram realizados de forma sequencial:

a. Mapeamento do Diagrama de Classes em uma Tabela de Contexto;

b. Geração do Reticulado de Conceitos;

c. Remoção de Classes Inconsistentes (ou Classes Vazias).;

d. Segmentação das classes onde atributos comuns têm semântica diferente;

O primeiro passo enunciado faz referência ao mapeamento do diagrama de classes do projeto do SIGCRF em uma tabela de contexto. Como uma solução inicialmente proposta considerou-se o diagrama de classes original do software com todas as suas relações de hierarquia.

A metodologia deste trabalho propõe também um passo referente à eliminação de herança múltipla do projeto de classes, nos casos em que a linguagem alvo não a suporta. Contudo, como a linguagem em que o SIGCRF foi desenvolvida suporta o conceito de herança múltipla, este passo não é executado.

A fim de automatizar o processo de refatoração do modelo inicial do projeto de classes, o framework apresentado na Figura 3 foi utilizado.

4.3.1 Mapeamento do Diagrama de Classes em uma Tabela de Contexto

Tendo como base a documentação atual do sistema SIGCRF para este estudo de caso, o projeto das entidades/objetos do modelo de classes, com suas propriedades e inter-relacionamentos, é mostrado na Figura 17. Em virtude da elevada dimensão do diagrama de classes, somente recortes de maior significância ao entendimento do projeto e dos resultados alcançados foram destacados.

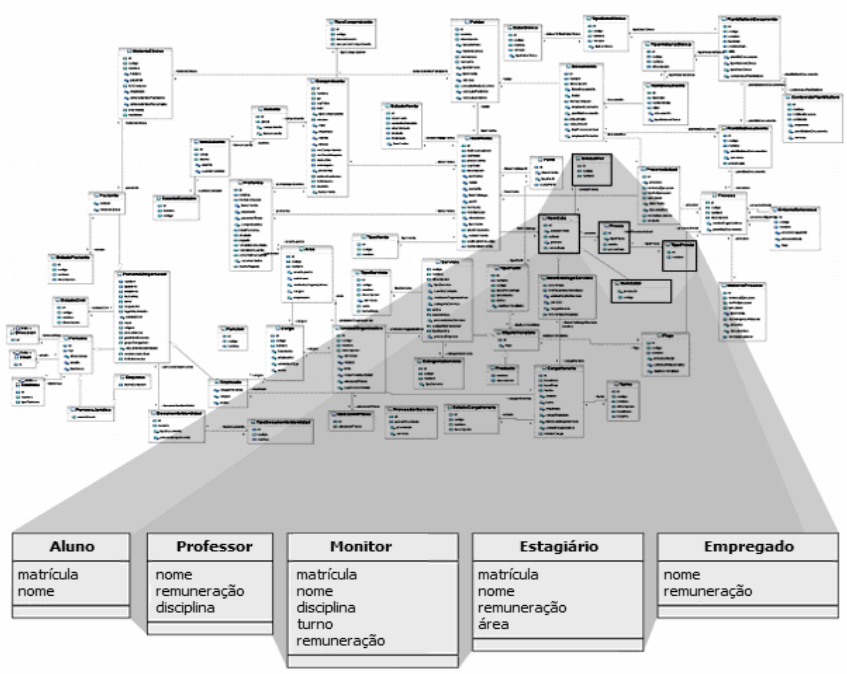


Figura 17: Projeto inicial de classes hierarquizado do SIGCRF

Como parte da primeira etapa de execução do framework AFC, o diagrama de classes do SIGCRF ilustrado na Figura 17, foi convertido para o padrão XMI, com o objetivo de mapear as entidades/objetos e atributos/métodos, em uma Tabela de Contexto. Um trecho do arquivo XMI gerado é ilustrado na Figura 18.

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
  <class xmlns="urn:nhibernate-mapping-2.2" name="SGC.Domain.Entities.Usuario" table="Usuario">
    <id name="Id">
      <column name="Id" />
      <generator class="identity" />
    </id>
    <property name="Aluno">
      <column name="Aluno" length="255" not-null="true" unique="true" />
    </property>
    <property name="Monitor">
      <column name="Monitor " length="255" not-null="true" />
    </property>
    <property name="Professor">
      <column name="Professor" length="255" not-null="true" />
    </property>
    <property name="Estagiário">
      <column name="Estagiário" not-null="true" />
    </property>
    <property name="Empregado">
      <column name="Empregado" length="255" not-null="true" />
    </property>
    <many-to-one class="SGC.Domain.Entities.Perfil" lazy="false" name="Perfil">
      <column name="Perfil_id" not-null="true" />
    </many-to-one>
  </class>
</hibernate-mapping>
```

Figura 18: Arquivo XMI gerado a partir do Diagrama de Classes

A partir da geração do arquivo XMI, uma leitura é realizada pelo framework AFC, para transposição dos atributos/métodos das classes ascendentes para as classes descendentes, de forma que as classes resultantes estejam no mesmo nível hierárquico, sem suas relações de generalização/especialização. A Figura 19 ilustra esta etapa de processamento do arquivo XMI para um segmento do projeto de classes do SIGCRF.

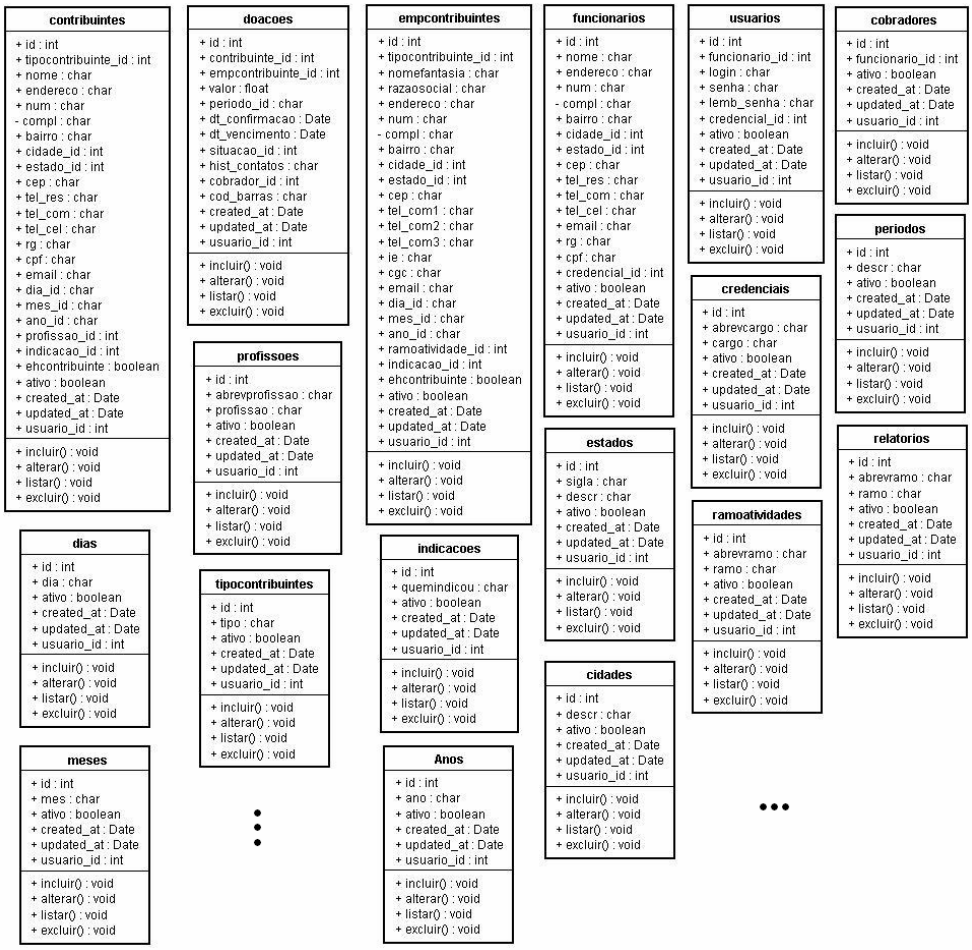
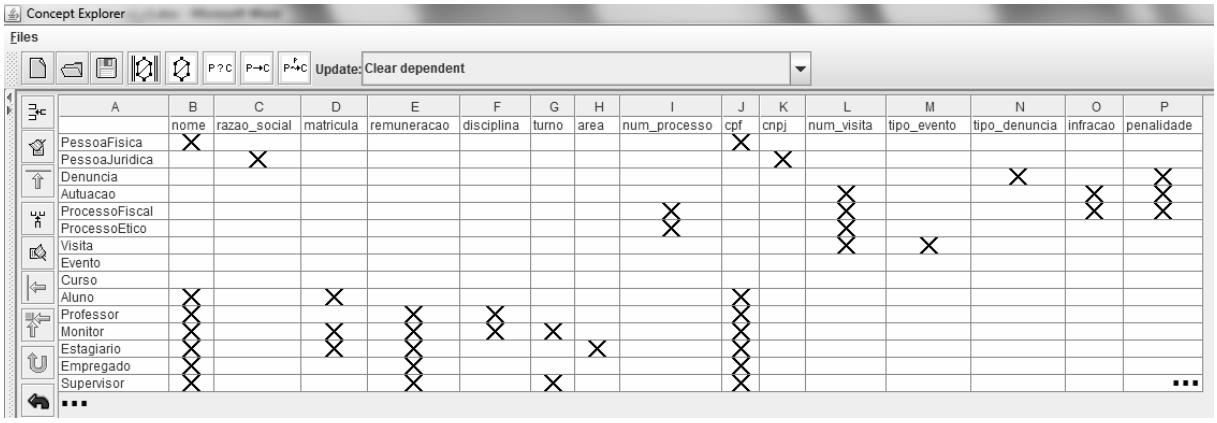


Figura 19: Diagrama de Classes transformado para um mesmo nível hierárquico

O Diagrama de Classes original foi transformado em um conjunto de classes concretas, não havendo perda dos atributos e métodos, contudo sem as relações de hierarquias preestabelecidas.

Um novo arquivo XMI é obtido, então em conformidade com o conjunto de

classes concretas, e transformado em uma Tabela de Contexto, a qual contém o mapeamento de todas as classes e atributos, bem como suas relações de incidência. A Figura 20 mostra um segmento da Tabela de Contexto gerada. Para fins de ilustração, os métodos não foram apresentados, contudo eles foram igualmente submetidos ao framework AFC.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
		nome	razao_social	matricula	remuneracao	disciplina	turno	area	num_processo	cpf	cnpj	num_visita	tipo_evento	tipo_denuncia	infracao	penalidade
A	PessoaFisica	X								X						
B	PessoaJuridica		X								X					
C	Denuncia													X		
D	Autuacao											X			X	X
E	ProcessoFiscal								X			X			X	X
F	ProcessoEtico								X			X				
G	Visita											X	X			
H	Evento															
I	Curso															
J	Aluno	X		X						X						
K	Professor	X			X	X				X						
L	Monitor	X		X	X	X	X			X						
M	Estagiario	X		X	X			X		X						
N	Empregado	X			X					X						
O	Supervisor	X			X		X			X						...
P	...															

Figura 20: Tabela de Contexto gerada a partir do arquivo XMI

A Tabela de Contexto gerada na Figura 20 representa um recorte das 87 classes existentes do projeto, sob a forma de linhas; e mais 795 colunas, as quais representam os atributos do modelo. Os atributos, cujos rótulos são idênticos, foram mapeados de forma individualizada, visto que podem possuir as mesmas propriedades. Os atributos comuns foram assinalados com **X** na tabela, os quais totalizam 117.

4.3.2 Geração do Reticulado de Conceitos

Baseado no segmento da Tabela de Contexto ilustrada na Figura 20, o Reticulado de Conceitos inicial do SIGCRF é gerado. A Figura 21 apresenta este reticulado.

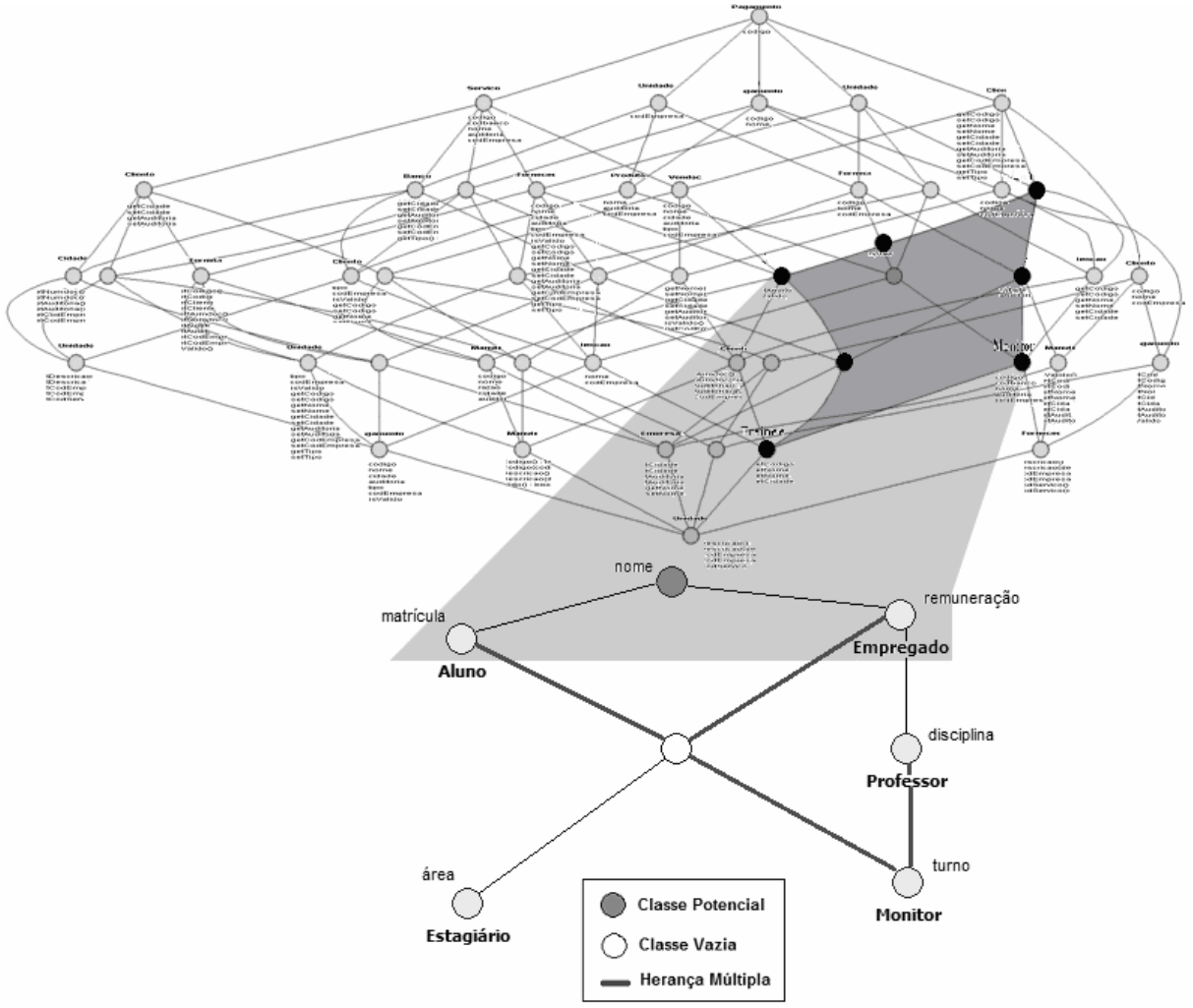


Figura 21: Reticulado de Conceitos gerado a partir da Tabela de Contexto com identificação de classes concretas, potenciais e inconsistentes

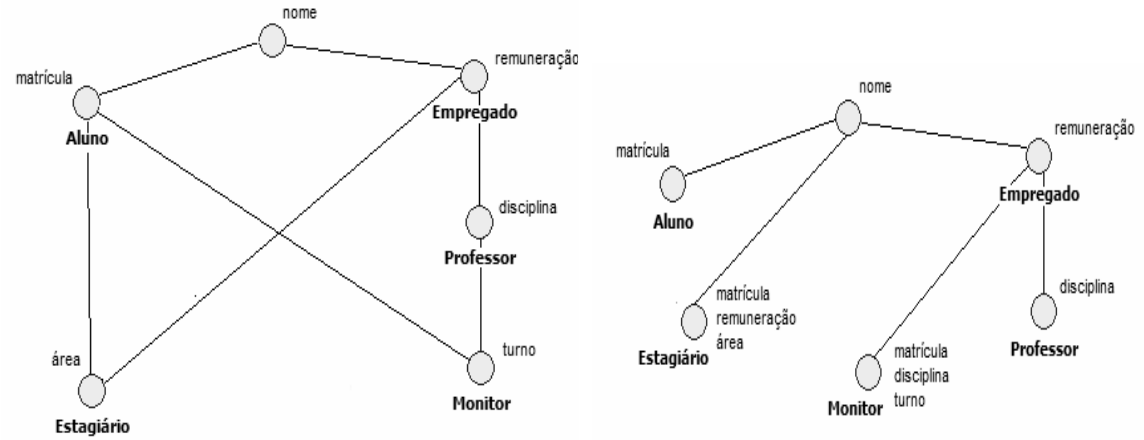
É importante observar que cada nó (ponto) do Reticulado de Conceitos é uma classe existente do modelo de classes inicial, uma classe em potencial, ou uma classe inconsistente, que pode ser removida do reticulado com os devidos ajustes das ligações existentes.

No Diagrama de Classes inicial do SIGCRF foram identificadas 87 classes. Constata-se a partir do Reticulado de Conceitos obtido, que o número de classes totaliza 101 classes, sendo 63 classes concretas, 31 classes em potencial e 17 classes inconsistentes (classes vazias).

4.3.3 Remoção de Classes Inconsistentes (ou Classes Vazias)

O próximo passo do framework AFC consiste na eliminação das classes inconsistentes identificadas, ou seja, a remoção de classes vazias que não declaram nenhuma propriedade ou método. Como o reticulado de conceitos é uma representação amplificada das classes, é possível que alguns de seus nós sejam ignorados, sem que haja violação dos critérios de qualidade desejáveis em projetos OO.

A Figura 22 ilustra o resultado da aplicação desta etapa de refatoração para o segmento destacado na Figura 21, sendo à direita com suporte herança múltipla e à esquerda com herança simples.



(a): Suporte à Herança Múltipla **(b): Suporte à Herança Simples**

Figura 22: Reticulado de Conceitos após Eliminação das Classes Inconsistentes

As classes inconsistentes, identificadas no Reticulado de Conceitos gerado para o SIGCRF, não possuem significado no contexto do modelo de classes, e portanto foram eliminadas.

4.3.4 Geração do Diagrama de Classes com Identificação das Classes Potenciais

As classes potenciais são nós do Reticulado de Conceitos que reúnem propriedades/métodos de outras classes, oriundos de nós descendentes do reticulado, sendo portanto candidatas a superclasses no modelo hierárquico. As propriedades e métodos identificados pelo framework AFC, que possuem mesmo valor semântico e finalidade, podem ser agrupados em classes mais gerais, as quais são ascendentes no modelo hierárquico.

A Figura 23 ilustra a criação de uma nova classe (Pessoa) ao segmento do modelo hierárquico gerado para o exemplo da Figura 17.

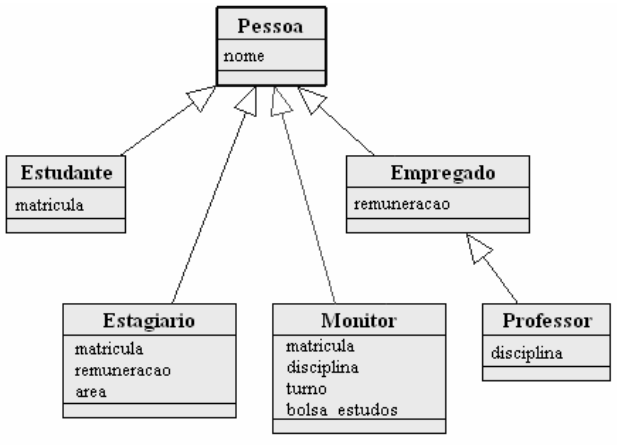


Figura 23: Acréscimo de Classe em Potencial ao Modelo

Na Figura 23, a classe em potencial *Pessoa* foi adicionada ao modelo, de forma que o atributo comum *nome* possa ser herdado pelas classes descendentes deste módulo do projeto.

A classe em potencial *Pessoa* estabelece uma nova relação de generalização no modelo, minimizando a redundância de classes e otimizando a etapa de projeto do sistema.

Ressalta-se aqui, que o framework AFC agrupa os atributos/métodos comuns das classes sem uma análise semântica dos mesmos, o que pode resultar, em alguns casos, na criação de superclasses incorretas. O próximo passo da etapa de refatoração vem a solucionar esta situação.

O Reticulado de Conceitos gerado para o SIGCRF aponta 31 classes em potencial ao novo modelo de classes, além das 87 classes preexistentes do projeto inicial modelado.

4.3.5 Segmentação das Classes baseado na Semântica de seus Atributos

O último passo do processo de refatoração das classes consiste na segmentação das mesmas, baseado na identificação semântica de seus atributos.

Conforme explanado anteriormente, os atributos de classe definidos pelo projetista de software podem possuir nomes idênticos, contudo suas propriedades podem ser diferentes, tornando-os assim semanticamente distintos. Isso implica que, embora eles tenham o mesmo nome, não compartilham das mesmas características. Sendo assim a classe que contém estes atributos deve ser segmentada em duas novas classes.

