3
Referências e Base Técnica

Existem muitos trabalhos relacionados a diferentes usos de diagramas UML, mas poucos são os relacionados à análise das informações extraídas deles. Neste capítulo serão descritos vários trabalhos agrupados pelos seguintes tópicos:

- Detecção de inconsistências extraídas de diagramas UML.
- Transformação de modelos em grafo.
- Algoritmos de agrupamento.
- Obtenção dos *core concepts*.
- Definições de métricas de diagramas UML.
- Ferramentas de análise UML.
- Estado da arte.

Para cada tópico, será esclarecida a relação deste com o trabalho proposto.

3.1.
Detecção de inconsistências extraídas de diagramas UML

Os artigos relacionados com a identificação e conserto de inconsistências de diagramas UML realizam, até um determinado ponto, uma pesquisa semelhante à proposta. Lê-se o diagrama UML e, a partir deste ponto, são detectadas possíveis inconsistências dos diagramas pela análise sintática deste grafo.

A diferença destes artigos para o estudo proposto é que a detecção de inconsistências do diagrama de classes não é feita sintaticamente, mas sim, contextualmente, através dos relacionamentos das classes com os seus vizinhos. Porém, ambos podem ser utilizados na detecção de classes desproporcionalmente grandes e complexas ou mau uso de diagramas.

Egyed et al. (2008) expõem uma técnica para a geração automática de um conjunto de mudanças concretas para consertar inconsistências de diagramas UML baseados na ferramenta *Rational Rose* e fornece informações sobre o impacto de cada uma destas mudanças. As restrições em que esta técnica se baseia são definidas na forma de regras de consistência, que não podem ser violadas.

Este artigo está interessado no uso correto e concreto da UML, por exemplo, se o usuário for colocar uma chamada de uma função no diagrama de sequência que não pertence a uma classe, este estudo vai apontar quebra em uma das regras de consistência.

3.2. Transformação de modelos em grafo

Os estudos cujo foco é a transformação de modelos UML em grafo são importantes para o trabalho proposto, visto que eles retiram dúvidas sobre a orientação do grafo, restrições de tamanho de nós e arestas do grafo, validação do modelo UML no grafo e práticas de leitor de diagramas UML.

Dentro deste contexto, existe um trabalho feito por Holscher et al. (2006) cujo objetivo é transformar um modelo UML em um grafo executável e que represente o estado atual da modelagem para o metamodelo UML 1.5. Este grafo simulado é capaz de ser executado no intuito de permitir ao arquiteto de software validar aspectos básicos do modelo, ainda em tempo de arquitetura.

Outro estudo na área de transformação em grafo foi realizado por Shen & Petriu (2002). Foi proposto um algoritmo baseado em grafo para transformar automaticamente um modelo UML em um LQN[4]. A entrada deste algoritmo é um XMI e a saída é uma estrutura baseada em grafo com o poder de realizar consultas sobre o modelo.

Existe um terceiro estudo que resultou em uma ferramenta chamada *SDMetrics*. Esta ferramenta foi foco de uma análise mais depurada, pois ela explora o *parser* de arquivos de exportação que representam o metamodelo UML1.0. As *tags* XML deste metamodelo e as formas de exportação dos relacionamentos entre classes são definidos e expostos. O *SDMetrics* detecta automaticamente designs incompletos, incorretos, redundantes ou inconsistentes, e percebe problemas como dependência circular, violação e convenções de nome (SDMetrics, 2012). Este *parser* foi utilizado como base para a leitura do arquivo de exportação proposta no trabalho.

[4] Layered Queueing Network

3.3. Algoritmos de agrupamento

Na área de clusterização, Fasulo (1999) desenvolveu um artigo que mostra o problema de agrupamento de uma perspectiva diferente e com objetivos diferentes. Analisa a força e a fraqueza de cada algoritmo de agrupamento dependendo da sua aplicação. No final, um conjunto de passos é estipulado para que a escolha de qual algoritmo utilizar se torne mais fácil.