A Figura 24 ilustra duas tabelas que apresentam o mapeamento dos atributos das classes em realce da Figura 17. A Figura 24(a) descreve os atributos comuns de cada uma destas classes, os quais estão identificados com o símbolo **X**.

Estudante	nome, matricula,area
Professor	nome, matricula, disciplina
Monitor	nome, matricula, disciplina, turno, remuneracao
Estagiario	nome, matricula, remuneracao, area
Empregado	nome, turno, remuneracao

	nome	*matricula*	*assunto*	*turno*	*remuneracao*	*area*
Estudante	x	x				x
Professor	x	x	x			
Monitor	x	x	x	x	x	
Estagiario	x	x			x	x
Empregado	x			x	x	

Figura 24(a): Mapeamento dos atributos às classes

Estudante	nome, matricula,area
Professor	nome, matricula, disciplina
Monitor	nome, matricula, disciplina, turno, bolsa_estudos
Estagiario	nome, matricula, remuneracao, area
Empregado	nome, turno, remuneracao

	nome	*matricula*	*disciplina*	*turno*	*remuneracao*	bolsa_estudos	*area*
Estudante	x	x					x
Professor	x	x	x				
Monitor	x	x	x	x		x	
Estagiario	x	x			x		x
Empregado	x			x	x		

Figura 24(b): Identificação das classes com os atributos comuns

Neste estágio do processo de refatoração das classes, o projetista de software interveio de forma a identificar os atributos que possuem o mesmo nome e cujas propriedades são distintas. Tais atributos foram circulados para melhor visualização. A identificação de atributos comuns demanda que seus nomes sejam alterados e classes segmentadas são então criadas, como mostrado na Figura 24(b).

Nota-se que um novo modelo de classes é então obtido, onde o atributo *remuneracao* não possui mais significado para a classe *Monitor*. Este atributo é então segmentado em duas novas classes, para que um novo atributo, cujo nome mais adequado é *bolsa_estudos*, apareça na classe *Monitor*.

4.4 Análise

Conforme explanado anteriormente há duas formas do projetista de software modelar as entidades e objetos: através do desenho das classes concretas, sem qualquer hierarquia prévia existente, ou por meio de uma hierarquia pré-existente, como o modelo adotado neste capítulo, onde as propriedades e métodos são replicados das classes ascendentes para as classes descendentes.

A arquitetura proposta neste trabalho pode ser também aplicada a um conjunto de classes concretas, situadas em um mesmo nível, sem suas relações de hierarquia. É possível considerar que o projetista de software tenha estabelecido as entidades/objetos do modelo de classes do SIGCRF sem suas propriedades e inter-relacionamentos, conforme pode ser visto no segmento do módulo de Recursos Humanos do SIGCRF, na Figura 25.

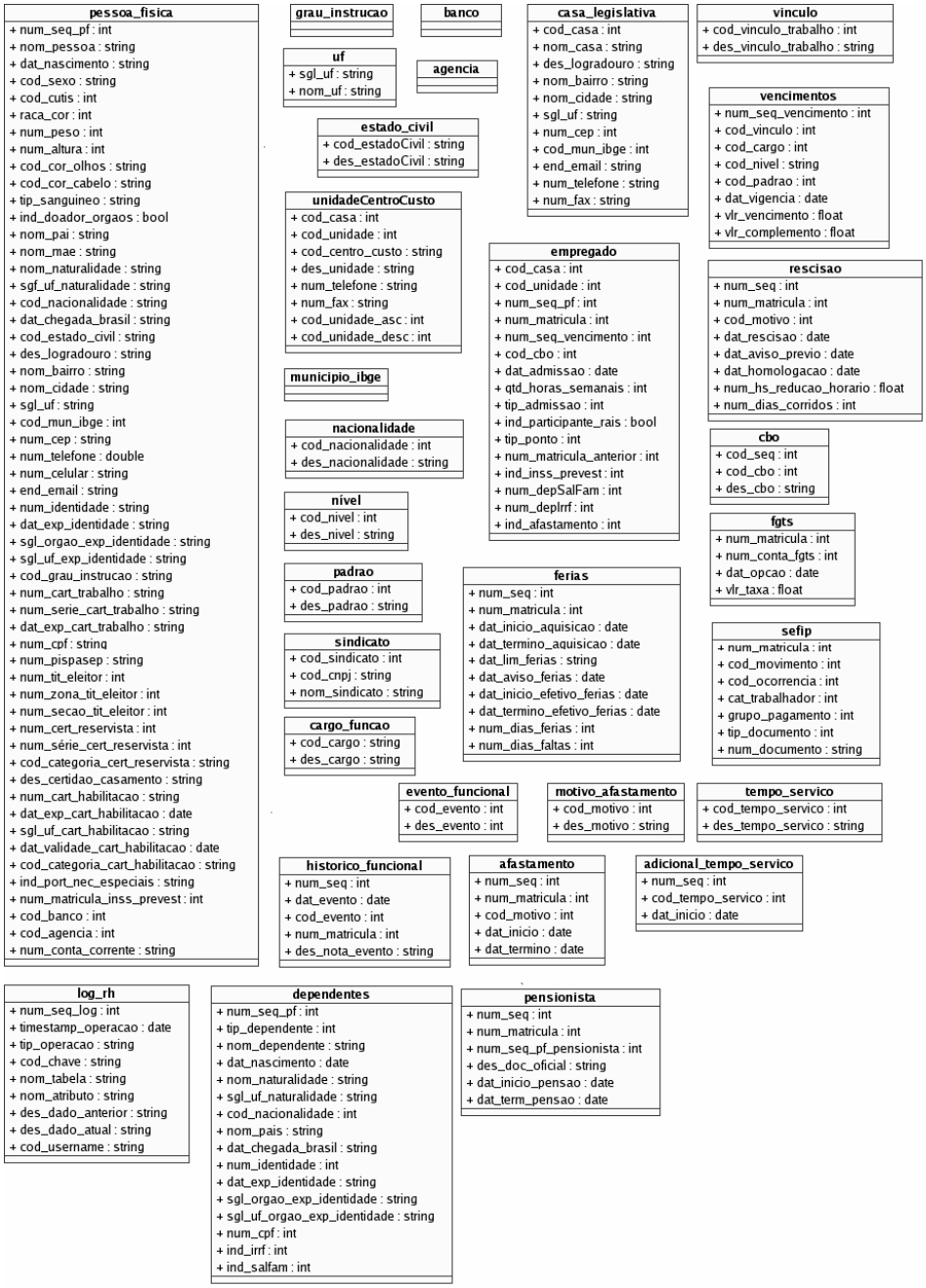


Figura 25: Conjunto de classes concretas e niveladas para o módulo RH

O modelo de classes da Figura 25 poderia ser interpretado como o conjunto exato de classes concretas do módulo RH que o SIGCRF contém, em outras palavras, estas classes são as únicas a produzir objetos neste módulo do sistema.

O modelo de classes da Figura 25 possui os mesmos atributos que o modelo inicial hierarquizado do SIGCRF (vide Anexo I), contudo houve a transposição dos atributos das classes ascendentes para as classes descendentes, de forma que a modelagem inicial dos atributos das classes permanecesse completa.

Percebe-se assim, que o contexto formal gerado, é idêntico para os modelos hierarquizado ou nivelado. Dessa forma os resultados obtidos para um conjunto de classes concretas niveladas do SIGCRF é o mesmo que para uma hierarquia prévia estabelecida para este mesmo software.

Através do Diagrama de Classes inicial do SIGCRF foram identificadas 87 classes. A partir da aplicação do passo de refatoração relativo à geração do Reticulado de Conceitos, o número de classes aumentou para 101, sendo 63 classes concretas, 31 classes em potencial e 17 classes inconsistentes (ou classes vazias).

Após a aplicação dos passos de refatoração de remoção das classes inconsistentes, identificação das classes em potencial, e segmentação das classes baseado na semântica de seus atributos, o resultado final obtido é o que se segue listado na Figura 26.

Recortes maximizados de casos particulares da Figura 26 são apresentados mais adiante, para uma análise mais detalhada.

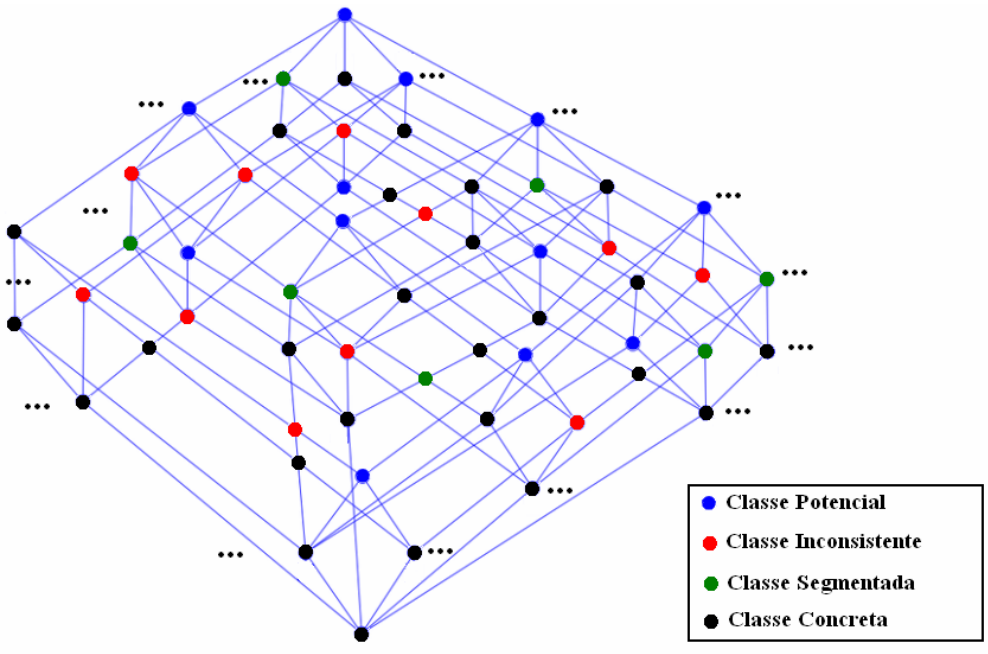


Figura 26: Reticulado de Conceitos obtido após as etapas de refatoração

A Figura 27 apresenta o diagrama de classes resultante após o término da execução dos passos do framework AFC. Recortes maximizados de segmentos da Figura 27 são apresentados mais adiante para uma análise mais detalhada.

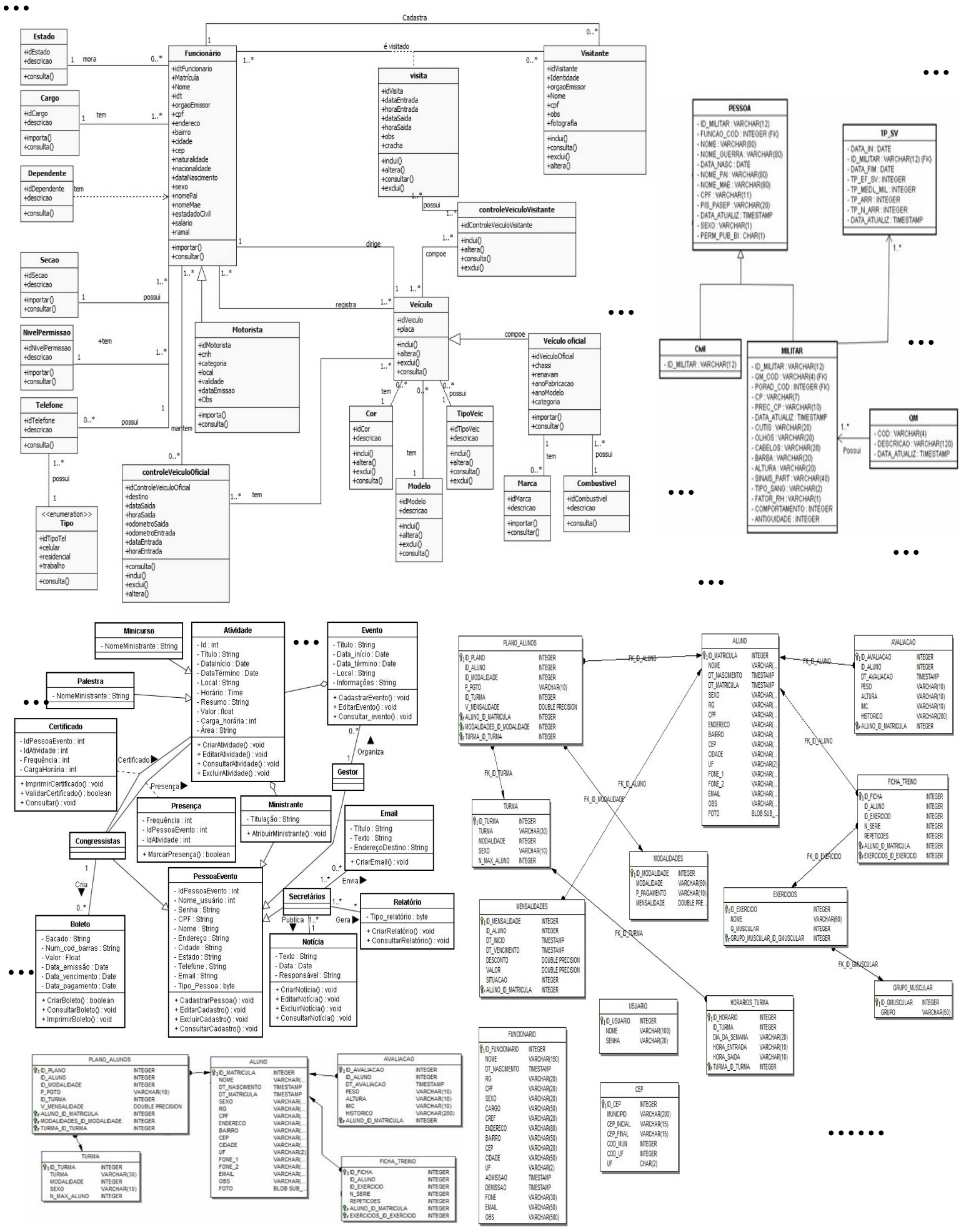


Figura 27: Novo modelo de classes obtido após as etapas de refatoração

Observa-se que o número de classes resultante é maior que o existente no modelo de classes inicialmente proposto pelo projetista. Contudo, nem toda classe original foi mantida no modelo final, em virtude da totalidade de seus atributos e métodos serem comuns a mais de uma classe, tornando-a assim uma classe inconsistente, e acarretando na criação de novas classes, que são potenciais classes ancestrais das existentes.

A Figura 28 ilustra um segmento específico do modelo de classes resultante da Figura 27, demonstrando a eliminação de classes a partir da refatoração realizada.

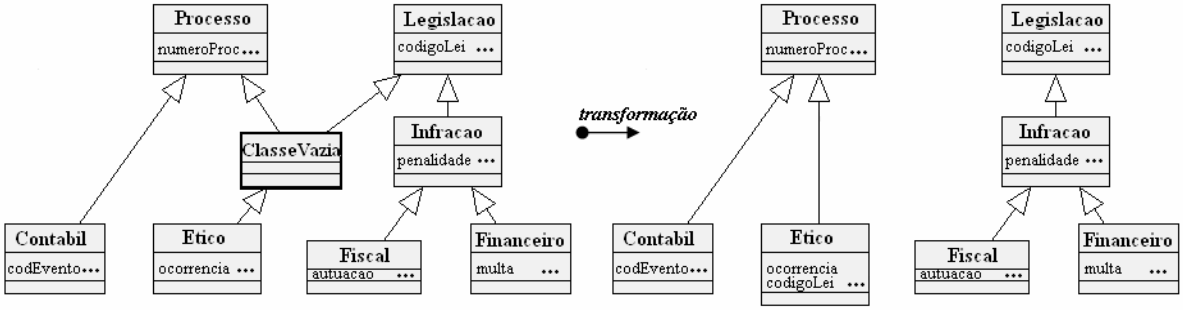


Figura 28: Eliminação de classe a partir da refatoração realizada

A classe denominada *ClasseVazia* apresentada na Figura 28 foi eliminada, visto que estabelece uma relação de generalização por meio de herança múltipla, o que não é desejável para o projeto de classes do SIGCRF.

Por outro lado, novas classes foram adicionadas ao projeto, tendo em vista que estabelecem uma nova relação de generalização ao modelo, como é o caso particular ilustrado na Figura 29.

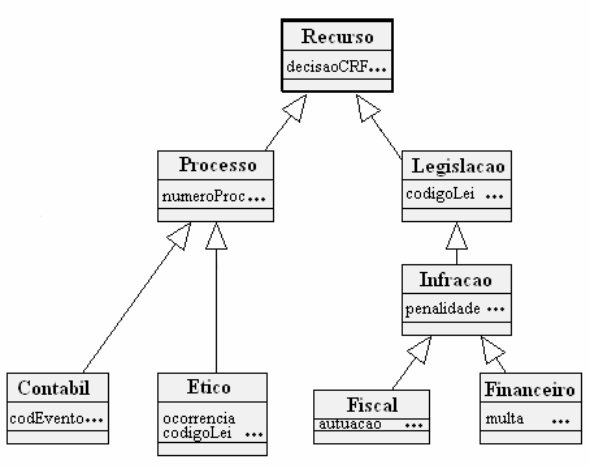


Figura 29: Nova relação de generalização no modelo de classes

Na Figura 29, a classe *Recurso* é uma nova superclasse, agrupando os atributos *decisaoCRF* e outros, de forma que possam ser herdados por suas classes descendentes.

No projeto final de classes do SIGCRF houve superioridade no aumento do número de classes que sua redução, o que caracteriza um projeto inicial do sistema ineficiente no que se refere ao uso otimizado de herança em DOO.

As classes inconsistentes ou nós vazios (vide seção 3.1.4), evidenciadas pelo Reticulado de Conceitos gerado, também foram eliminadas, visto que não adicionam ao modelo de classes nenhuma relação de generalização ou especialização. A Figura 30 ilustra esta eliminação, a partir de um segmento do reticulado do módulo de Treinamento do SIGCRF.

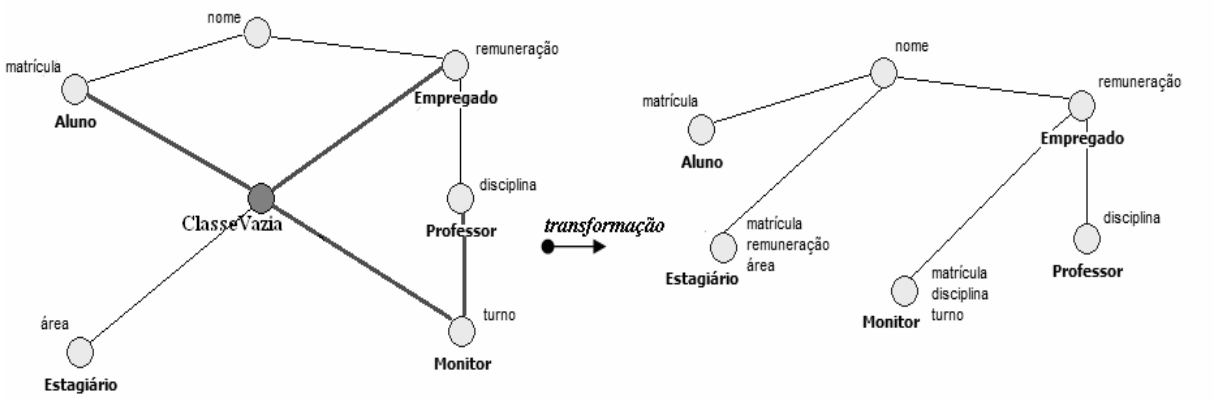


Figura 30: Eliminação de classe inconsistente do modelo

Nota-se a partir da Figura 30, que a classe vazia realçada em negrito, foi eliminada do Reticulado de Conceitos. Tal classe em evidência não possui relevância no que se refere ao uso de herança, visto que não contém atributos ou métodos que possam ser herdados por subclasses. Além disso, é objetivo deste trabalho, a otimização do número de classes, e a poda no reticulado de conceitos é uma das etapas de refatoração desejáveis.

Ressalta-se aqui, que classes inconsistentes podem, contudo, ser bastante úteis em modelos de classes onde somente herança simples é permitida, pois assim estas classes poderiam se tornar classes potenciais ao realizar mais ligações de ascendência ao modelo. Quando isso se faz necessário, convém ao projetista reduzir a eliminação de classes inconsistentes, e avaliar se tais classes podem ser transformadas em classes em potencial no modelo final. No projeto do SIGCRF, a eliminação das classes inconsistentes foi mantida sem prejuízo dos resultados encontrados.

Em menor número, mas com resultados significativos, 8 classes preexistentes foram segmentadas, devido à divergência semântica de seus atributos (vide seção 3.1.5), ainda que seus rótulos fossem originalmente idênticos. A Figura 31 ilustra um caso particular de segmentação de classes, proveniente do módulo de Treinamento do projeto de classes do SIGCF.