Este estudo foi utilizado para se definir o *k-means* como o algoritmo de agrupamento. A escolha do algoritmo passa por quatro etapas:

- *Tipo do dado de entrada*: Consiste no formato do dado de entrada do algoritmo. Pode ser numérico (quaisquer números do conjunto dos números reais como é o caso da entrada do algoritmo *k-means*) ou categórico (números discretos pré-definidos, como zero ou um, verdadeiro ou falso, que é o caso da entrada do algoritmo de *Gibson*).
- *Escalabilidade*: Consiste na capacidade do algoritmo em aumentar o número de entradas, sem que o processamento torne impossível o seu uso. Nesta etapa, é realizado um estudo da complexidade computacional dos algoritmos.
- *Ruído*: Consiste no estudo da aleatoriedade da inicialização do algoritmo. A aleatoriedade da inicialização pode levar elementos a grupos diferentes, por isso é necessário executar o algoritmo determinado número de vezes para que haja uma convergência do resultado.
- *Apresentação do resultado*: Consiste no formato do resultado e no número de grupos, pré-definidos ou calculados pelo próprio algoritmo.

3.4. Obtenção dos core concepts

Este é um dos tópicos mais significantes porque dirá quais as classes mais importantes dentro de um diagrama UML, de acordo com uma métrica a ser aplicada no grafo de classes.

Hsi et al. (2003) desenvolveram um trabalho para extrair os conceitos centrais de um grafo que representa uma ontologia. O artigo utiliza dois grafos, um que representa uma parte específica do *Notepad* e outro que representa o *Windows 95/98 CD Player*. Em cima destes dois grafos foram testadas cinco métricas:

- *Grau de centralidade*: Mede o número de arestas de cada nó. Quanto maior o número de arestas, mais central é o conceito.
- *Proximidade*: Mede a distância média deste nó a todos os outros.
- *Caminho*: Mede o número de vezes que um determinado nó está presente em um caminho mais curto de um nó ao outro no grafo, para todos os nós do grafo.
- *Informação da centralidade*: Mede o número de vezes que um nó está presente em todos os caminhos de nó a nó, onde os caminhos começam de um nó específico ou aleatório.
- *Eigenvector*: Mede a centralidade de um nó de acordo com a centralidade dos nós vizinhos.

O teste para achar a melhor métrica era retirar cada nó que tinha centralidade alta, rodar novamente a métrica, e ver a diferença que ocorreu. Se houver uma diferença grande, quer dizer que aquele nó é, de fato, central no grafo.

Depois de fazer este teste para todos os nós do grafo, estes são ordenados, na ordem do maior para o menor, segundo a diferença no resultado quando retirado. A melhor métrica encontrada depois da execução dos testes foi o *Caminho*.

Este artigo foi fundamental porque é possível transpor a métrica do *Caminho* da ontologia para o diagrama de classe UML, basta dar pesos às arestas apropriadamente no grafo. No caso das ontologias, todas as arestas tinham o mesmo peso.

3.5. Definições de métricas de diagramas UML

Os artigos que abordam definições de métricas para diagramas UML têm o mesmo objetivo fim deste trabalho, visto que tentam agregar informações extras aos diagramas ainda em fase de arquitetura e design dentro do processo de software.

Podgorelec & Heric (2007) comparam a complexidade de um modelo ao grau de entropia de códigos Java. O pilar do estudo é um método denominado *Long-range power law correlation*. Basicamente, o diagrama UML e a codificação Java são transformados em um conjunto de símbolos. A partir daí, calcula-se a correlação entre as *strings* dos símbolos dos diagramas e as *strings* dos símbolos da codificação para estabelecer uma medida fractal.

A medida fractal α[5] é calculada para vários projetos e estas medidas são comparadas com a dificuldade real encontrada nestes projetos, para se estabelecer uma correlação entre o design e a codificação. Esta correlação ficou entre 0.62 e 0.64. Este valor representa a relação existente no esforço de conceber os diagramas UML com o esforço de codificar um projeto.

Este artigo foi importante para definir que a métrica de complexidade das classes do design pode ser utilizada para definir uma estimativa de complexidade da codificação.

Chen & Tang (2002) definiram métricas sobre os diagramas de classe, colaboração e atividade para a ferramenta *Rational Rose*. O diagrama de classe é transformado em uma lista de classes, o diagrama de colaboração em uma sequência de colaboração e o diagrama de atividades em um grafo. Algumas métricas são calculadas, juntamente com regras pré-definidas de boas práticas de orientação a objetos, com o objetivo de identificar classes com potenciais inconsistências em OO. Além disto, o estudo calcula métricas de esforço de manutenção, produtividade e possibilidade de retrabalho.