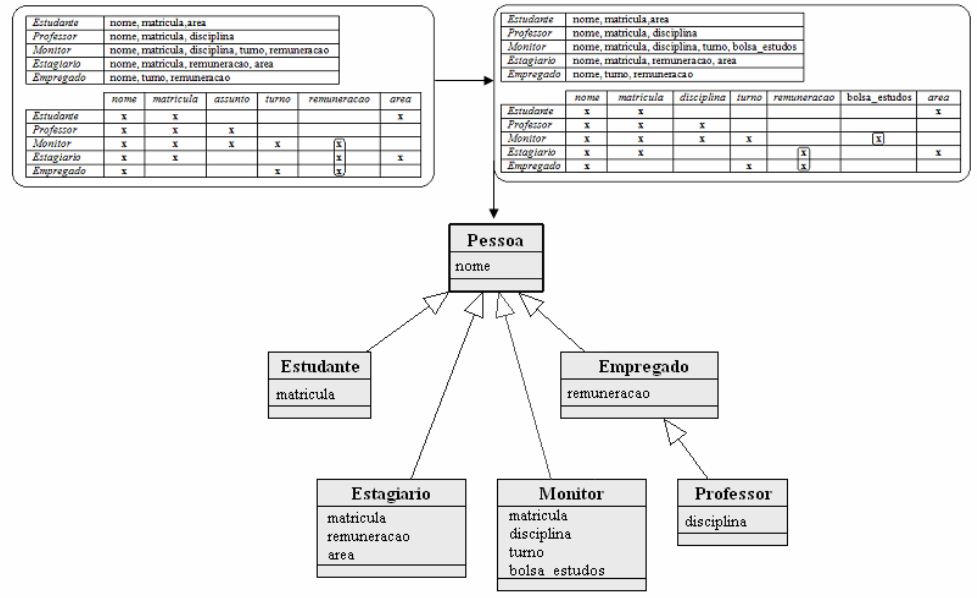


Figura 31: Segmentação de classes do modelo

O atributo *remuneracao* realçado nas tabelas da Figura 31 possui o mesmo rótulo para as classes *Monitor*, *Estagiário* e *Empregado*, contudo seu significado é distinto para elas. Enquanto o atributo *remuneracao* das classes *Estagiario* e *Empregado* se referem ao salário dos mesmos, o que pertence à classe *Monitor* se refere à uma ajuda de custo em forma de bolsa de estudos. Como os atributos são semanticamente diferentes, torna-se necessária a segmentação das classes de forma a representá-los adequadamente.

Observa-se ainda, como resultados encontrados neste trabalho, a redução de 17% do número de atributos e métodos declarados nas classes, o que sugere um nível mais alto de especialização do modelo obtido.

Outro indicador do projeto final de classes se refere à profundidade hierárquica das classes. Como o DOO do SIGCRF não foi remodelado ao longo de sua existência, não houve uma análise mais aprofundada por parte dos analistas e projetistas quanto a eventuais novas relações de herança geradas.

Em virtude disso, os resultados obtidos quanto à profundidade hierárquica gerou uma maior amplitude na cadeia, após aplicação da metodologia. É desejável em DOO, que a profundidade nas relações de hierarquia das classes não seja demasiada grande, de forma a não generalizar conceitos que poderiam ser mais facilmente interpretados quando agrupados em classes inferiores, e nem tão pequenas, para que não se deixe de obter todos os benefícios do conceito de herança, como maior reuso e coesão, e menor acoplamento (Sommerville, 2011).

A complexidade computacional para execução do algoritmo de execução do framework possui no pior caso, complexidade O(|conceitos||O||A'|^2) (Carpineto; Romano, 2004). Levando em consideração a operação mais cara do algoritmo (extração dos conceitos formais), para cada conceito formal gerado (|conceitos|), é preciso produzir seus subconceitos e percorrer os mesmos. Para produzir subconceitos, recorre-se aos operadores de derivação (|O||A'|), no máximo |A'| vezes. É interessante mencionar que o número máximo de conceitos formais no reticulado de um contexto formal (O,A',R) é $2^{|A'|}$ e o reticulado de conceitos cresce, portanto, exponencialmente, dependendo da incidência do contexto formal.

A Figura 32 ilustra o gráfico de desempenho de execução do algoritmo implementado para o framework AFC, em função do tempo (em segundos) e objetos

de entrada (classes). Os testes foram executados em um microcomputador cujo processador é Intel® Core™ i5 3ª Geração 3Ghz e com 4Gb de memória principal em ambiente operacional Windows 7.

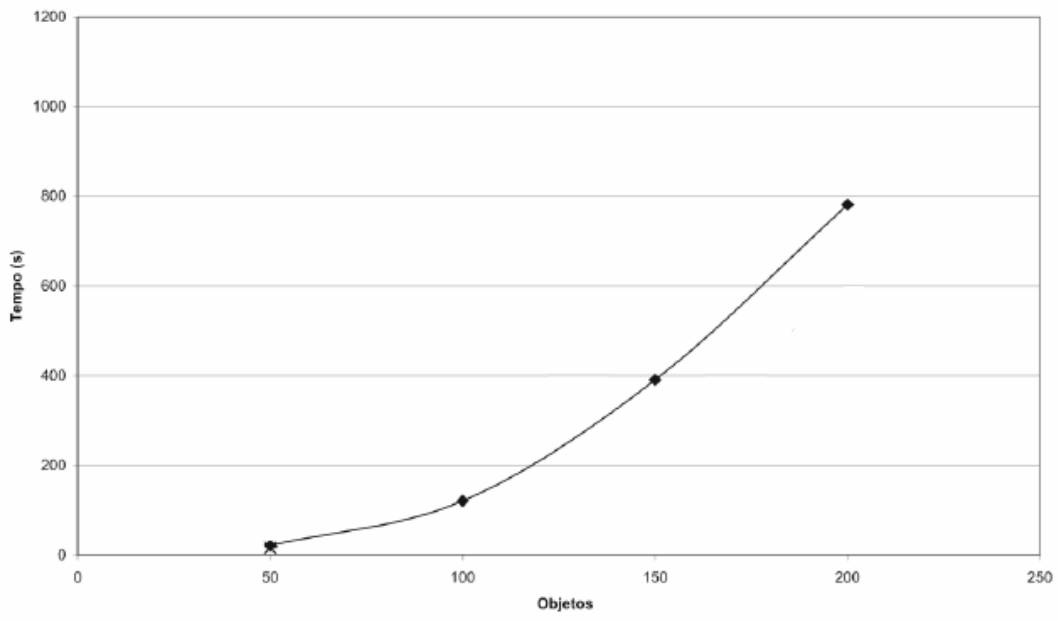


Figura 32: Desempenho de execução do algoritmo AFC Framework

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou o embasamento teórico e exemplo da aplicação da Análise Formal de Conceitos na refatoração e minimização de redundância de classes em Projetos Orientados a Objeto. Com o uso de um framework apropriado é possível automatizar e otimizar a etapa de projeto de um sistema, mantendo as características inerentes aos modelos de classes orientados a objeto.

O processo de refatoração de classes através da metodologia proposta se mostrou eficaz no que diz respeito à melhor compreensão do problema, uma vez que resulta em uma re-hierarquização do modelo, aproximando-se das características desejáveis de um projeto orientado a objetos.

Em alguns modelos de classes, o processo de refatoração deve ser realizado de forma iterativa, a fim de se obter melhores resultados. Com o uso da teoria da Análise Formal de Conceitos foi possível revelar abstrações do domínio do problema antes não detectadas, fornecendo um modelo de classes de fácil entendimento, reuso e manutenção.

Foi constatado através da aplicação do framework, que as relações de generalização se tornaram mais elevadas quando existe um grande número de atributos comuns, e pouca relação inicial de especialização no modelo. Mas, é possível com o uso deste arcabouço, que o projetista de software não se preocupe com as relações originais de hierarquia em seu modelo, e que projete automaticamente os diagramas levando em consideração a existência ou não de herança múltipla.

Além disso, o framework desenvolvido neste trabalho, possibilita que a qualquer instante uma nova classe seja inserida ou removida do modelo, de forma que a nova entrada lida gere, como saída, um novo diagrama de classes da UML, coerente com o problema que se está modelando.

Uma limitação da abordagem proposta está na intervenção do projetista para distinguir a semântica dos atributos e métodos das classes de seu projeto. Sem sua intervenção nesta etapa de refatoração possivelmente um modelo incoerente seria gerado, mas o processo se torna mais lento visto que é realizado de forma semi-automática.

Outra limitação é a transformação da herança múltipla em simples. A abordagem proposta seleciona uma das classes ancestrais apenas e replica os atributos e métodos das demais.

Como trabalhos futuros há possibilidade de automação da segmentação de classes, onde atributos comuns possuem diferentes semânticas. Outro trabalho relevante seria a geração automática de código fonte a partir das classes geradas e, a criação de uma biblioteca de classes, ou seja, um repositório que pudesse ser utilizado em novos projetos.

A aplicação de outras técnicas como a RCA (Huchard, 2011) também pode ser utilizada, visando a otimização da etapa referente à geração do Reticulado de Conceitos e consequentemente do projeto final de classes gerado.

Além disso, possibilitar a intervenção do projetista quando da transformação de herança múltipla em simples.

REFERÊNCIAS

ARÉVALO G., DUCASSE S. et al. GENERATING A CATALOG OF UNANTICIPATED SCHEMAS IN CLASS HIERARCHIES USING AFC. **Information Software Technology**, v. 2, p. 1145–1187, 2010.

ARÉVALO G., DUCASSE S. et al. UNDERSTANDING CLASSES USING X-RAY VIEWS**. In Proceedings of 2nd. MASPEGHI IEEE/ACM International Conference on Automated Software Engineering (ASE)**, v. 3, p. 2–18, 2003.

ARÉVALO G., FALLERI J. et. al. BUILDING ABSTRACTIONS IN CLASS MODELS. **AFC in MDA Models**, v. 13, p. 503-527, 2006.

BOOCH G. OBJECT ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS. **Second Edition. Assison-Wesley**, v. 4, p. 112-187, 1994.

CARPINETO C., ROMANO G. CONCEPT DATA ANALYSIS: THEORY AND APPLICATIONS. **John Wiley and Sons**, v. 12, p. 214-233, 2004.

DAO M., HUCHARD M. et al. IMPROVING GENERALIZATION LEVEL IN UML MODELS ITERATIVE CROSS GENERALIZATION IN PRACTICE. **Conceptual Structures at Work**, Lecture Notes in Computer Science, v.3127, 2004, pp 346-360

DAVEY B. A. AND PRIESTLEY H. A. INTRODUCTION TO LATTICES AND ORDER. **Cambridge Mathematical Textbooks, 2nd edition**, v. 13, p. 145-187, 2001.

FALLERI J., HUCHARD M. et. al. A GENERIC APPROACH FOR CLASS MODEL NORMALIZATION. **In Proceedings of 2nd. LISBON IEEE/ACM International Conference on Automated Software Engineering (ASE)**, v.64, p. 225-274, 2008.

GANTER, B. FORMAL CONCEPTS ANALYSIS TEORY: ALGORITHMIC ASPECTS. **Contributions to the 11th Conference on Concepts Practice**, v. 87, p. 157-221, 2003.

GANTER B. TWO BASIC ALGORITHMS IN CONCEPT ANALYSIS. **Technical Report, TU Darmstadt, Germany**, v. 12, p. 214-312, 1984.

GANTER B. AND WILLE R. FORMAL CONCEPT ANALYSIS: MATHEMATICAL FOUNDATIONS. **Springer-Verlag**, v. 18, p. 114-178, 1999.

GLINZ, M. ON NON-FUNCTIONAL REQUIREMENTS. **In Proceeding of 15th IEEE International Requirements Engineering Conference (RE)**, v. 35, p. 08–26, 2007.

GODIN R. and MILI H. BUILDING AND MAINTAINING ANALYSIS-LEVEL CLASS HIERARCHIES. **In Proceedings of OOPSLA**, p. 374–410, 1993.

GODIN R. and T.T. Chau. COMPARAISON D'ALGORITHMES DE CONSTRUCTION DE HIERARCHIES DE CLASSES. **L'Objet**, v.18, p. 311–338, 2000.

GRIGORIEV P. AND YEVTUSHENKO S. QUDA: APPLYING FORMAL CONCEPT ANALYSIS IN A DATA MINING ENVIRONMENT. **In Peter Eklund, editor, Concept Lattices. 2nd Intl. Conference on Formal Concept Analysis, ICAFC Proceedings**, v. 59, p. 257-282, 2004.

GUEDES G. UML 2: UMA ABORDAGEM PRÁTICA. **Novatec**, v. 3, p. 132-167, 2011.

HUCHARD M., HACENE R. et al. RELATIONAL CONCEPT DISCOVERY IN STRUCTURED DATASETS. **Ann. Math. Artificial Intelligence**, v. 10, p. 32–76, 2007.

HUCHARD M., HERVÉ D. et. al. GALOIS LATTICE AS A FRAMEWORK TO SPECIFY BUILDING CLASS. **In Proceedings of Theoretical Informatics and Applications**, v17, p. 511-548, 2000.

JOSHI P. and JOSHI R. CONCEPT ANALYSIS FOR CLASS COHESION. **In Proceedings of European Conference on Software Maintenance and Reengineering (CSMR)**, v. 36, p. 207–240, 2009.

KESTER Q. APPLICATION OF FORMAL CONCEPT ANALYSIS TO VISUALIZATION OF THE EVALUATION OF RISKS MATRIX IN SOFTWARE ENGINEERING PROJECTS. **International Journal of Science, Engineering and Technology Research (IJSETR)**, v. 2, p. 4-6, 2013.

MOHA N., HACENE A, et al. REFACTORINGS OF DESIGN DEFECTS USING RELATIONAL CONCEPT ANALYSIS. **In Proceedings of MADRID IEEE/ACM International Conference on Automated Software Engineering (ASE)**, v. 3, p. 114-151, 2008.

NILANDER R. and ZÁRATE L., HANDLING LARGE FORMAL CONTEXTS WITH SUPPORT OF DISTRIBUTED SYSTEMS. **In Proceedings of ladis International Conference**, v. 22, p. 1-15, 2011.

OOSTHUIZEN G. D. THE USE OF A LATTICE IN KNOWLEDGE PROCESSING. **PhD thesis, University of Strathclyde**, p. 78-106, 1988.

OOSTHUIZEN G. D. AND MCGREGOR D. R. INDUCTION THROUGH KNOWLEDGE BASE NORMALIZATION. **In Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)**, v. 12, p. 396–401,1988.

POELMANS J., KUZNETSOV S, IGNATOV D, DEDENE G. FORMAL CONCEPT ANALYSIS IN KNOWLEDGE PROCESSING: A SURVEY ON MODELS AND TECHNIQUES. **In Proceedings of the 4th International Conference on Concept Lattices and their Applications (CLA)**, v. 13, p. 123-138, 2012.

PRISS U. FORMAL CONCEPT ANALYSIS IN INFORMATION SCIENCE. **Annual Revision Information Science & Technology, John Willey & Sons,** v.40, n. 1, p. 521-543, 2007.

RAPICAULT P. and NAPOLI A. EVOLUTION D'UNE HIERARCHIE DE CLASSES PAR INTERCLASSEMENT. **In Proceedings of Barcelona L'Objet**, v. 32, p. 332-358, 2001.

RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., AND LORENSEN W. OBJECT ORIENTED MODELING AND DESIGN. **Prentice Hall**, v.9, p. 211-232, 1991.

SAFAVI A. AND SHAIKH M. EFFORT ESTIMATION MODEL FOR EACH PHASE OF SOFTWARE DEVELOPMENT LIFE CYCLE. **Handbook of Research on E-Services in the Public Sector: E-Government Strategies and Advancements**, v. 11, p. 271-302, 2011.

SNELTING G. and TIP F. SEMANTICS-BASED COMPOSITION OF CLASS HIERARCHIES. **In Proceedings of the 16th European conference on Object-Oriented Programming (ECOOP) London**, v. 24, p. 221-263, 2002.

SNELTING G. and TIP F. UNDERSTANDING CLASS HIERARCHIES USING CONCEPT ANALYSIS. **Association for Computing Machinery** (**ACM) Rome**, v. 21, p. 114-137, 2000.

SOMMERVILLE I. ENGENHARIA DE SOFTWARE, 9ª EDIÇÃO. **Pearson Education**, v.3, p. 214-263, 2011.

USMAN B., ANQUETIL N. et al. A CATALOG OF PATTERNS FOR CONCEPT LATTICE INTERPRETATION IN SOFTWARE REENGINEERING. **In Proceedings of International Conference on Software Engineering and Knowledge Engineering (SEKE) Boston**, v. 23, p. 68-103, 2012.

VALTCHEV P., GROSSER D. et al. GALICIA: AN OPEN PLATFORM FOR LATTICES. **In Proceedings of Aldo de Moor, 11th Conference on Conceptual Structures (CCS) Paris**, v.32, p. 212-231, 2003.

WILLE R. DYADIC MATHEMATICS – ABSTRACTIONS FROM LOGICAL THOUGHT. **In K. Denecke, M. Ern´e**, v. 7, p. 167-221, 2004.

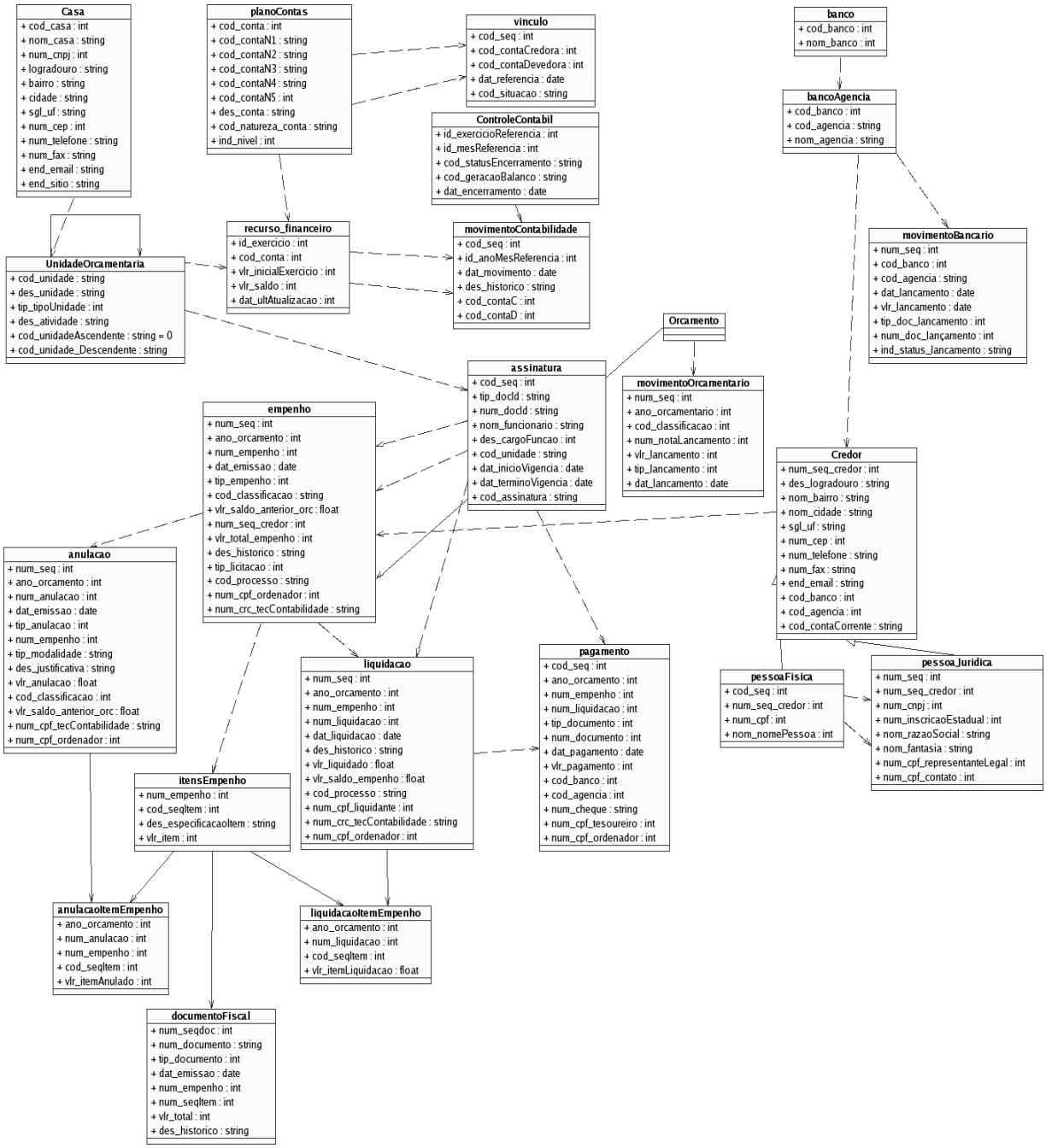
WILLE R. RESTRUCTURING LATTICE THEORY: AN APPROACH BASED ON HIERARCHIES OF CONCEPTS. **In Rival: Ordered sets. Reidel, Dordrech**, v. 11, p. 62-91, 1982.