Outro artigo da área de métricas é o desenvolvido por Debnath et al. (2008) e trata da junção do diagrama UML com aspectos. O paradigma de orientação a aspectos remonta o paradigma de orientação a objetos na introdução de uma noção de aspecto como um novo tipo de classe. Assim, foram criadas métricas para UML que operam no AOP[6]. A classe *Aspect* estende a metaclasse *Class* e a classe *Realize* estende a metaclasse *Association*. Um *Aspect* se relaciona com

---

[5] Valor entre 0.5 e 1.0 que representa a convergência de F(l) e l em uma escala logaritmica, onde l é o ponto final e determina a diferença entre dois pontos no eixo X, e F(l) é a distinção de dois comportamentos diferentes, utilizando a diferença entre l e l_0(ponto inicial) no eixo Y. $F(l) = l^{\alpha}$.

[6] Aspect Oriented Design

vários *Realize* para modelar os *crosscutting concerns*, definidos na orientação a aspectos.

Desta maneira, pela combinação dos diagramas de sequência e de classes com os aspectos que interceptam associações, pode-se inferir o WMA [7] dos métodos, e na soma de todos os WMA's dos métodos de uma classe, extrai-se a estimativa de esforço de aspectualização de uma classe. Além disto, é possível, através do AOP, calcular estimativas de coesão entre classes (número de *Aspects* que as classes compartilham) e a dificuldade de implementar aspectos em um projeto (número de *Aspects* e *Realizes* encontrados no diagrama).

O último trabalho relacionado estudado neste tópico foi o de Martin (1994). O ponto principal é o estudo da interdependência dos subsistemas dos diagramas. Quanto maior for a interdependência de um projeto, maior a sua rigidez e a sua dificuldade de manutenção e menor o seu reúso.

A rigidez se dá no fato de uma simples mudança em uma classe, dá início a uma cascata de mudanças em variados pontos do design. O não reúso e a dificuldade da manutenção são consequências da rigidez.

Para estudar a dependência entre os módulos do design e entre as classes, foram definidas três categorias:

- Classes dentro de uma categoria que estão fechadas em conjunto contra qualquer mudança. Significa que, se uma classe precisa ser modificada, todas as classes dentro desta categoria deveriam ser modificadas.
- Classes dentro de uma categoria que são reusadas em conjunto. Estas classes têm interdependência forte e não podem ser separadas uma das outras.
- Classes dentro de uma categoria que compartilham algumas funções ou querem conquistar objetivos em comum.

O foco, neste momento, é aplicar métricas para estas categorias e suas dependências. Foram definidas três métricas para as categorias:

1. *Ca*: Número de classes de fora da categoria que dependem de classes de dentro desta categoria.

---

[7] Weight of Method by Aspect

2. *Ce*: Número de classes de dentro da categoria que dependem de classes de fora da categoria
3. *Instabilidade*: $I = Ce/(Ca + Ce)$. Esta métrica tem valores entre zero e um. $I = 0$ indica categoria com o máximo de estabilidade. $I = 1$ indica categoria com o máximo de instabilidade.

Para ter interdependência baixa, os conjuntos de classe definidos em categorias deveriam ter um nível de estabilidade maior. Assim, uma mudança neste conjunto não acarretaria uma cascata de mudanças em outras classes, aumentando o reúso, a flexibilidade e a manutenção.

As métricas de interdependência *Ca*, *Ce* e *Instabilidade* podem ser aplicadas depois da componentização proposta no *AnalisadorUML*, para agregar mais métricas à ferramenta.

3.6. Ferramentas de Análise UML

Existem no mercado muitas ferramentas de edição de diagramas UML, mas poucas que se propõem a analisá-los. Além do *SDMetrics*, descrito na seção 3.2, outras duas ferramentas ganharam notoriedade na análise e aplicação de métricas para diagramas.

A primeira ferramenta é a *UML Class diagram metrics tool* (Girgis & Mahmoud, 2009). Ela lê o arquivo de exportação dos diagramas de classes que utilizam o metamodelo UML 1.0, grava informações das classes, seus métodos e atributos, seus relacionamentos com outras classes e, aplica métricas conhecidas de tamanho, acoplamento e herança, que são listadas abaixo. Não existe nenhum algoritmo por trás, apenas execução de fórmulas das métricas.

- Métricas de Tamanho

Nome	Descrição
Métricas de Li & Henry (1993a, b)	RFC (Número de métodos locais + número de métodos chamados por métodos locais), LCOM (Número de conjuntos disjuntos de métodos locais), WML (Somatório da complexidade ciclomática de todos os métodos locais).
NOM	Número de métodos locais.
SIZE2	Número de atributos + número de métodos locais.
PIM	Métodos públicos.

NIM	Número de todos os métodos públicos, protegidos e privados do diagrama.
NIV	Número de variáveis instanciáveis.