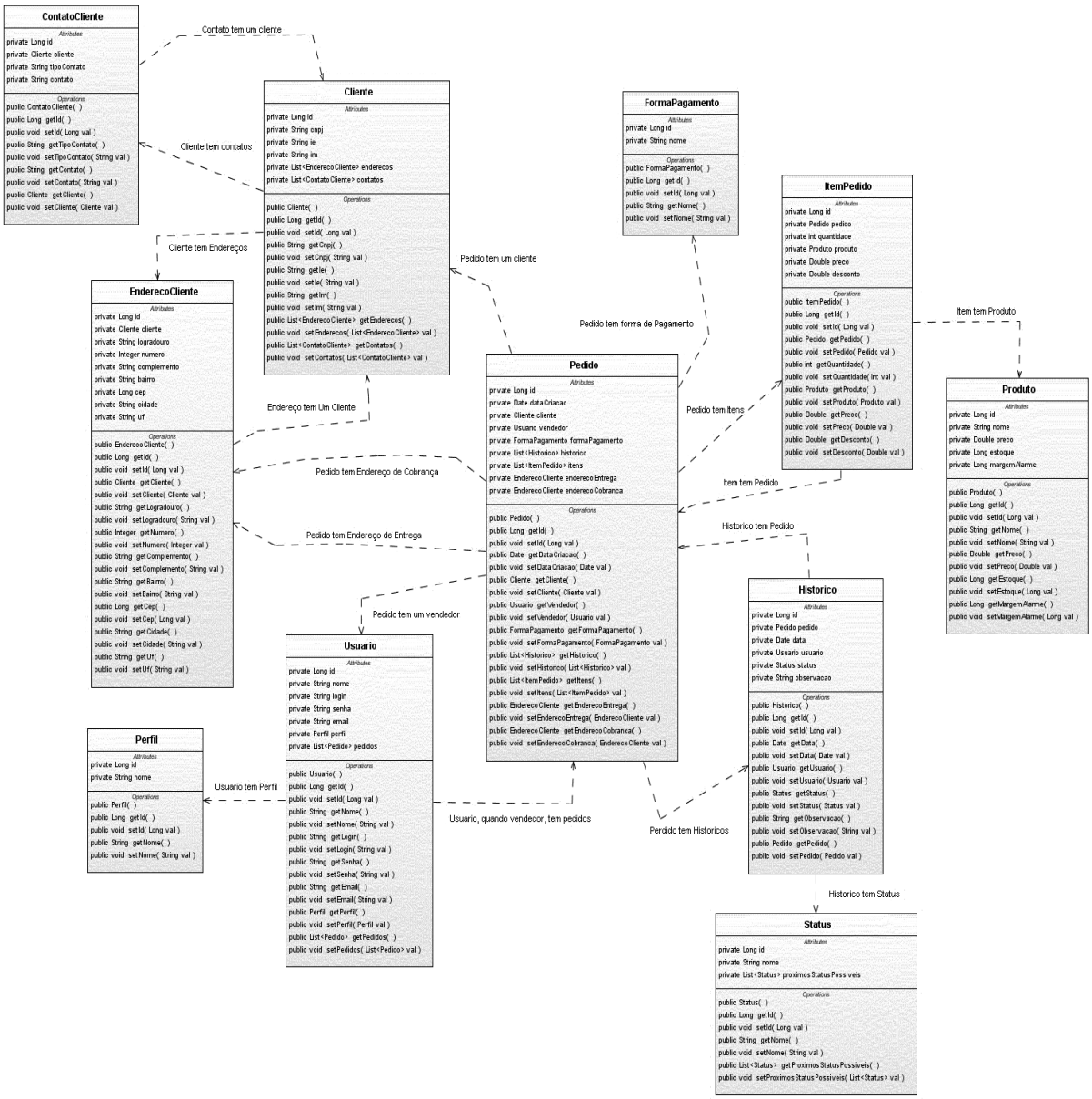
WOLFF K. AND YAMEOGO W. TURING MACHINE REPRESENTION IN TEMPORAL CONCEPT ANALYSIS. **In the Proceedings of the 3rd International Conference on Formal Concept Analysis (CFCA) Otawa**, p. 263-314, 2005.

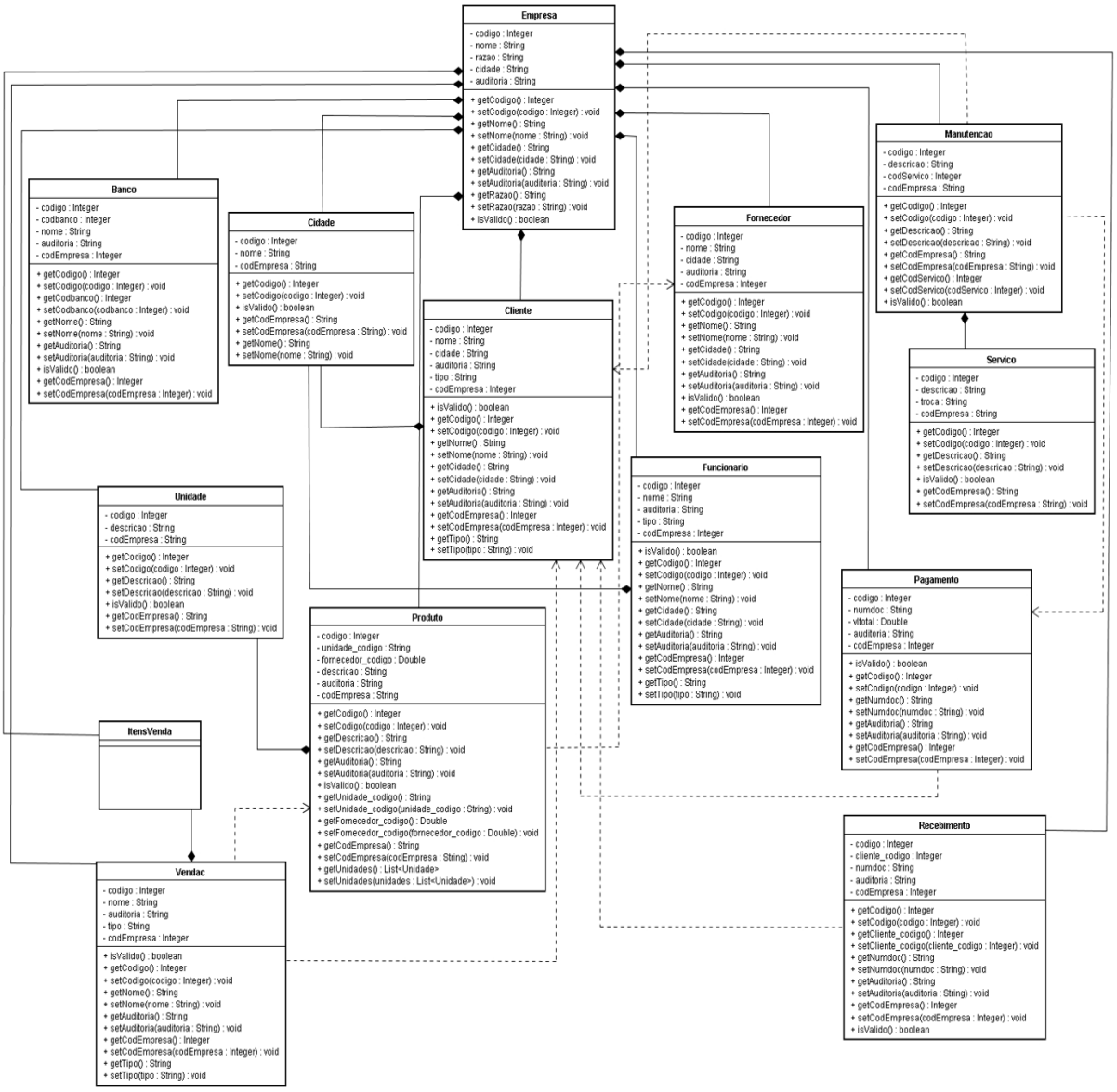
YEVTUSHENKO S. SYSTEM OF DATA ANALYSIS “CONCEPT EXPLORER”. **In Proceedings of the 7th National Conference on Artificial Intelligence KII Boston**, v. 14, p. 74-103, 2000.

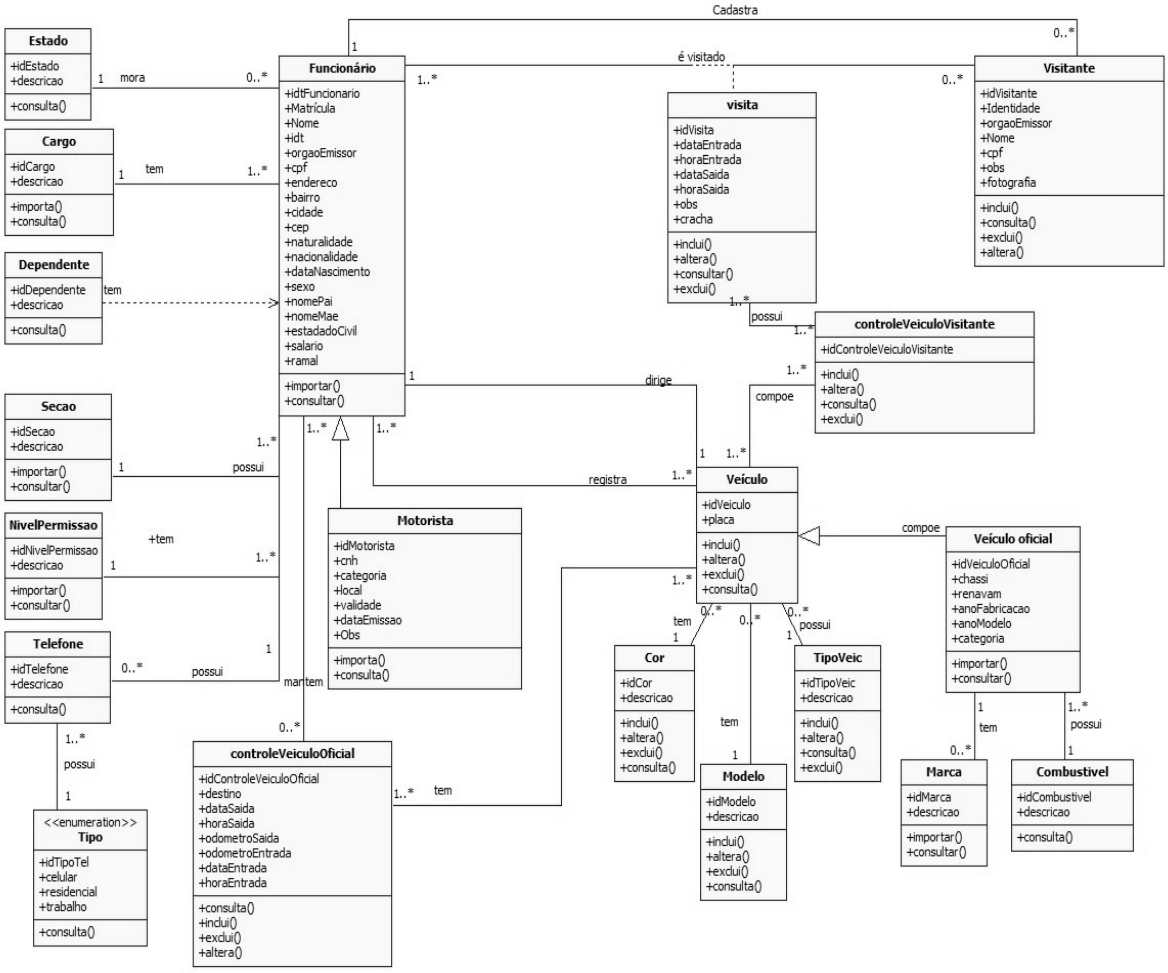
ANEXO I

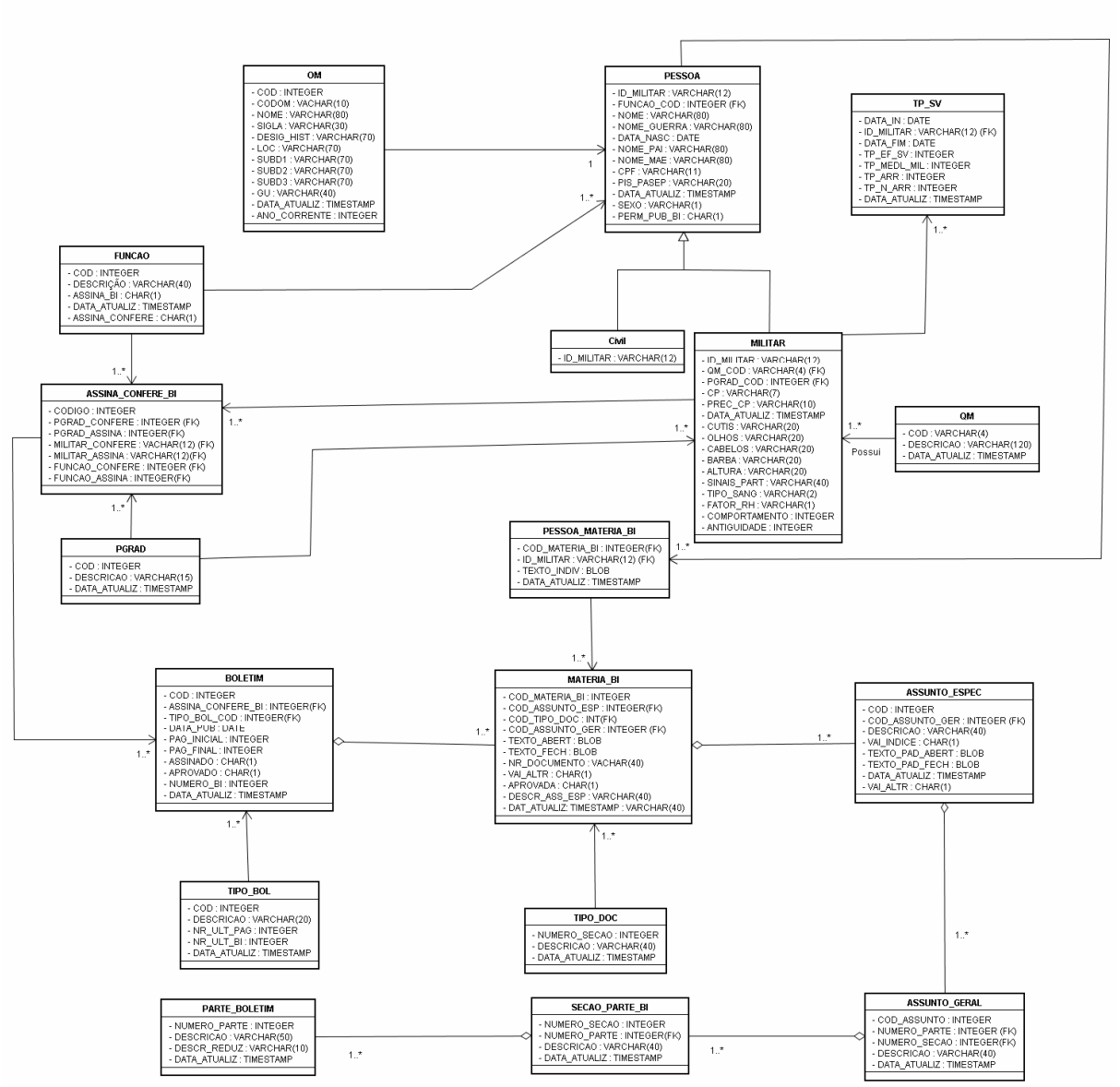
Diagramas de Classes - SIGCRF

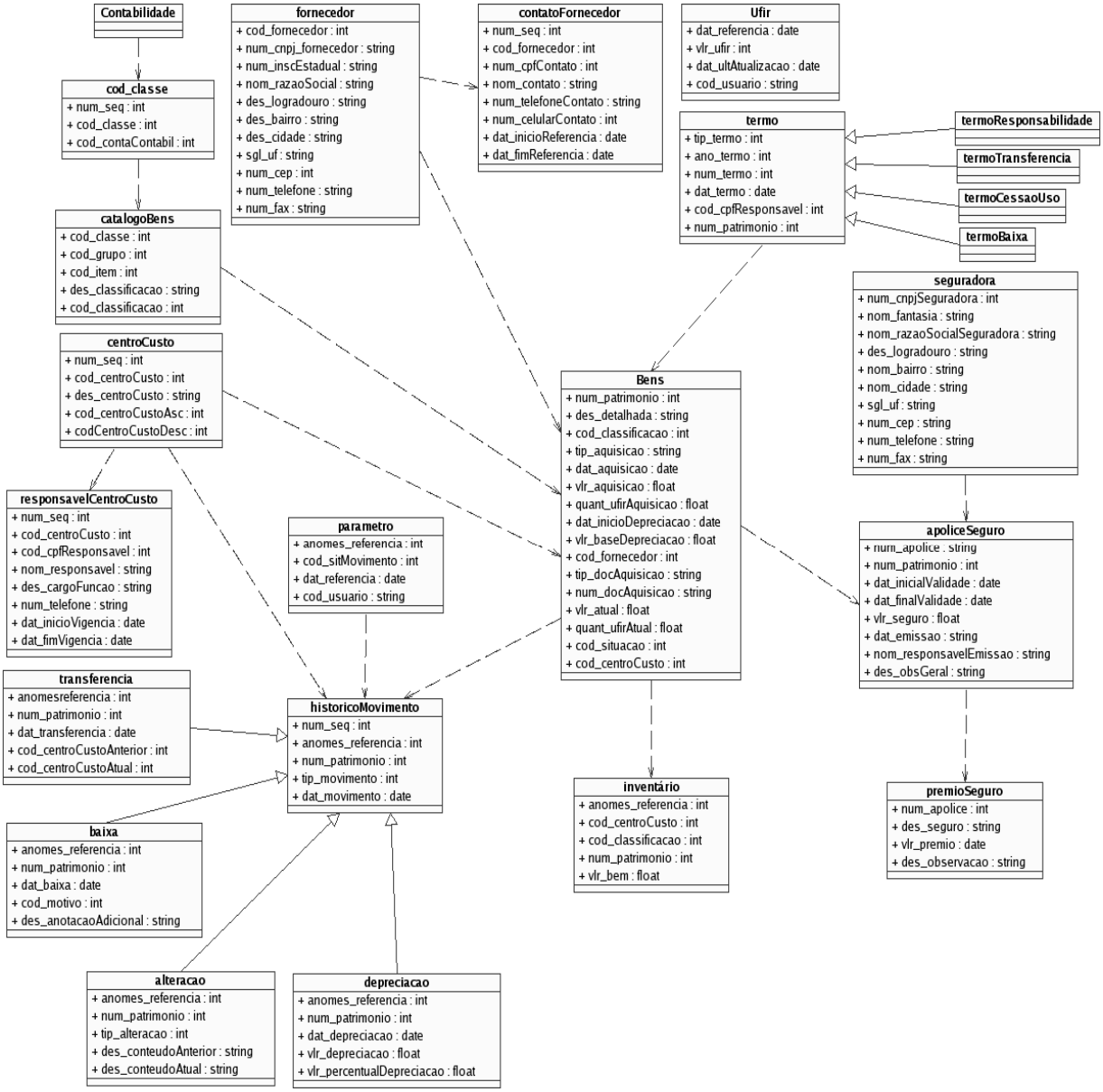


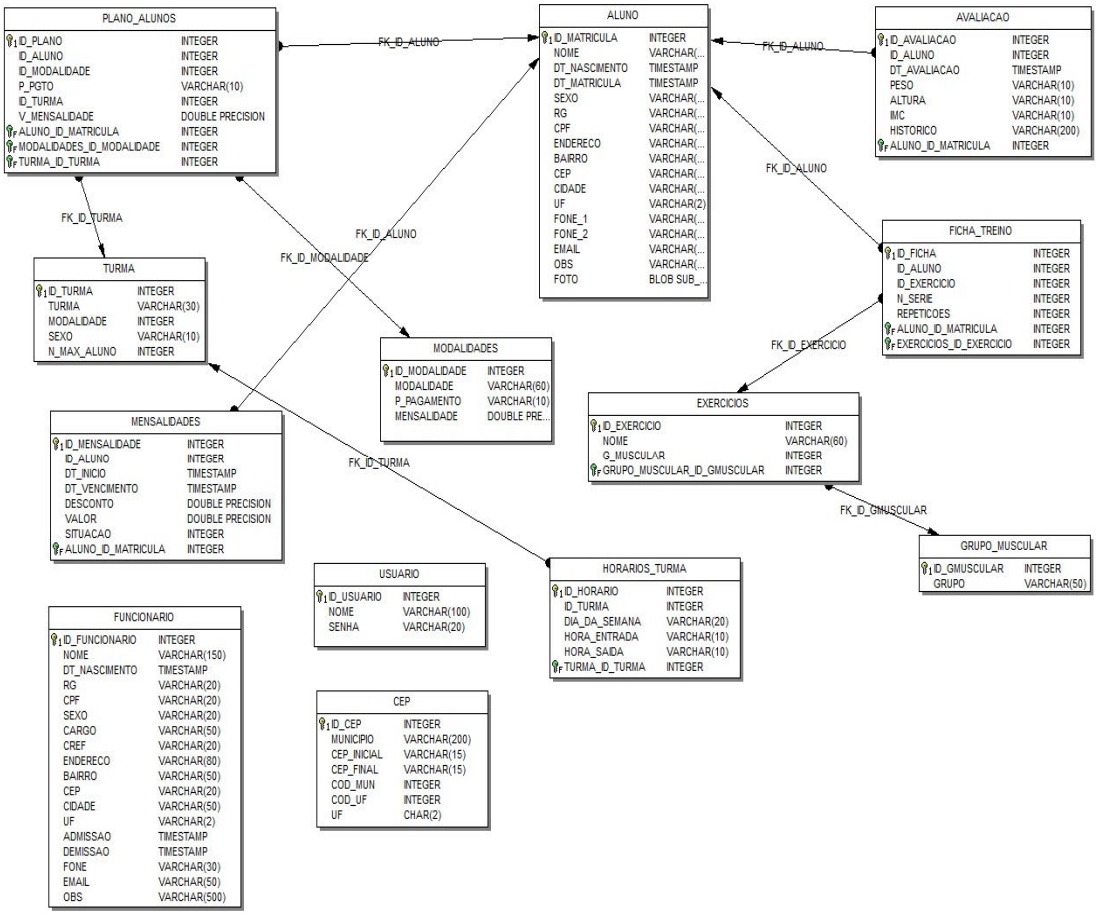

ContatoCliente
Attributes
private Long id
private Cliente cliente
private String tipoContato
private String contato
Operations
public ContatoCliente( )
public Long getId( )
public void setId(Long val)
public String getTipoContato( )
public void setTipoContato(String val)
public String getContato( )
public void setContato(String val)
public Cliente getCliente( )
public void setCliente(Cliente val)
Contato tem um cliente
Cliente tem contatos
Cliente
Attributes
private Long id
private String cnpj
private String ie
private String im
private List<EnderecoCliente> enderecos
private List<ContatoCliente> contatos
Operations
public Cliente( )
public Long getId( )
public void setId(Long val)
public String getCnpj( )
public void setCnpj(String val)
public String getIe( )
public void setIe(String val)
public String getIm( )
public void setIm(String val)
public List<EnderecoCliente> getEnderecos( )
public void setEnderecos(List<EnderecoCliente> val)
public List<ContatoCliente> getContatos( )
public void setContatos(List<ContatoCliente> val)
Cliente tem Endereços
EnderecoCliente
Attributes
private Long id
private Cliente cliente
private String logradouro
private Integer numero
private String complemento
private String bairro
private Long cep
private String cidade
private String uf
Operations
public EnderecoCliente( )
public Long getId( )
public void setId(Long val)
public Cliente getCliente( )
public void setCliente(Cliente val)
public String getLogradouro( )
public void setLogradouro(String val)
public Integer getNumero( )
public void setNumero(Integer val)
public String getComplemento( )
public void setComplemento(String val)
public String getBairro( )
public void setBairro(String val)
public Long getCep( )
public void setCep(Long val)
public String getCidade( )
public void setCidade(String val)
public String getUf( )
public void setUf(String val)
Endereço tem Um Cliente
Pedido tem um cliente
FormaPagamento
Attributes
private Long id
private String nome
Operations
public FormaPagamento( )
public Long getId( )
public void setId(Long val)
public String getNome( )
public void setNome(String val)
Pedido tem forma de Pagamento
ItemPedido
Attributes
private Long id
private Pedido pedido
private int quantidade
private Produto produto
private Double preco
private Double desconto
Operations
public ItemPedido( )
public Long getId( )
public void setId(Long val)
public Pedido getPedido( )
public void setPedido(Pedido val)
public int getQuantidade( )
public void setQuantidade(int val)
public Produto getProduto( )
public void setProduto(Produto val)
public Double getPreco( )
public void setPreco(Double val)
public Double getDesconto( )
public void setDesconto(Double val)
Item tem Produto
Produto
Attributes
private Long id
private String nome
private Double preco
private Long estoque
private Long margemAlarme
Operations
public Produto( )
public Long getId( )
public void setId(Long val)
public String getNome( )
public void setNome(String val)
public Double getPreco( )
public void setPreco(Double val)
public Long getEstoque( )
public void setEstoque(Long val)
public Long getMargemAlarme( )
public void setMargemAlarme(Long val)
Pedido
Attributes
private Long id
private Date dataCriacao
private Cliente cliente
private Usuario vendedor
private FormaPagamento formaPagamento
private List<Historico> historico
private List<ItemPedido> itens
private EnderecoCliente enderecoEntrega
private EnderecoCliente enderecoCobranca
Operations
public Pedido( )
public Long getId( )
public void setId(Long val)
public Date getDataCriacao( )
public void setDataCriacao(Date val)
public Cliente getCliente( )
public void setCliente(Cliente val)
public Usuario getVendedor( )
public void setVendedor(Usuario val)
public FormaPagamento getFormaPagamento( )
public void setFormaPagamento(FormaPagamento val)
public List<Historico> getHistorico( )
public void setHistorico(List<Historico> val)
public List<ItemPedido> getItens( )
public void setItens(List<ItemPedido> val)
public EnderecoCliente getEnderecoEntrega( )
public void setEnderecoEntrega(EnderecoCliente val)
public EnderecoCliente getEnderecoCobranca( )
public void setEnderecoCobranca(EnderecoCliente val)
Pedido tem Endereço de Cobrança
Pedido tem Endereço de Entrega
Pedido tem um vendedor
Pedido tem itens
Item tem Pedido
Historico tem Pedido
Historico
Attributes
private Long id
private Pedido pedido
private Date data
private Usuario usuario
private Status status
private String observacao
Operations
public Historico( )
public Long getId( )
public void setId(Long val)
public Date getData( )
public void setData(Date val)
public Usuario getUsuario( )
public void setUsuario(Usuario val)
public Status getStatus( )
public void setStatus(Status val)
public String getObservacao( )
public void setObservacao(String val)
public Pedido getPedido( )
public void setPedido(Pedido val)
Historico tem Status
Status
Attributes
private Long id
private String nome
private List<Status> proximosStatusPossiveis
Operations
public Status( )
public Long getId( )
public void setId(Long val)
public String getNome( )
public void setNome(String val)
public List<Status> getProximosStatusPossiveis( )
public void setProximosStatusPossiveis(List<Status> val)
Perfil
Attributes
private Long id
private String nome
Operations
public Perfil( )
public Long getId( )
public void setId(Long val)
public String getNome( )
public void setNome(String val)
Usuario tem Perfil
Usuario
Attributes
private Long id
private String nome
private String login
private String senha
private String email
private Perfil perfil
private List<Pedido> pedidos
Operations
public Usuario( )
public Long getId( )
public void setId(Long val)
public String getNome( )
public void setNome(String val)
public String getLogin( )
public void setLogin(String val)
public String getSenha( )
public void setSenha(String val)
public String getEmail( )
public void setEmail(String val)
public Perfil getPerfil( )
public void setPerfil(Perfil val)
public List<Pedido> getPedidos( )
public void setPedidos(List<Pedido> val)
Usuario, quando vendedor, tem pedidos
Perdido tem Historicos

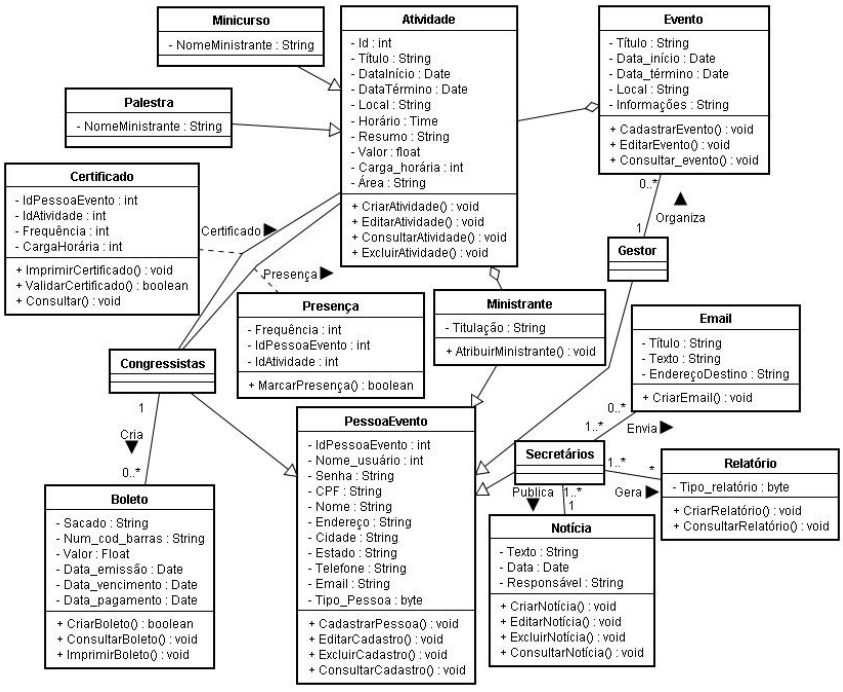

Empresa
- codigo : Integer
- nome : String
- razao : String
- cidade : String
- auditoria : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getCidade() : String
+ setCidade(cidade : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getRazao() : String
+ setRazao(razao : String) : void
+ isValido() : boolean
Banco
- codigo : Integer
- codbanco : Integer
- nome : String
- auditoria : String
- codEmpresa : Integer
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getCodbanco() : Integer
+ setCodbanco(codbanco : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ isValido() : boolean
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
Cidade
- codigo : Integer
- nome : String
- codEmpresa : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ isValido() : boolean
+ getCodEmpresa() : String
+ setCodEmpresa(codEmpresa : String) : void
+ getNome() : String
+ setNome(nome : String) : void
Cliente
- codigo : Integer
- nome : String
- cidade : String
- auditoria : String
- tipo : String
- codEmpresa : Integer
+ isValido() : boolean
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getCidade() : String
+ setCidade(cidade : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
+ getTipo() : String
+ setTipo(tipo : String) : void
Fornecedor
- codigo : Integer
- nome : String
- cidade : String
- auditoria : String
- codEmpresa : Integer
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getCidade() : String
+ setCidade(cidade : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ isValido() : boolean
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
Manutencao
- codigo : Integer
- descricao : String
- codServico : Integer
- codEmpresa : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getDescricao() : String
+ setDescricao(descricao : String) : void
+ getCodEmpresa() : String
+ setCodEmpresa(codEmpresa : String) : void
+ getCodServico() : Integer
+ setCodServico(codServico : Integer) : void
+ isValido() : boolean
Servico
- codigo : Integer
- descricao : String
- troca : String
- codEmpresa : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getDescricao() : String
+ setDescricao(descricao : String) : void
+ isValido() : boolean
+ getCodEmpresa() : String
+ setCodEmpresa(codEmpresa : String) : void
Funcionario
- codigo : Integer
- nome : String
- auditoria : String
- tipo : String
- codEmpresa : Integer
+ isValido() : boolean
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getCidade() : String
+ setCidade(cidade : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
+ getTipo() : String
+ setTipo(tipo : String) : void
Unidade
- codigo : Integer
- descricao : String
- codEmpresa : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getDescricao() : String
+ setDescricao(descricao : String) : void
+ isValido() : boolean
+ getCodEmpresa() : String
+ setCodEmpresa(codEmpresa : String) : void
Produto
- codigo : Integer
- unidade_codigo : String
- fornecedor_codigo : Double
- descricao : String
- auditoria : String
- codEmpresa : String
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getDescricao() : String
+ setDescricao(descricao : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ isValido() : boolean
+ getUnidade_codigo() : String
+ setUnidade_codigo(unidade_codigo : String) : void
+ getFornecedor_codigo() : Double
+ setFornecedor_codigo(fornecedor_codigo : Double) : void
+ getCodEmpresa() : String
+ setCodEmpresa(codEmpresa : String) : void
+ getUnidades() : List<Unidade>
+ setUnidades(unidades : List<Unidade>) : void
ItensVenda
Vendac
- codigo : Integer
- nome : String
- auditoria : String
- tipo : String
- codEmpresa : Integer
+ isValido() : boolean
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNome() : String
+ setNome(nome : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
+ getTipo() : String
+ setTipo(tipo : String) : void
Pagamento
- codigo : Integer
- numdoc : String
- vltotal : Double
- auditoria : String
- codEmpresa : Integer
+ isValido() : boolean
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getNumdoc() : String
+ setNumdoc(numdoc : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
Recebimento
- codigo : Integer
- cliente_codigo : Integer
- numdoc : String
- auditoria : String
- codEmpresa : Integer
+ getCodigo() : Integer
+ setCodigo(codigo : Integer) : void
+ getCliente_codigo() : Integer
+ setCliente_codigo(cliente_codigo : Integer) : void
+ getNumdoc() : String
+ setNumdoc(numdoc : String) : void
+ getAuditoria() : String
+ setAuditoria(auditoria : String) : void
+ getCodEmpresa() : Integer
+ setCodEmpresa(codEmpresa : Integer) : void
+ isValido() : boolean

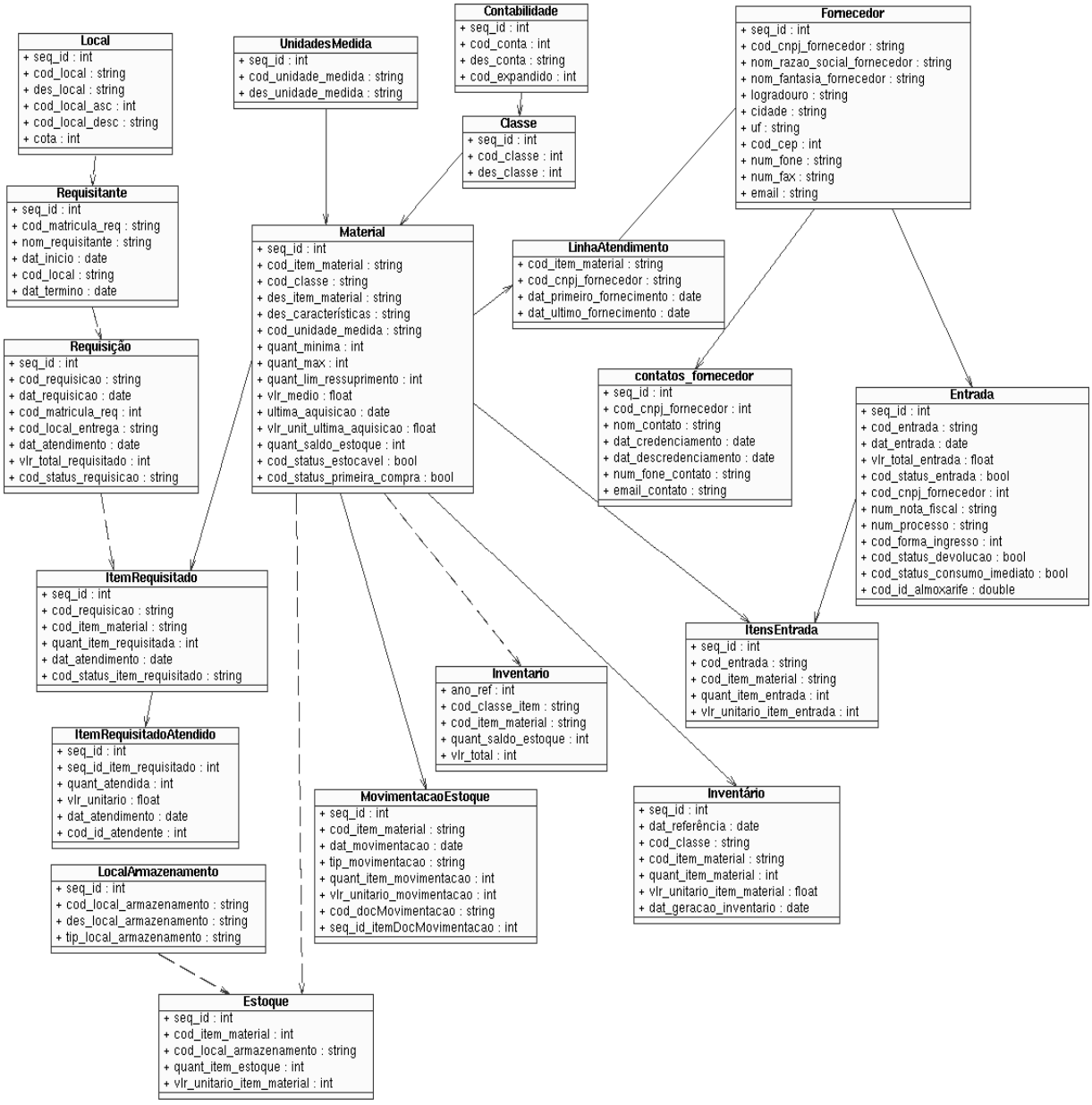

Cadastra
Estado
+idEstado
+descricao
+consulta()
mora
Funcionário
+idtFuncionario
+Matrícula
+Nome
+idt
+orgaoEmissor
+cpf
+endereco
+bairro
+cidade
+cep
+naturalidade
+nacionalidade
+dataNascimento
+sexo
+nomePai
+nomeMae
+estadadoCivil
+salario
+ramal
+importar()
+consultar()
é visitado
Visitante
+idVisitante
+Identidade
+orgaoEmissor
+Nome
+cpf
+obs
+fotografia
+inclui()
+consulta()
+exclui()
+altera()
visita
+idVisita
+dataEntrada
+horaEntrada
+dataSaida
+horaSaida
+obs
+cracha
+inclui()
+altera()
+consultar()
+exclui()
Cargo
+idCargo
+descricao
+importa()
+consulta()
tem
Dependente
+idDependente
+descricao
+consulta()
tem
possui
controleVeiculoVisitante
+idControleVeiculoVisitante
+inclui()
+altera()
+consulta()
+exclui()
dirige
compoe
Secao
+idSecao
+descricao
+importar()
+consultar()
possui
registra
Veículo
+idVeiculo
+placa
+inclui()
+altera()
+exclui()
+consulta()
NivelPermissao
+idNivelPermissao
+descricao
+importar()
+consultar()
+tem
Motorista
+idMotorista
+cnh
+categoria
+local
+validade
+dataEmissao
+Obs
+importa()
+consulta()
compoe
Veículo oficial
+idVeiculoOficial
+chassi
+renavam
+anoFabricacao
+anoModelo
+categoria
+importar()
+consultar()
Telefone
+idTelefone
+descricao
+consulta()
possui
mantem
Cor
+idCor
+descricao
+inclui()
+altera()
+exclui()
+consulta()
tem
TipoVeic
+idTipoVeic
+descricao
+inclui()
+altera()
+consulta()
+exclui()
possui
tem
Modelo
+idModelo
+descricao
+inclui()
+altera()
+exclui()
+consulta()
tem
Marca
+idMarca
+descricao
+importar()
+consultar()
tem
Combustivel
+idCombustivel
+descricao
+consulta()
possui
possui
<<enumeration>>
Tipo
+idTipoTel
+celular
+residencial
+trabalho
+consulta()
controleVeiculoOficial
+idControleVeiculoOficial
+destino
+dataSaida
+horaSaida
+odometroSaida
+odometroEntrada
+dataEntrada
+horaEntrada
+consulta()
+inclui()
+exclui()
+altera()
tem

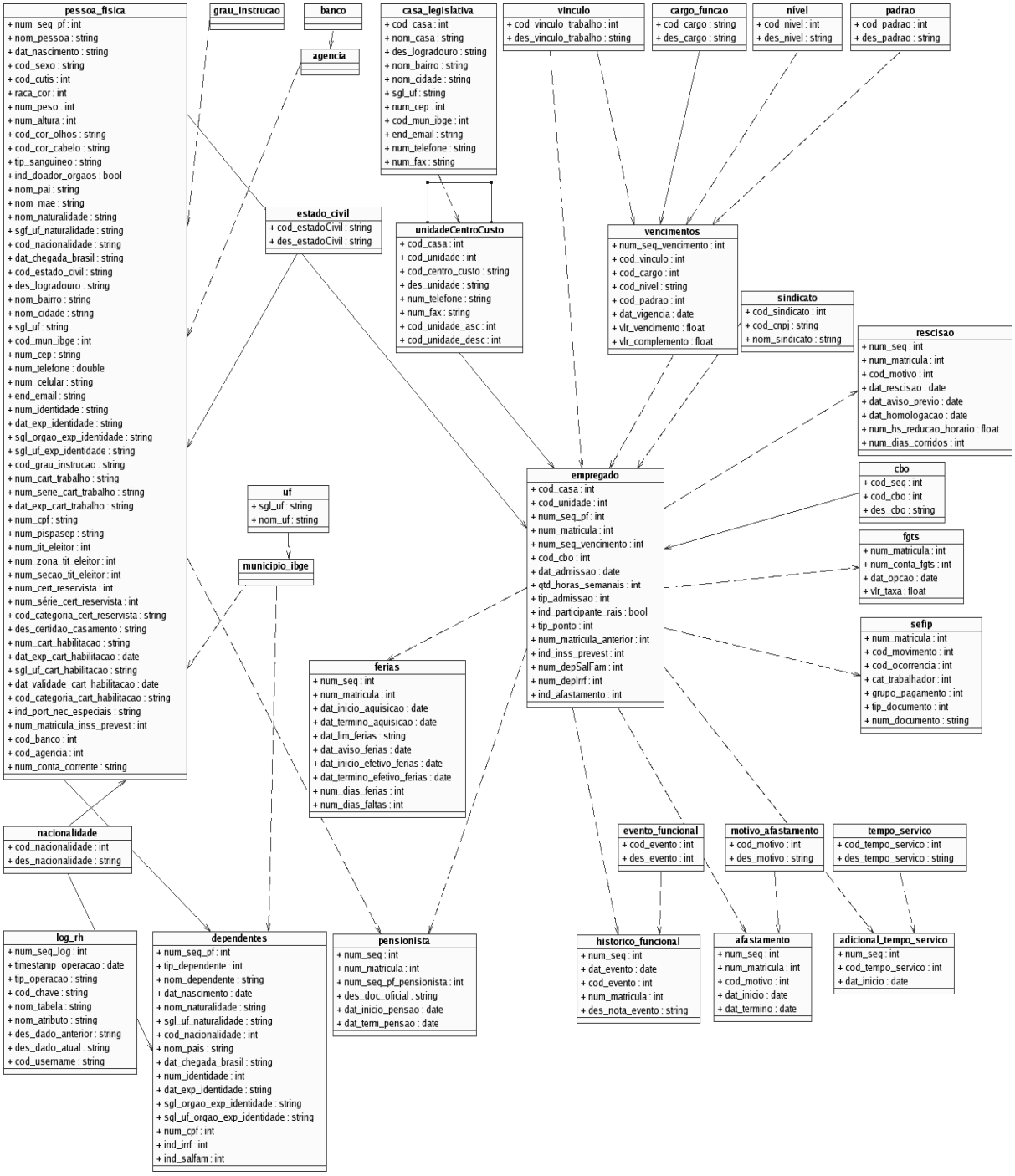

OM
- COD : INTEGER
- CODOM : VACHAR(10)
- NOME : VARCHAR(80)
- SIGLA : VARCHAR(30)
- DESIG_HIST : VARCHAR(70)
- LOC : VARCHAR(70)
- SUBD1 : VARCHAR(70)
- SUBD2 : VARCHAR(70)
- SUBD3 : VARCHAR(70)
- GU : VARCHAR(40)
- DATA_ATUALIZ : TIMESTAMP
- ANO_CORRENTE : INTEGER
PESSOA
- ID_MILITAR : VARCHAR(12)
- FUNCAO_COD : INTEGER (FK)
- NOME : VARCHAR(80)
- NOME_GUERRA : VARCHAR(80)
- DATA_NASC : DATE
- NOME_PAI : VARCHAR(80)
- NOME_MAE : VARCHAR(80)
- CPF : VARCHAR(11)
- PIS_PASEP : VARCHAR(20)
- DATA_ATUALIZ : TIMESTAMP
- SEXO : VARCHAR(1)
- PERM_PUB_BI : CHAR(1)
TP_SV
- DATA_IN : DATE