Tabela 1 - Métricas de tamanho do UML Class Diagram Metrics Tool.

- Métricas de Acoplamento

Nome	Descrição
DAC	Número de atributos de uma classe que contêm outra classe.
DAC'	Número de classes diferentes que são utilizadas como tipos.
Métricas de Briand et al. (1997)	IFCMIC (*Inverse Friend Class-Method import Coupling*), IFCAIC (*Inverse Friend Class-Attribute Import Coupling*), IFMMIC (*Inverse Friend Method-Method Import Coupling*), IFCMEC (*Inverse Friend Class-Method export Coupling*), IFCAEC (*Inverse Friend Class-Attribute Export Coupling*), IFMMIC (*Inverse Friend Method-Method Export Coupling*).
OCAEC	Acoplamento de exportação do atributo da classe.
OCAIC	Acoplamento de importação do atributo da classe.
ACAIC	Acoplamento de importação do atributo da classe ancestral.
DCAEC	Acoplamento de exportação do atributo da classe descendente.
ACAEC	Acoplamento de exportação do atributo da classe ancestral.
DCAIC	Acoplamento de importação do atributo da classe descendente.
ACMEC	Acoplamento de exportação do método da classe ancestral.
OCMIC	Acoplamento de importação do método da classe.
DCMEC	Acoplamento de exportação do método da classe descendente.
OCMEC	Acoplamento de exportação do método da classe.
ACMIC	Acoplamento de importação do método da classe ancestral.
DCMIC	Acoplamento de importação do método da classe descendente.
Métricas de Harrison et al. (1998)	CBO (*Coupling Between Objects*, número de classes às quais uma determinada classe está acoplada) e NAS (Número de associações entre a classe e seus pares).
NASS	Número de Associações do sistema.

Métricas de Bansiya & Davis (2002)	QMOOD++ (*Quality Metrics for Object-Oriented Development*).
DCC	*Direct Class Coupling*.

Tabela 2 - Métricas de Acoplamento do UML Class Diagram Metrics Tool.

- Métricas de Herança

Nome	Descrição
DIT	Profundidade da Herança
NOC	Número de Filhos.
Métricas de Brito et al. (1996)	MHF (*Method Hiding Factor*), AHF (*Attribute Hiding Factor*), POF (*Polymorphism Factor*) e COF (*Coupling Factor*)
AIF	Fator de Herança do Atributo.
MIF	Fator de Herança do Método.
NMO	Número de métodos substituídos em uma sub-classe.
NMI	Número de métodos herdados em uma sub-classe.
NMA	Número de métodos definidos em uma sub-classe.
SIX	Índice de especialização de cada classe.

Tabela 3 - Métricas de Herança do UML Class Diagram Metrics Tool.

A segunda ferramenta em questão foi fruto do artigo desenvolvido por Ghosheh & Black (2009). O *WapMetrics* também define o processo de leitura do arquivo de exportação dos diagramas UML, mas foca no uso de métricas utilizadas para aplicações Web através de uma extensão do UML denominada *Conallen*.

As métricas aplicadas na modelagem Web são descritas a seguir:

- *Métricas de Tamanho*: Número total de páginas do servidor, número de total de páginas do cliente, número total de páginas, número total de páginas de formulário, número total de elementos do formulário, número total de componentes de script do cliente.
- *Complexidade estrutural*: Número total de relacionamentos de links, número total de relacionamentos de submit, número total de relacionamentos de build, número total de relacionamentos de forward, número total de relacionamentos de include, número total de relacionamentos de *tag*.
- *Acoplamento de controle*: Razão entre a soma de todas as métricas de complexidade estrutural e o número de páginas.

- *Acoplamento de dados*: Razão entre o número total de páginas de formulário e o número total de páginas do servidor.
- *Reusabilidade*: Razão entre o número total de relacionamentos include e o número total de páginas do servidor.
- *Outras métricas*: Número total de classes, número total de atributos, número total de métodos, número total de associações, número total de relacionamentos de agregação.

3.7. Estado da arte

Nesta seção será introduzida uma noção de ferramentas CASE em *workbenches* e quais são as ferramentas atuais de edição de diagramas.

Antes de definir um *workbench*, é necessário definir o que é uma ferramenta CASE. A sigla CASE significa *Computer-Aided Software Engineering*. Uma ferramenta CASE é um aplicativo que auxilia os profissionais envolvidos na tarefa de produzir sistemas. O tipo de ajuda que a ferramenta fornece depende exclusivamente da proposta do fabricante (Carlos, 2012). Por este motivo, as ferramentas são divididas em três categorias. São elas:

- *Lower CASE*: Ferramentas de codificação.
- *Upper CASE*: Ferramentas de análise, projeto e implementação.
- *Integrated CASE*: União de Upper e Lower CASE.