- ID_MILITAR : VARCHAR(12) (FK)
- DATA_FIM : DATE
- TP_EF_SV : INTEGER
- TP_MEDL_MIL : INTEGER
- TP_ARR : INTEGER
- TP_N_ARR : INTEGER
- DATA_ATUALIZ : TIMESTAMP
FUNCAO
- COD : INTEGER
- DESCRIÇÃO : VARCHAR(40)
- ASSINA_BI : CHAR(1)
- DATA_ATUALIZ : TIMESTAMP
- ASSINA_CONFERE : CHAR(1)
Civil
- ID_MILITAR : VARCHAR(12)
MILITAR
- ID_MILITAR : VARCHAR(12)
- QM_COD : VARCHAR(4) (FK)
- PGRAD_COD : INTEGER (FK)
- CP : VARCHAR(7)
- PREC_CP : VARCHAR(10)
- DATA_ATUALIZ : TIMESTAMP
- CUTIS : VARCHAR(20)
- OLHOS : VARCHAR(20)
- CABELOS : VARCHAR(20)
- BARBA : VARCHAR(20)
- ALTURA : VARCHAR(20)
- SINAIS_PART : VARCHAR(40)
- TIPO_SANG : VARCHAR(2)
- FATOR_RH : VARCHAR(1)
- COMPORTAMENTO : INTEGER
- ANTIGUIDADE : INTEGER
QM
- COD : VARCHAR(4)
- DESCRICAO : VARCHAR(120)
- DATA_ATUALIZ : TIMESTAMP
Possui
ASSINA_CONFERE_BI
- CODIGO : INTEGER
- PGRAD_CONFERE : INTEGER (FK)
- PGRAD_ASSINA : INTEGER(FK)
- MILITAR_CONFERE : VACHAR(12) (FK)
- MILITAR_ASSINA : VARCHAR(12)(FK)
- FUNCAO_CONFERE : INTEGER (FK)
- FUNCAO_ASSINA : INTEGER(FK)
PGRAD
- COD : INTEGER
- DESCRICAO : VARCHAR(15)
- DATA_ATUALIZ : TIMESTAMP
PESSOA_MATERIA_BI
- COD_MATERIA_BI : INTEGER(FK)
- ID_MILITAR : VARCHAR(12) (FK)
- TEXTO_INDIV : BLOB
- DATA_ATUALIZ : TIMESTAMP
BOLETIM
- COD : INTEGER
- ASSINA_CONFERE_BI : INTEGER(FK)
- TIPO_BOL_COD : INTEGER(FK)
- DATA_PUB : DATE
- PAG_INICIAL : INTEGER
- PAG_FINAL : INTEGER
- ASSINADO : CHAR(1)
- APROVADO : CHAR(1)
- NUMERO_BI : INTEGER
- DATA_ATUALIZ : TIMESTAMP
MATERIA_BI
- COD_MATERIA_BI : INTEGER
- COD_ASSUNTO_ESP : INTEGER(FK)
- COD_TIPO_DOC : INT(FK)
- COD_ASSUNTO_GER : INTEGER (FK)
- TEXTO_ABERT : BLOB
- TEXTO_FECH : BLOB
- NR_DOCUMENTO : VACHAR(40)
- VAI_ALTR : CHAR(1)
- APROVADA : CHAR(1)
- DESCR_ASS_ESP : VARCHAR(40)
- DAT_ATUALIZ: TIMESTAMP : VARCHAR(40)
ASSUNTO_ESPEC
- COD : INTEGER
- COD_ASSUNTO_GER : INTEGER (FK)
- DESCRICAO : VARCHAR(40)
- VAI_INDICE : CHAR(1)
- TEXTO_PAD_ABERT : BLOB
- TEXTO_PAD_FECH : BLOB
- DATA_ATUALIZ : TIMESTAMP
- VAI_ALTR : CHAR(1)
TIPO_BOL
- COD : INTEGER
- DESCRICAO : VARCHAR(20)
- NR_ULT_PAG : INTEGER
- NR_ULT_BI : INTEGER
- DATA_ATUALIZ : TIMESTAMP
TIPO_DOC
- NUMERO_SECAO : INTEGER
- DESCRICAO : VARCHAR(40)
- DATA_ATUALIZ : TIMESTAMP
PARTE_BOLETIM
- NUMERO_PARTE : INTEGER
- DESCRICAO : VARCHAR(50)
- DESCR_REDUZ : VARCHAR(10)
- DATA_ATUALIZ : TIMESTAMP
SECAO_PARTE_BI
- NUMERO_SECAO : INTEGER
- NUMERO_PARTE : INTEGER(FK)
- DESCRICAO : VARCHAR(40)
- DATA_ATUALIZ : TIMESTAMP
ASSUNTO_GERAL
- COD_ASSUNTO : INTEGER
- NUMERO_PARTE : INTEGER (FK)
- NUMERO_SECAO : INTEGER(FK)
- DESCRICAO : VARCHAR(40)
- DATA_ATUALIZ : TIMESTAMP

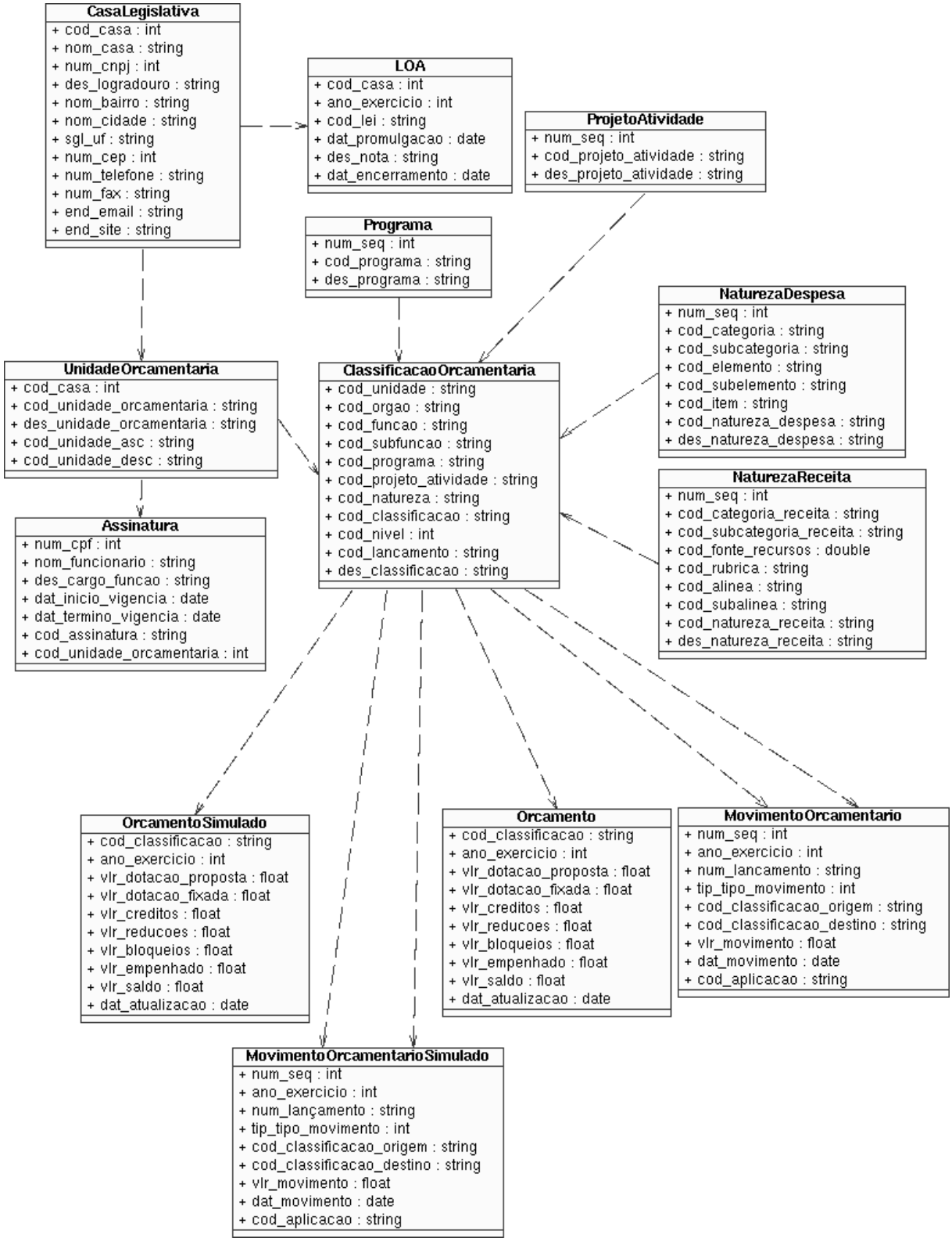

Contabilidade
cod_classe
+ num_seq : int
+ cod_classe : int
+ cod_contaContabil : int
catalogoBens
+ cod_classe : int
+ cod_grupo : int
+ cod_item : int
+ des_classificacao : string
+ cod_classificacao : int
centroCusto
+ num_seq : int
+ cod_centroCusto : int
+ des_centroCusto : string
+ cod_centroCustoAsc : int
+ codCentroCustoDesc : int
responsavelCentroCusto
+ num_seq : int
+ cod_centroCusto : int
+ cod_cpfResponsavel : int
+ nom_responsavel : string
+ des_cargoFuncao : string
+ num_telefone : string
+ dat_inicioVigencia : date
+ dat_fimVigencia : date
transferencia
+ anomesreferencia : int
+ num_patrimonio : int
+ dat_transferencia : date
+ cod_centroCustoAnterior : int
+ cod_centroCustoAtual : int
baixa
+ anomes_referencia : int
+ num_patrimonio : int
+ dat_baixa : date
+ cod_motivo : int
+ des_anotacaoAdicional : string
alteracao
+ anomes_referencia : int
+ num_patrimonio : int
+ tip_alteracao : int
+ des_conteudoAnterior : string
+ des_conteudoAtual : string
fornecedor
+ cod_fornecedor : int
+ num_cnpj_fornecedor : string
+ num_inscEstadual : string
+ nom_razaoSocial : string
+ des_logradouro : string
+ des_bairro : string
+ des_cidade : string
+ sgl_uf : string
+ num_cep : int
+ num_telefone : string
+ num_fax : string
contatoFornecedor
+ num_seq : int
+ cod_fornecedor : int
+ num_cpfContato : int
+ nom_contato : string
+ num_telefoneContato : string
+ num_celularContato : int
+ dat_inicioReferencia : date
+ dat_fimReferencia : date
parametro
+ anomes_referencia : int
+ cod_sitMovimento : int
+ dat_referencia : date
+ cod_usuario : string
historicoMovimento
+ num_seq : int
+ anomes_referencia : int
+ num_patrimonio : int
+ tip_movimento : int
+ dat_movimento : date
depreciacao
+ anomes_referencia : int
+ num_patrimonio : int
+ dat_depreciacao : date
+ vlr_depreciacao : float
+ vlr_percentualDepreciacao : float
Ufir
+ dat_referencia : date
+ vlr_ufir : int
+ dat_ultAtualizacao : date
+ cod_usuario : string
termo
+ tip_termo : int
+ ano_termo : int
+ num_termo : int
+ dat_termo : date
+ cod_cpfResponsavel : int
+ num_patrimonio : int
termoResponsabilidade
termoTransferencia
termoCessaoUso
termoBaixa
Bens
+ num_patrimonio : int
+ des_detalhada : string
+ cod_classificacao : int
+ tip_aquisicao : string
+ dat_aquisicao : date
+ vlr_aquisicao : float
+ quant_ufirAquisicao : float
+ dat_inicioDepreciacao : date
+ vlr_baseDepreciacao : float
+ cod_fornecedor : int
+ tip_docAquisicao : string
+ num_docAquisicao : string
+ vlr_atual : float
+ quant_ufirAtual : float
+ cod_situacao : int
+ cod_centroCusto : int
inventário
+ anomes_referencia : int
+ cod_centroCusto : int
+ cod_classificacao : int
+ num_patrimonio : int
+ vlr_bem : float
seguradora
+ num_cnpjSeguradora : int
+ nom_fantasia : string
+ nom_razaoSocialSeguradora : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : string
+ num_telefone : string
+ num_fax : string
apoliceSeguro
+ num_apolice : string
+ num_patrimonio : int
+ dat_inicialValidade : date
+ dat_finalValidade : date
+ vlr_seguro : float
+ dat_emissao : string
+ nom_responsavelEmissao : string
+ des_obsGeral : string
premioSeguro
+ num_apolice : int
+ des_seguro : string
+ vlr_premio : date
+ des_observacao : string

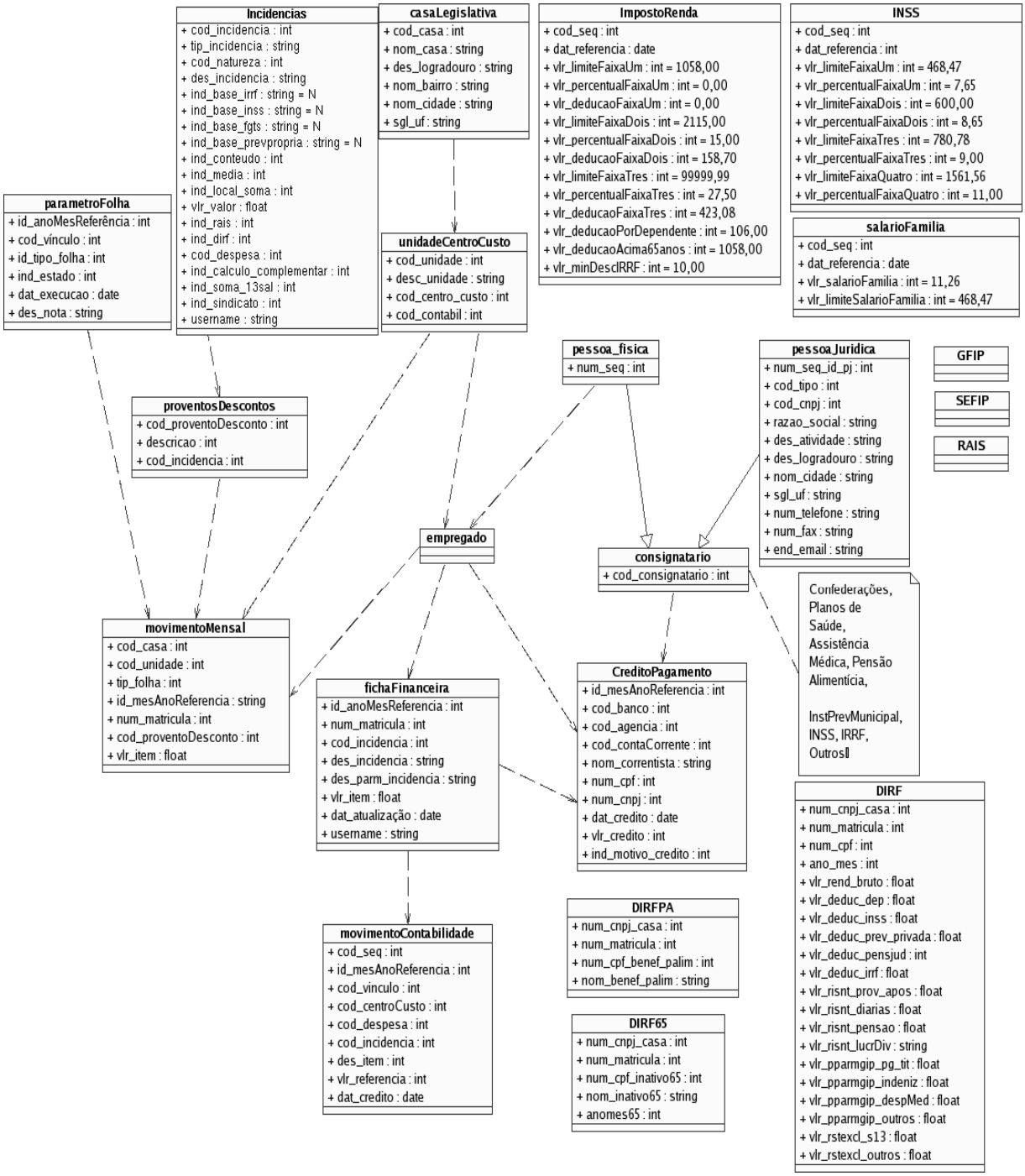

PLANO_ALUNOS
ID_PLANO INTEGER
ID_ALUNO INTEGER
ID_MODALIDADE INTEGER
P_PGTO VARCHAR(10)
ID_TURMA INTEGER
V_MENSALIDADE DOUBLE PRECISION
ALUNO_ID_MATRICULA INTEGER
MODALIDADES_ID_MODALIDADE INTEGER
TURMA_ID_TURMA INTEGER
ALUNO
ID_MATRICULA INTEGER
NOME VARCHAR(...
DT_NASCIMENTO TIMESTAMP
DT_MATRICULA TIMESTAMP
SEXO VARCHAR(...
RG VARCHAR(...
CPF VARCHAR(...
ENDERECO VARCHAR(...
BAIRRO VARCHAR(...
CEP VARCHAR(...
CIDADE VARCHAR(...
UF VARCHAR(2)
FONE_1 VARCHAR(...
FONE_2 VARCHAR(...
EMAIL VARCHAR(...
OBS VARCHAR(...
FOTO BLOB SUB_...
AVALIACAO
ID_AVALIACAO INTEGER
ID_ALUNO INTEGER
DT_AVALIACAO TIMESTAMP
PESO VARCHAR(10)
ALTURA VARCHAR(10)
IMC VARCHAR(10)
HISTORICO VARCHAR(200)
ALUNO_ID_MATRICULA INTEGER
FICHA_TREINO
ID_FICHA INTEGER
ID_ALUNO INTEGER
ID_EXERCICIO INTEGER
N_SERIE INTEGER
REPETICOES INTEGER
ALUNO_ID_MATRICULA INTEGER
EXERCICIOS_ID_EXERCICIO INTEGER
TURMA
ID_TURMA INTEGER
TURMA VARCHAR(30)
MODALIDADE INTEGER
SEXO VARCHAR(10)
N_MAX_ALUNO INTEGER
MODALIDADES
ID_MODALIDADE INTEGER
MODALIDADE VARCHAR(60)
P_PAGAMENTO VARCHAR(10)
MENSALIDADE DOUBLE PRE...
EXERCICIOS
ID_EXERCICIO INTEGER
NOME VARCHAR(60)
G_MUSCULAR INTEGER
GRUPO_MUSCULAR_ID_GMUSCULAR INTEGER
MENSALIDADES
ID_MENSALIDADE INTEGER
ID_ALUNO INTEGER
DT_INICIO TIMESTAMP
DT_VENCIMENTO TIMESTAMP
DESCONTO DOUBLE PRECISION
VALOR DOUBLE PRECISION
SITUACAO INTEGER
ALUNO_ID_MATRICULA INTEGER
GRUPO_MUSCULAR
ID_GMUSCULAR INTEGER
GRUPO VARCHAR(50)
HORARIOS_TURMA
ID_HORARIO INTEGER
ID_TURMA INTEGER
DIA_DA_SEMANA VARCHAR(20)
HORA_ENTRADA VARCHAR(10)
HORA_SAIDA VARCHAR(10)
TURMA_ID_TURMA INTEGER
USUARIO
ID_USUARIO INTEGER
NOME VARCHAR(100)
SENHA VARCHAR(20)
FUNCIONARIO
ID_FUNCIONARIO INTEGER
NOME VARCHAR(150)
DT_NASCIMENTO TIMESTAMP
RG VARCHAR(20)
CPF VARCHAR(20)
SEXO VARCHAR(20)
CARGO VARCHAR(50)
CREF VARCHAR(20)
ENDERECO VARCHAR(80)
BAIRRO VARCHAR(50)
CEP VARCHAR(20)
CIDADE VARCHAR(50)
UF VARCHAR(2)
ADMISSAO TIMESTAMP
DEMISSAO TIMESTAMP
FONE VARCHAR(30)
EMAIL VARCHAR(50)
OBS VARCHAR(500)
CEP
ID_CEP INTEGER
MUNICIPIO VARCHAR(200)
CEP_INICIAL VARCHAR(15)
CEP_FINAL VARCHAR(15)
COD_MUN INTEGER
COD_UF INTEGER
UF CHAR(2)
FK_ID_ALUNO
FK_ID_ALUNO
FK_ID_ALUNO
FK_ID_TURMA
FK_ID_MODALIDADE
FK_ID_EXERCICIO
FK_ID_TURMA
FK_ID_GMUSCULAR

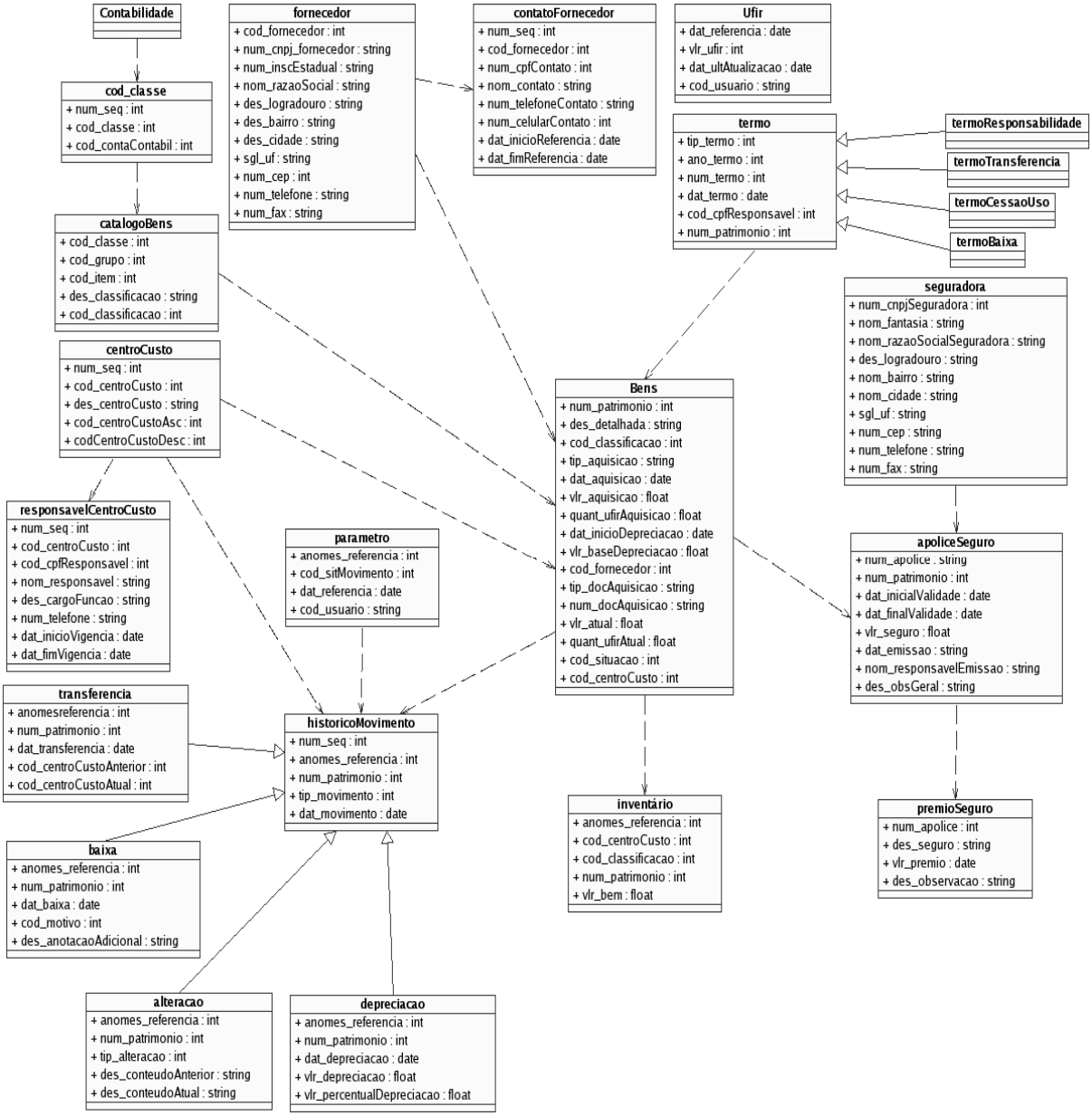

Minicurso
- NomeMinistrante : String
Atividade
- Id : int
- Título : String
- DataInício : Date
- DataTérmino : Date
- Local : String
- Horário : Time
- Resumo : String
- Valor : float
- Carga_horária : int
- Área : String
+ CriarAtividade() : void
+ EditarAtividade() : void
+ ConsultarAtividade() : void
+ ExcluirAtividade() : void
Evento
- Título : String
- Data_início : Date
- Data_término : Date
- Local : String
- Informações : String
+ CadastrarEvento() : void
+ EditarEvento() : void
+ Consultar_evento() : void
Palestra
- NomeMinistrante : String
Certificado
- IdPessoaEvento : int
- IdAtividade : int
- Frequência : int
- CargaHorária : int
+ ImprimirCertificado() : void
+ ValidarCertificado() : boolean
+ Consultar() : void
Certificado
Presença
Organiza
0..*
1
Gestor
Presença
- Frequência : int
- IdPessoaEvento : int
- IdAtividade : int
+ MarcarPresença() : boolean
Ministrante
- Titulação : String
+ AtribuirMinistrante() : void
Email
- Título : String
- Texto : String
- EndereçoDestino : String
+ CriarEmail() : void
Congressistas
1
Cria
0..*
PessoaEvento
- IdPessoaEvento : int
- Nome_usuário : int
- Senha : String
- CPF : String
- Nome : String
- Endereço : String
- Cidade : String
- Estado : String
- Telefone : String
- Email : String
- Tipo_Pessoa : byte
+ CadastrarPessoa() : void
+ EditarCadastro() : void
+ ExcluirCadastro() : void
+ ConsultarCadastro() : void
0..*
1..*
Envia
Secretários
1..*
*
Gera
Relatório
- Tipo_relatório : byte
+ CriarRelatório() : void
+ ConsultarRelatório() : void
Publica
1..*
1
Notícia
- Texto : String
- Data : Date
- Responsável : String
+ CriarNotícia() : void
+ EditarNotícia() : void
+ ExcluirNotícia() : void
+ ConsultarNotícia() : void
Boleto
- Sacado : String
- Num_cod_barras : String
- Valor : Float
- Data_emissão : Date
- Data_vencimento : Date
- Data_pagamento : Date
+ CriarBoleto() : boolean
+ ConsultarBoleto() : void
+ ImprimirBoleto() : void

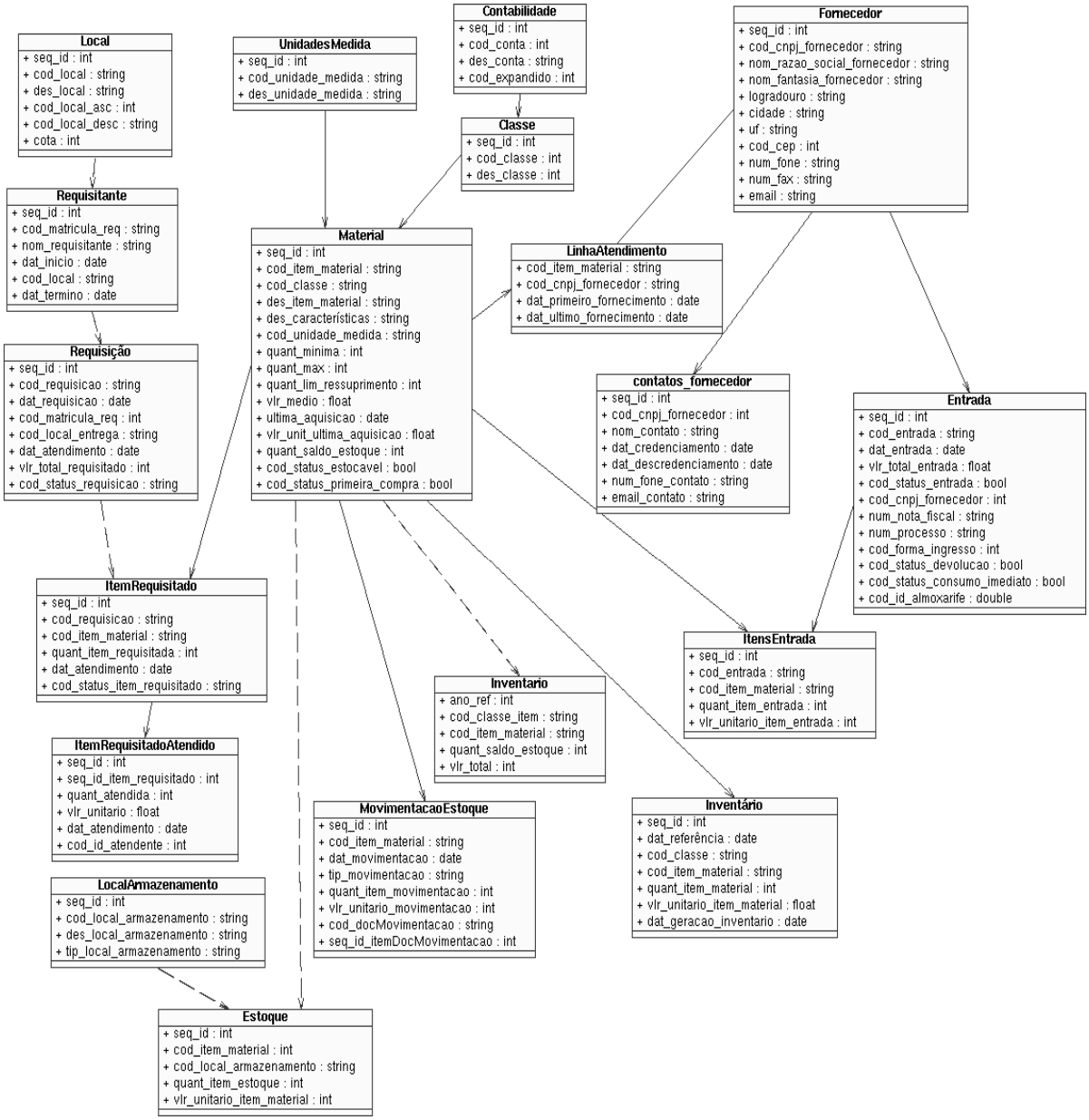

Local
+ seq_id : int
+ cod_local : string
+ des_local : string
+ cod_local_asc : int
+ cod_local_desc : string
+ cota : int
UnidadesMedida
+ seq_id : int
+ cod_unidade_medida : string
+ des_unidade_medida : string
Contabilidade
+ seq_id : int
+ cod_conta : int
+ des_conta : string
+ cod_expandido : int
Fornecedor
+ seq_id : int
+ cod_cnpj_fornecedor : string
+ nom_razao_social_fornecedor : string
+ nom_fantasia_fornecedor : string
+ logradouro : string
+ cidade : string
+ uf : string
+ cod_cep : int
+ num_fone : string
+ num_fax : string
+ email : string
Classe
+ seq_id : int
+ cod_classe : int
+ des_classe : int
Requisitante
+ seq_id : int
+ cod_matricula_req : string
+ nom_requisitante : string
+ dat_inicio : date
+ cod_local : string
+ dat_termino : date
Material
+ seq_id : int
+ cod_item_material : string
+ cod_classe : string
+ des_item_material : string
+ des_características : string
+ cod_unidade_medida : string
+ quant_minima : int
+ quant_max : int
+ quant_lim_ressuprimento : int
+ vlr_medio : float
+ ultima_aquisicao : date
+ vlr_unit_ultima_aquisicao : float
+ quant_saldo_estoque : int
+ cod_status_estocavel : bool
+ cod_status_primeira_compra : bool
LinhaAtendimento
+ cod_item_material : string
+ cod_cnpj_fornecedor : string
+ dat_primeiro_fornecimento : date
+ dat_ultimo_fornecimento : date
Requisição
+ seq_id : int
+ cod_requisicao : string
+ dat_requisicao : date
+ cod_matricula_req : int
+ cod_local_entrega : string
+ dat_atendimento : date
+ vlr_total_requisitado : int
+ cod_status_requisicao : string
contatos_fornecedor
+ seq_id : int
+ cod_cnpj_fornecedor : int
+ nom_contato : string
+ dat_credenciamento : date
+ dat_descredenciamento : date
+ num_fone_contato : string
+ email_contato : string
Entrada
+ seq_id : int
+ cod_entrada : string
+ dat_entrada : date
+ vlr_total_entrada : float
+ cod_status_entrada : bool
+ cod_cnpj_fornecedor : int
+ num_nota_fiscal : string
+ num_processo : string
+ cod_forma_ingresso : int
+ cod_status_devolucao : bool
+ cod_status_consumo_imediato : bool
+ cod_id_almoxarife : double
ItemRequisitado
+ seq_id : int
+ cod_requisicao : string
+ cod_item_material : string
+ quant_item_requisitada : int
+ dat_atendimento : date
+ cod_status_item_requisitado : string
ItensEntrada
+ seq_id : int
+ cod_entrada : string
+ cod_item_material : string
+ quant_item_entrada : int
+ vlr_unitario_item_entrada : int
Inventario
+ ano_ref : int
+ cod_classe_item : string
+ cod_item_material : string
+ quant_saldo_estoque : int
+ vlr_total : int
ItemRequisitadoAtendido
+ seq_id : int
+ seq_id_item_requisitado : int
+ quant_atendida : int
+ vlr_unitario : float
+ dat_atendimento : date
+ cod_id_atendente : int
MovimentacaoEstoque
+ seq_id : int
+ cod_item_material : string
+ dat_movimentacao : date
+ tip_movimentacao : string
+ quant_item_movimentacao : int
+ vlr_unitario_movimentacao : int
+ cod_docMovimentacao : string
+ seq_id_itemDocMovimentacao : int
Inventário
+ seq_id : int
+ dat_referência : date
+ cod_classe : string
+ cod_item_material : string
+ quant_item_material : int
+ vlr_unitario_item_material : float
+ dat_geracao_inventario : date
LocalArmazenamento
+ seq_id : int
+ cod_local_armazenamento : string
+ des_local_armazenamento : string
+ tip_local_armazenamento : string
Estoque
+ seq_id : int
+ cod_item_material : int
+ cod_local_armazenamento : string
+ quant_item_estoque : int
+ vlr_unitario_item_material : int

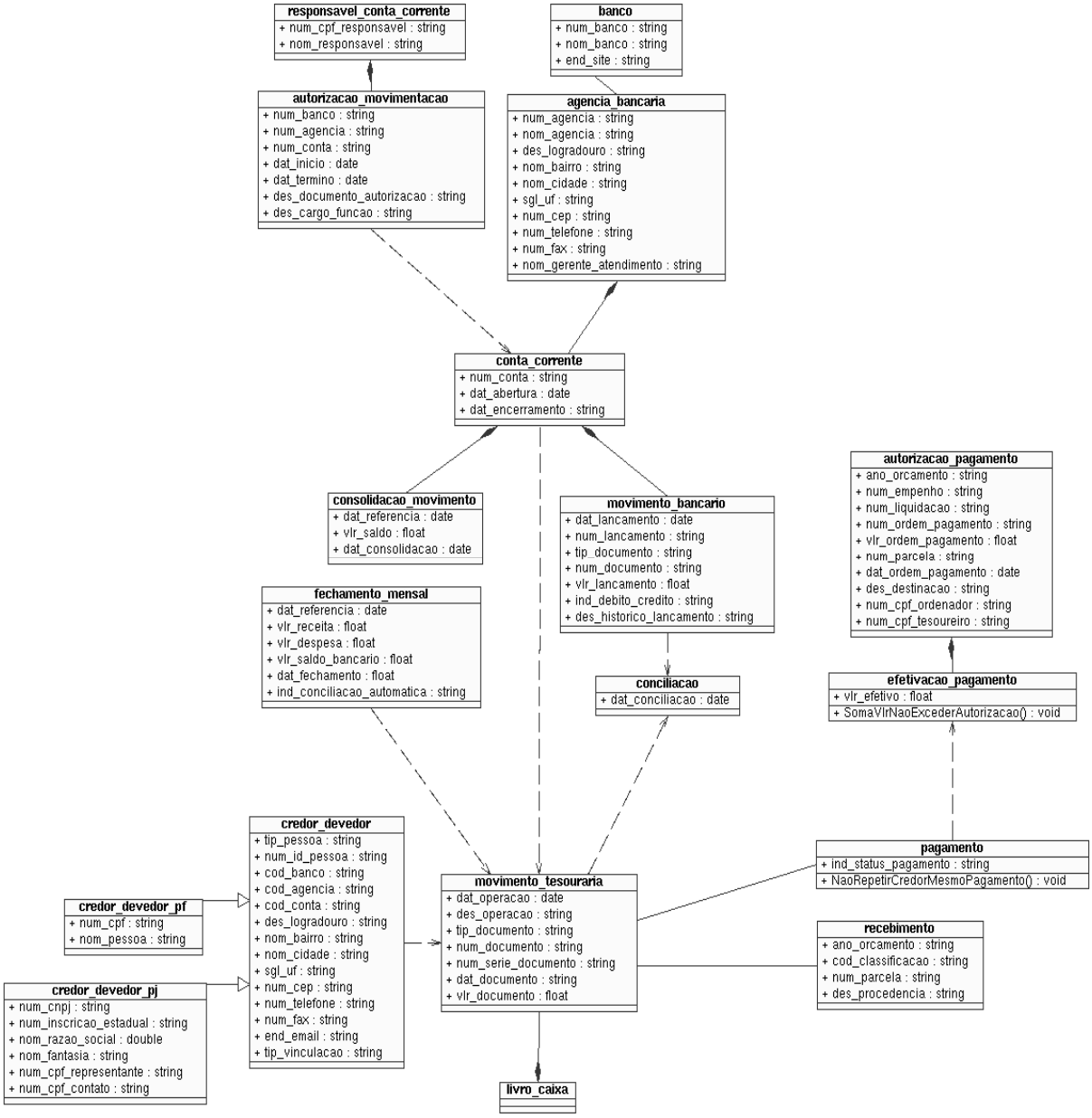

pessoa_fisica
+ num_seq_pf : int
+ nom_pessoa : string
+ dat_nascimento : string
+ cod_sexo : string
+ cod_cutis : int
+ raca_cor : int
+ num_peso : int
+ num_altura : int
+ cod_cor_olhos : string
+ cod_cor_cabelo : string
+ tip_sanguineo : string
+ ind_doador_orgaos : bool
+ nom_pai : string
+ nom_mae : string
+ nom_naturalidade : string
+ sgl_uf_naturalidade : string
+ cod_nacionalidade : string
+ dat_chegada_brasil : string
+ cod_estado_civil : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ cod_mun_ibge : int
+ num_cep : string
+ num_telefone : double
+ num_celular : string
+ end_email : string
+ num_identidade : string
+ dat_exp_identidade : string
+ sgl_orgao_exp_identidade : string
+ sgl_uf_exp_identidade : string
+ cod_grau_instrucao : string
+ num_cart_trabalho : string
+ num_serie_cart_trabalho : string
+ dat_exp_cart_trabalho : string
+ num_cpf : string
+ num_pispasep : string
+ num_tit_eleitor : int
+ num_zona_tit_eleitor : int
+ num_secao_tit_eleitor : int
+ num_cert_reservista : int
+ num_série_cert_reservista : int
+ cod_categoria_cert_reservista : string
+ des_certidao_casamento : string
+ num_cart_habilitacao : string
+ dat_exp_cart_habilitacao : date
+ sgl_uf_cart_habilitacao : string
+ dat_validade_cart_habilitacao : date
+ cod_categoria_cart_habilitacao : string
+ ind_port_nec_especiais : string
+ num_matricula_inss_prevest : int
+ cod_banco : int
+ cod_agencia : int
+ num_conta_corrente : string
grau_instrucao
banco
agencia
casa_legislativa
+ cod_casa : int
+ nom_casa : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : int
+ cod_mun_ibge : int
+ end_email : string
+ num_telefone : string
+ num_fax : string
vinculo
+ cod_vinculo_trabalho : int
+ des_vinculo_trabalho : string
cargo_funcao
+ cod_cargo : string
+ des_cargo : string
nivel
+ cod_nivel : int
+ des_nivel : string
padrao
+ cod_padrao : int
+ des_padrao : string
estado_civil
+ cod_estadoCivil : string
+ des_estadoCivil : string
unidadeCentroCusto
+ cod_casa : int
+ cod_unidade : int
+ cod_centro_custo : string
+ des_unidade : string
+ num_telefone : string
+ num_fax : string
+ cod_unidade_asc : int
+ cod_unidade_desc : int
vencimentos
+ num_seq_vencimento : int
+ cod_vinculo : int
+ cod_cargo : int
+ cod_nivel : string
+ cod_padrao : int
+ dat_vigencia : date
+ vlr_vencimento : float
+ vlr_complemento : float
sindicato
+ cod_sindicato : int
+ cod_cnpj : string
+ nom_sindicato : string
rescisao
+ num_seq : int
+ num_matricula : int
+ cod_motivo : int
+ dat_rescisao : date
+ dat_aviso_previo : date
+ dat_homologacao : date
+ num_hs_reducao_horario : float
+ num_dias_corridos : int
empregado
+ cod_casa : int
+ cod_unidade : int
+ num_seq_pf : int
+ num_matricula : int
+ num_seq_vencimento : int
+ cod_cbo : int
+ dat_admissao : date
+ qtd_horas_semanais : int
+ tip_admissao : int
+ ind_participante_rais : bool
+ tip_ponto : int
+ num_matricula_anterior : int
+ ind_inss_prevest : int
+ num_depSalFam : int
+ num_deplrrf : int
+ ind_afastamento : int
cbo
+ cod_seq : int
+ cod_cbo : int
+ des_cbo : string
fgts
+ num_matricula : int
+ num_conta_fgts : int
+ dat_opcao : date
+ vlr_taxa : float
sefip
+ num_matricula : int
+ cod_movimento : int
+ cod_ocorrencia : int
+ cat_trabalhador : int
+ grupo_pagamento : int
+ tip_documento : int
+ num_documento : string
uf
+ sgl_uf : string
+ nom_uf : string
municipio_ibge
ferias
+ num_seq : int
+ num_matricula : int
+ dat_inicio_aquisicao : date
+ dat_termino_aquisicao : date
+ dat_lim_ferias : string
+ dat_aviso_ferias : date
+ dat_inicio_efetivo_ferias : date
+ dat_termino_efetivo_ferias : date
+ num_dias_ferias : int
+ num_dias_faltas : int
nacionalidade
+ cod_nacionalidade : int
+ des_nacionalidade : string
evento_funcional
+ cod_evento : int
+ des_evento : int
motivo_afastamento
+ cod_motivo : int
+ des_motivo : string
tempo_servico
+ cod_tempo_servico : int
+ des_tempo_servico : string
log_rh
+ num_seq_log : int
+ timestamp_operacao : date
+ tip_operacao : string
+ cod_chave : string
+ nom_tabela : string
+ nom_atributo : string
+ des_dado_anterior : string
+ des_dado_atual : string
+ cod_username : string
dependentes
+ num_seq_pf : int
+ tip_dependente : int
+ nom_dependente : string
+ dat_nascimento : date
+ nom_naturalidade : string
+ sgl_uf_naturalidade : string
+ cod_nacionalidade : int
+ nom_pais : string
+ dat_chegada_brasil : string
+ num_identidade : int
+ dat_exp_identidade : string
+ sgl_orgao_exp_identidade : string
+ sgl_uf_orgao_exp_identidade : string
+ num_cpf : int
+ ind_irrf : int
+ ind_salfam : int
pensionista
+ num_seq : int
+ num_matricula : int
+ num_seq_pf_pensionista : int
+ des_doc_oficial : string
+ dat_inicio_pensao : date
+ dat_term_pensao : date
historico_funcional
+ num_seq : int
+ dat_evento : date
+ cod_evento : int
+ num_matricula : int
+ des_nota_evento : string
afastamento
+ num_seq : int
+ num_matricula : int
+ cod_motivo : int
+ dat_inicio : date
+ dat_termino : date
adicional_tempo_servico
+ num_seq : int
+ cod_tempo_servico : int
+ dat_inicio : date


CasaLegislativa
+ cod_casa : int
+ nom_casa : string
+ num_cnpj : int
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : int
+ num_telefone : string
+ num_fax : string
+ end_email : string
+ end_site : string
LOA
+ cod_casa : int
+ ano_exercicio : int
+ cod_lei : string
+ dat_promulgacao : date
+ des_nota : string
+ dat_encerramento : date
ProjetoAtividade
+ num_seq : int
+ cod_projeto_atividade : string
+ des_projeto_atividade : string
Programa
+ num_seq : int
+ cod_programa : string
+ des_programa : string
NaturezaDespesa
+ num_seq : int
+ cod_categoria : string
+ cod_subcategoria : string
+ cod_elemento : string
+ cod_subelemento : string
+ cod_item : string
+ cod_natureza_despesa : string
+ des_natureza_despesa : string
UnidadeOrcamentaria
+ cod_casa : int
+ cod_unidade_orcamentaria : string
+ des_unidade_orcamentaria : string
+ cod_unidade_asc : string
+ cod_unidade_desc : string
ClassificacaoOrcamentaria
+ cod_unidade : string
+ cod_orgao : string
+ cod_funcao : string
+ cod_subfuncao : string
+ cod_programa : string
+ cod_projeto_atividade : string
+ cod_natureza : string
+ cod_classificacao : string
+ cod_nivel : int
+ cod_lancamento : string
+ des_classificacao : string
NaturezaReceita
+ num_seq : int
+ cod_categoria_receita : string
+ cod_subcategoria_receita : string
+ cod_fonte_recursos : double
+ cod_rubrica : string
+ cod_alinea : string
+ cod_subalinea : string
+ cod_natureza_receita : string
+ des_natureza_receita : string
Assinatura
+ num_cpf : int
+ nom_funcionario : string
+ des_cargo_funcao : string
+ dat_inicio_vigencia : date
+ dat_termino_vigencia : date
+ cod_assinatura : string
+ cod_unidade_orcamentaria : int
OrcamentoSimulado
+ cod_classificacao : string
+ ano_exercicio : int
+ vlr_dotacao_proposta : float
+ vlr_dotacao_fixada : float
+ vlr_creditos : float
+ vlr_reducoes : float
+ vlr_bloqueios : float
+ vlr_empenhado : float
+ vlr_saldo : float
+ dat_atualizacao : date
Orcamento
+ cod_classificacao : string
+ ano_exercicio : int
+ vlr_dotacao_proposta : float
+ vlr_dotacao_fixada : float
+ vlr_creditos : float