O *workbench* CASE é um conjunto de ferramentas utilizadas em determinadas fases do processo de software, como a fase de projeto, implementação e teste. A vantagem de agrupar as ferramentas CASE em um *workbench* é que as ferramentas podem trabalhar em conjunto, para oferecer um apoio mais abrangente. As ferramentas de *workbench* podem ser integradas por meio de arquivos compartilhados, de um repositório compartilhado ou de estruturas de dados compartilhadas (Sommerville, 2004).

Os *workbenches* de análise e projeto são projetados para dar apoio à modelagem de sistemas durante os estágios de análise e projeto do processo de software, e são estas que interessam ao escopo do trabalho. Elas fornecem apoio à criação, à edição e à análise das notações gráficas utilizadas em métodos estruturados.

As ferramentas que compõem um *workbench* de análise e projeto são integradas por meio de um repositório ou um arquivo compartilhado, cuja estrutura é de propriedade do fornecedor do *workbench*. A Figura 2 demonstra as ferramentas de um *workbench* de análise e projeto:

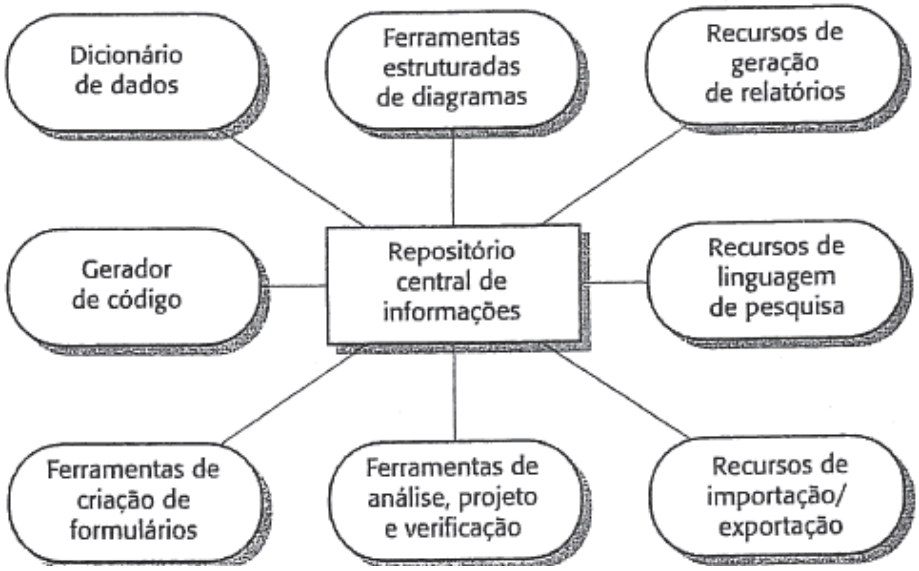

Figura 2 - Ferramentas de um *workbench*.

O *workbench* de análise e projeto inclui:

- *Editores de diagramas*: Criação de diagramas de fluxo de dados, hierarquias de objetos, diagramas de relacionamento de entidades, etc. Estes editores não são apenas ferramentas de desenho, mas associam o tipo de entidades no diagrama. Eles capturam informações sobre essas entidades e armazenam essas informações no repositório central.
- *Ferramentas de análise, de projeto e de verificação*: Processam o projeto e produzem um relatório de erros e anomalias. Elas podem ser integradas com o sistema de edição, de modo que os erros do usuário sejam percebidos em um estágio inicial do processo.
- *Linguagens de pesquisa de repositório*: Permitem ao projetista encontrar, no repositório, projetos e informações relacionadas a projetos.
- *Dicionário de dados*: Mantém informações sobre as entidades utilizadas em um projeto do sistema.

- *Ferramentas de definição e geração de relatórios*: Captam informações do depósito central e, automaticamente, geram documentação do sistema.
- *Ferramentas de definição de formulários*: Permitem especificar formatos de tela e de documentos.
- *Recursos de importação/exportação*: Permitem o intercâmbio de informações do repositório central com outras ferramentas de desenvolvimento.
- *Geradores de código*: Geram código ou estruturas de código automaticamente, a partir do projeto capturado no repositório central.

Existem diversas ferramentas que podem ser consideradas workbenches CASE no desenvolvimento de modelos UML. A maioria delas possuem, primeiramente, o editor de diagramas e a ferramenta de análise, de projeto e de verificação embutidos, bem como o dicionário de dados e os recursos de importação/exportação. O XMI se transformou em um quase-padrão de exportação de diagramas UML, o que dificulta o uso do XMI é que cada *workbench* CASE tem o seu próprio dicionário de dados. Assim, para o usuário importar algum diagrama, é necessário o estudo do dicionário específico da ferramenta.

As ferramentas de definição de geração de relatórios e de formulários, e os geradores de código são opcionais nas ferramentas CASE que existem no mercado e têm um suporte pequeno.

O estudo proposto é um ponto adicional destes componentes que formam o *workbench* de análise e projeto. Ele será empregado depois que a arquitetura estiver pronta, pela utilização do dicionário de dados e do recurso de importação/exportação. O tipo de ferramenta, fruto deste trabalho, poderia ser facilmente aderido a um *workbench* CASE.

De acordo com Silva & Videira (2008), as principais workbenches de análise e projeto *open source* estão descritos conforme a Tabela 4:

Título	URL
Argo UML	*argouml.tigris.org*
Eclipse Modeling Tools	*www.eclipse.org/modeling/mdt/?project=uml2*
MonoUML	*monouml.sourceforge.net*
Papyrus UML	*www.papyrusuml.org*
Star UML	*staruml.sourceforge.net/en/*
TopCased	*www.topcased.org*
Umbrello	*uml.sourceforge.net/index.php*
UMLLet	*www.umlet.com*

Tabela 4 - Ferramentas *Open Source*.

E os principais *workbenches* de análise e projeto comerciais estão descritas na Tabela 5:

Empresa	Ferramenta	URL
Aonix	Software through Pictures	*www.aonix.com/index.html*
Artisan	Artisan Studio	*www.artisansw.com*
Borland	Together	*www.borland.com/us/products/together/index.html*
Casewise	Casewise Corporate Modeler	*www.casewise.com*
Computer Associates	Erwin, Bpwin	*www.ca.com*
Compuware	OptimalJ	*www.compuware.com/products/optimalj/*
Gentleware	Poseidon	*www.gentleware.com*
IBM Rational	Rational Software Architect	*www-306.ibm.com/software/awdtools/architect/swarchitect*
IBM Rational	Rational Software Modeler	*www-306.ibm.com/software/awdtools/modeler/swmodeler/*
IDS Scheer AG	Aris	*www.ids-scheer.com*
Mega International	Mega Modeling Suite	*www.mega.com*
Metastorm	ProVision	*www.metastorm.com/products/mpea.asp*
Microgold Software	WithClass	*www.microgold.com*
Microsoft	Visio	*www.microsoft.com/office/visio*
No Magic's	MagicDraw UML	*www.magicdraw.com*
Oracle	Designer, Developer	*www.oracle.com*
Grandite	Silverrun	*www.silverrun.com*
Softeam	Objecteering	*www.objecteering.com*
Sparx Systems	Enterprise Architect	*www.sparxsystems.com*
Sybase	PowerDesigner	*www.sybase.com*
Telelogic	Doors, Focal Point, System Architect, Rhapsody, TAU	*www.telelogic.com*
Troux Technologies	Metis Architect	*www.troux.com/products/metis_architect/*
Universidade de Paderborn, Alemanha	Fujaba	*wwwcs.upb.de/cs/fujaba/*
Visible Systems Corporation	Visible Analyst	*www.visible.com*
Visual Object Modelers	Visual UML	*www.visualobject.com*

Tabela 5: Ferramentas Comerciais.

Vale salientar que existem também duas ferramentas UML para JUDE conhecidas como *Astah Community* (versão *open source*) e o *Astah Professional* (versão comercial), que não se encontram nas tabelas anteriores, e são umas das mais utilizadas. O *Astah Professional 6.5* será a ferramenta de edição UML escolhida por este trabalho para a montagem dos diagramas de classe do projeto e exportação dos dados.

Todas estas ferramentas seguem a ideia do *workbench* CASE de análise e projeto de acordo com a Figura 2. Estudos feitos para se extrair informações mais concretas destes *workbenches* CASE foram descritos nesta seção.

Este trabalho se dispõe a entrar nesta mesma área. Métricas de tamanho e acoplamento, informações de conceitos centrais, criação de componentes e estimativas de complexidade e coesão serão os resultados aderidos aos *workbenches* CASE.