+ vlr_reducoes : float
+ vlr_bloqueios : float
+ vlr_empenhado : float
+ vlr_saldo : float
+ dat_atualizacao : date
MovimentoOrcamentario
+ num_seq : int
+ ano_exercicio : int
+ num_lancamento : string
+ tip_tipo_movimento : int
+ cod_classificacao_origem : string
+ cod_classificacao_destino : string
+ vlr_movimento : float
+ dat_movimento : date
+ cod_aplicacao : string
MovimentoOrcamentarioSimulado
+ num_seq : int
+ ano_exercicio : int
+ num_lançamento : string
+ tip_tipo_movimento : int
+ cod_classificacao_origem : string
+ cod_classificacao_destino : string
+ vlr_movimento : float
+ dat_movimento : date
+ cod_aplicacao : string


Incidencias
+ cod_incidencia : int
+ tip_incidencia : string
+ cod_natureza : int
+ des_incidencia : string
+ ind_base_irrf : string = N
+ ind_base_inss : string = N
+ ind_base_fgts : string = N
+ ind_base_prevpropria : string = N
+ ind_conteudo : int
+ ind_media : int
+ ind_local_soma : int
+ vlr_valor : float
+ ind_rais : int
+ ind_dirf : int
+ cod_despesa : int
+ ind_calculo_complementar : int
+ ind_soma_13sal : int
+ ind_sindicato : int
+ username : string
casaLegislativa
+ cod_casa : int
+ nom_casa : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
ImpostoRenda
+ cod_seq : int
+ dat_referencia : date
+ vlr_limiteFaixaUm : int = 1058,00
+ vlr_percentualFaixaUm : int = 0,00
+ vlr_deducaoFaixaUm : int = 0,00
+ vlr_limiteFaixaDois : int = 2115,00
+ vlr_percentualFaixaDois : int = 15,00
+ vlr_deducaoFaixaDois : int = 158,70
+ vlr_limiteFaixaTres : int = 99999,99
+ vlr_percentualFaixaTres : int = 27,50
+ vlr_deducaoFaixaTres : int = 423,08
+ vlr_deducaoPorDependente : int = 106,00
+ vlr_deducaoAcima65anos : int = 1058,00
+ vlr_minDesclRRF : int = 10,00
INSS
+ cod_seq : int
+ dat_referencia : int
+ vlr_limiteFaixaUm : int = 468,47
+ vlr_percentualFaixaUm : int = 7,65
+ vlr_limiteFaixaDois : int = 600,00
+ vlr_percentualFaixaDois : int = 8,65
+ vlr_limiteFaixaTres : int = 780,78
+ vlr_percentualFaixaTres : int = 9,00
+ vlr_limiteFaixaQuatro : int = 1561,56
+ vlr_percentualFaixaQuatro : int = 11,00
parametroFolha
+ id_anoMesReferência : int
+ cod_vínculo : int
+ id_tipo_folha : int
+ ind_estado : int
+ dat_execucao : date
+ des_nota : string
unidadeCentroCusto
+ cod_unidade : int
+ desc_unidade : string
+ cod_centro_custo : int
+ cod_contabil : int
salarioFamilia
+ cod_seq : int
+ dat_referencia : date
+ vlr_salarioFamilia : int = 11,26
+ vlr_limiteSalarioFamilia : int = 468,47
pessoa_fisica
+ num_seq : int
pessoaJuridica
+ num_seq_id_pj : int
+ cod_tipo : int
+ cod_cnpj : int
+ razao_social : string
+ des_atividade : string
+ des_logradouro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_telefone : string
+ num_fax : string
+ end_email : string
GFIP
SEFIP
RAIS
proventosDescontos
+ cod_proventoDesconto : int
+ descricao : int
+ cod_incidencia : int
empregado
consignatario
+ cod_consignatario : int
Confederações,
Planos de
Saúde,
Assistência
Médica, Pensão
Alimentícia,
InstPrevMunicipal,
INSS, IRRF,
Outros
movimentoMensal
+ cod_casa : int
+ cod_unidade : int
+ tip_folha : int
+ id_mesAnoReferencia : string
+ num_matricula : int
+ cod_proventoDesconto : int
+ vlr_item : float
fichaFinanceira
+ id_anoMesReferencia : int
+ num_matricula : int
+ cod_incidencia : int
+ des_incidencia : string
+ des_parm_incidencia : string
+ vlr_item : float
+ dat_atualização : date
+ username : string
CreditoPagamento
+ id_mesAnoReferencia : int
+ cod_banco : int
+ cod_agencia : int
+ cod_contaCorrente : int
+ nom_correntista : string
+ num_cpf : int
+ num_cnpj : int
+ dat_credito : date
+ vlr_credito : int
+ ind_motivo_credito : int
DIRF
+ num_cnpj_casa : int
+ num_matricula : int
+ num_cpf : int
+ ano_mes : int
+ vlr_rend_bruto : float
+ vlr_deduc_dep : float
+ vlr_deduc_inss : float
+ vlr_deduc_prev_privada : float
+ vlr_deduc_pensjud : int
+ vlr_deduc_irrf : float
+ vlr_risnt_prov_apos : float
+ vlr_risnt_diarias : float
+ vlr_risnt_pensao : float
+ vlr_risnt_lucrDiv : string
+ vlr_pparmgip_pg_tit : float
+ vlr_pparmgip_indeniz : float
+ vlr_pparmgip_despMed : float
+ vlr_pparmgip_outros : float
+ vlr_rstexcl_s13 : float
+ vlr_rstexcl_outros : float
movimentoContabilidade
+ cod_seq : int
+ id_mesAnoReferencia : int
+ cod_vinculo : int
+ cod_centroCusto : int
+ cod_despesa : int
+ cod_incidencia : int
+ des_item : int
+ vlr_referencia : int
+ dat_credito : date
DIRFPA
+ num_cnpj_casa : int
+ num_matricula : int
+ num_cpf_benef_palim : int
+ nom_benef_palim : string
DIRF65
+ num_cnpj_casa : int
+ num_matricula : int
+ num_cpf_inativo65 : int
+ nom_inativo65 : string
+ anomes65 : int


Contabilidade
cod_classe
+ num_seq : int
+ cod_classe : int
+ cod_contaContabil : int
fornecedor
+ cod_fornecedor : int
+ num_cnpj_fornecedor : string
+ num_inscEstadual : string
+ nom_razaoSocial : string
+ des_logradouro : string
+ des_bairro : string
+ des_cidade : string
+ sgl_uf : string
+ num_cep : int
+ num_telefone : string
+ num_fax : string
contatoFornecedor
+ num_seq : int
+ cod_fornecedor : int
+ num_cpfContato : int
+ nom_contato : string
+ num_telefoneContato : string
+ num_celularContato : int
+ dat_inicioReferencia : date
+ dat_fimReferencia : date
Ufir
+ dat_referencia : date
+ vlr_ufir : int
+ dat_ultAtualizacao : date
+ cod_usuario : string
termo
+ tip_termo : int
+ ano_termo : int
+ num_termo : int
+ dat_termo : date
+ cod_cpfResponsavel : int
+ num_patrimonio : int
termoResponsabilidade
termoTransferencia
termoCessaoUso
termoBaixa
catalogoBens
+ cod_classe : int
+ cod_grupo : int
+ cod_item : int
+ des_classificacao : string
+ cod_classificacao : int
seguradora
+ num_cnpjSeguradora : int
+ nom_fantasia : string
+ nom_razaoSocialSeguradora : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : string
+ num_telefone : string
+ num_fax : string
centroCusto
+ num_seq : int
+ cod_centroCusto : int
+ des_centroCusto : string
+ cod_centroCustoAsc : int
+ codCentroCustoDesc : int
Bens
+ num_patrimonio : int
+ des_detalhada : string
+ cod_classificacao : int
+ tip_aquisicao : string
+ dat_aquisicao : date
+ vlr_aquisicao : float
+ quant_ufirAquisicao : float
+ dat_inicioDepreciacao : date
+ vlr_baseDepreciacao : float
+ cod_fornecedor : int
+ tip_docAquisicao : string
+ num_docAquisicao : string
+ vlr_atual : float
+ quant_ufirAtual : float
+ cod_situacao : int
+ cod_centroCusto : int
responsavelCentroCusto
+ num_seq : int
+ cod_centroCusto : int
+ cod_cpfResponsavel : int
+ nom_responsavel : string
+ des_cargoFuncao : string
+ num_telefone : string
+ dat_inicioVigencia : date
+ dat_fimVigencia : date
parametro
+ anomes_referencia : int
+ cod_sitMovimento : int
+ dat_referencia : date
+ cod_usuario : string
apoliceSeguro
+ num_apolice : string
+ num_patrimonio : int
+ dat_inicialValidade : date
+ dat_finalValidade : date
+ vlr_seguro : float
+ dat_emissao : string
+ nom_responsavelEmissao : string
+ des_obsGeral : string
transferencia
+ anomesreferencia : int
+ num_patrimonio : int
+ dat_transferencia : date
+ cod_centroCustoAnterior : int
+ cod_centroCustoAtual : int
historicoMovimento
+ num_seq : int
+ anomes_referencia : int
+ num_patrimonio : int
+ tip_movimento : int
+ dat_movimento : date
inventário
+ anomes_referencia : int
+ cod_centroCusto : int
+ cod_classificacao : int
+ num_patrimonio : int
+ vlr_bem : float
premioSeguro
+ num_apolice : int
+ des_seguro : string
+ vlr_premio : date
+ des_observacao : string
baixa
+ anomes_referencia : int
+ num_patrimonio : int
+ dat_baixa : date
+ cod_motivo : int
+ des_anotacaoAdicional : string
alteracao
+ anomes_referencia : int
+ num_patrimonio : int
+ tip_alteracao : int
+ des_conteudoAnterior : string
+ des_conteudoAtual : string
depreciacao
+ anomes_referencia : int
+ num_patrimonio : int
+ dat_depreciacao : date
+ vlr_depreciacao : float
+ vlr_percentualDepreciacao : float


Local
+ seq_id : int
+ cod_local : string
+ des_local : string
+ cod_local_asc : int
+ cod_local_desc : string
+ cota : int
UnidadesMedida
+ seq_id : int
+ cod_unidade_medida : string
+ des_unidade_medida : string
Contabilidade
+ seq_id : int
+ cod_conta : int
+ des_conta : string
+ cod_expandido : int
Fornecedor
+ seq_id : int
+ cod_cnpj_fornecedor : string
+ nom_razao_social_fornecedor : string
+ nom_fantasia_fornecedor : string
+ logradouro : string
+ cidade : string
+ uf : string
+ cod_cep : int
+ num_fone : string
+ num_fax : string
+ email : string
Classe
+ seq_id : int
+ cod_classe : int
+ des_classe : int
Requisitante
+ seq_id : int
+ cod_matricula_req : string
+ nom_requisitante : string
+ dat_inicio : date
+ cod_local : string
+ dat_termino : date
Material
+ seq_id : int
+ cod_item_material : string
+ cod_classe : string
+ des_item_material : string
+ des_características : string
+ cod_unidade_medida : string
+ quant_minima : int
+ quant_max : int
+ quant_lim_ressuprimento : int
+ vlr_medio : float
+ ultima_aquisicao : date
+ vlr_unit_ultima_aquisicao : float
+ quant_saldo_estoque : int
+ cod_status_estocavel : bool
+ cod_status_primeira_compra : bool
LinhaAtendimento
+ cod_item_material : string
+ cod_cnpj_fornecedor : string
+ dat_primeiro_fornecimento : date
+ dat_ultimo_fornecimento : date
Requisição
+ seq_id : int
+ cod_requisicao : string
+ dat_requisicao : date
+ cod_matricula_req : int
+ cod_local_entrega : string
+ dat_atendimento : date
+ vlr_total_requisitado : int
+ cod_status_requisicao : string
contatos_fornecedor
+ seq_id : int
+ cod_cnpj_fornecedor : int
+ nom_contato : string
+ dat_credenciamento : date
+ dat_descredenciamento : date
+ num_fone_contato : string
+ email_contato : string
Entrada
+ seq_id : int
+ cod_entrada : string
+ dat_entrada : date
+ vlr_total_entrada : float
+ cod_status_entrada : bool
+ cod_cnpj_fornecedor : int
+ num_nota_fiscal : string
+ num_processo : string
+ cod_forma_ingresso : int
+ cod_status_devolucao : bool
+ cod_status_consumo_imediato : bool
+ cod_id_almoxarife : double
ItemRequisitado
+ seq_id : int
+ cod_requisicao : string
+ cod_item_material : string
+ quant_item_requisitada : int
+ dat_atendimento : date
+ cod_status_item_requisitado : string
ItensEntrada
+ seq_id : int
+ cod_entrada : string
+ cod_item_material : string
+ quant_item_entrada : int
+ vlr_unitario_item_entrada : int
Inventario
+ ano_ref : int
+ cod_classe_item : string
+ cod_item_material : string
+ quant_saldo_estoque : int
+ vlr_total : int
ItemRequisitadoAtendido
+ seq_id : int
+ seq_id_item_requisitado : int
+ quant_atendida : int
+ vlr_unitario : float
+ dat_atendimento : date
+ cod_id_atendente : int
MovimentacaoEstoque
+ seq_id : int
+ cod_item_material : string
+ dat_movimentacao : date
+ tip_movimentacao : string
+ quant_item_movimentacao : int
+ vlr_unitario_movimentacao : int
+ cod_docMovimentacao : string
+ seq_id_itemDocMovimentacao : int
Inventário
+ seq_id : int
+ dat_referência : date
+ cod_classe : string
+ cod_item_material : string
+ quant_item_material : int
+ vlr_unitario_item_material : float
+ dat_geracao_inventario : date
LocalArmazenamento
+ seq_id : int
+ cod_local_armazenamento : string
+ des_local_armazenamento : string
+ tip_local_armazenamento : string
Estoque
+ seq_id : int
+ cod_item_material : int
+ cod_local_armazenamento : string
+ quant_item_estoque : int
+ vlr_unitario_item_material : int


responsavel_conta_corrente
+ num_cpf_responsavel : string
+ nom_responsavel : string
banco
+ num_banco : string
+ nom_banco : string
+ end_site : string
autorizacao_movimentacao
+ num_banco : string
+ num_agencia : string
+ num_conta : string
+ dat_inicio : date
+ dat_termino : date
+ des_documento_autorizacao : string
+ des_cargo_funcao : string
agencia_bancaria
+ num_agencia : string
+ nom_agencia : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : string
+ num_telefone : string
+ num_fax : string
+ nom_gerente_atendimento : string
conta_corrente
+ num_conta : string
+ dat_abertura : date
+ dat_encerramento : string
autorizacao_pagamento
+ ano_orcamento : string
+ num_empenho : string
+ num_liquidacao : string
+ num_ordem_pagamento : string
+ vlr_ordem_pagamento : float
+ num_parcela : string
+ dat_ordem_pagamento : date
+ des_destinacao : string
+ num_cpf_ordenador : string
+ num_cpf_tesoureiro : string
consolidacao_movimento
+ dat_referencia : date
+ vlr_saldo : float
+ dat_consolidacao : date
movimento_bancario
+ dat_lancamento : date
+ num_lancamento : string
+ tip_documento : string
+ num_documento : string
+ vlr_lancamento : float
+ ind_debito_credito : string
+ des_historico_lancamento : string
fechamento_mensal
+ dat_referencia : date
+ vlr_receita : float
+ vlr_despesa : float
+ vlr_saldo_bancario : float
+ dat_fechamento : float
+ ind_conciliacao_automatica : string
conciliacao
+ dat_conciliacao : date
efetivacao_pagamento
+ vlr_efetivo : float
+ SomaVlrNaoExcederAutorizacao() : void
credor_devedor
+ tip_pessoa : string
+ num_id_pessoa : string
+ cod_banco : string
+ cod_agencia : string
+ cod_conta : string
+ des_logradouro : string
+ nom_bairro : string
+ nom_cidade : string
+ sgl_uf : string
+ num_cep : string
+ num_telefone : string
+ num_fax : string
+ end_email : string
+ tip_vinculacao : string
movimento_tesouraria
+ dat_operacao : date
+ des_operacao : string
+ tip_documento : string
+ num_documento : string
+ num_serie_documento : string
+ dat_documento : string
+ vlr_documento : float
pagamento
+ ind_status_pagamento : string
+ NaoRepetirCredorMesmoPagamento() : void
credor_devedor_pf
+ num_cpf : string
+ nom_pessoa : string
credor_devedor_pj
+ num_cnpj : string
+ num_inscricao_estadual : string
+ nom_razao_social : double
+ nom_fantasia : string
+ num_cpf_representante : string
+ num_cpf_contato : string
recebimento
+ ano_orcamento : string
+ cod_classificacao : string
+ num_parcela : string
+ des_procedencia : string
livro_caixa