Emprego de Métricas de Qualidade em um Protótipo de uma Ferramenta CASE para UML

RESUMO. O presente artigo descreve o atual estágio de desenvolvimento de um protótipo de ferramenta CASE para UML – Unified Modeling Language desenvolvido na FURB - Universidade Regional de Blumenau. O protótipo apresenta como diferencial em relação as demais ferramentas disponíveis, dois aspectos importantes: um módulo de análise de verbos e substantivos e um módulo de quantificação de métricas de qualidade do texto descritivo dos casos de uso que permitem a obtenção de software de melhor qualidade.

ABSTRACT. This paper describes the actual project's development stage of a UML CASE tool that is being developed at FURB - Universidade Regional de Blumenau. The prototype presents two main points that should be commented: first of all a verbs-nouns analysis module and a metric-quality module. Both of them are applied to the textual description of use cases in sense of assure a better software quality specification .

Palavras-chave: UML, Ferramentas CASE, Orientação a Objetos.

1. Introdução

A UML é a mais recente proposta do mercado para padrão de notação gráfica usada para desenvolvimento de sistemas orientados a objetos. A ferramenta descrita neste documento tem caráter didático, e visa instrumentalizar o aluno na disciplina de Tópicos III - Análise e Projeto Orientados a Objetos, tendo em vista que, as ferramentas comerciais apresentam algumas restrições, tais como: complexidade e altos requisitos de hardware , os quais limitam seu emprego em ambiente acadêmico.

1.1. Retrospectiva

As linguagens de modelagem tem sido propostas como uma solução para documentação e comunicação entre cada estágio de um projeto de software seguindo a filosofia de orientação a objetos (porém servindo também , em alguns casos, para as metodologias tradicionais).

Verificou-se ao longo do tempo, e após várias propostas metodológicas, que o resultado de cada fase deve ser preservado e passado para a fase seguinte através de uma linguagem de modelagem que possua uma sintaxe formal, e que portanto, permita a geração de um modelo não ambíguo do sistema a ser implementado.

Apesar de sua aprovação relativamente recente (set/97) pela OMG - Object Management Group, já existem ferramentas CASE no mercado que implementam a nova notação. Naturalmente, estas ferramentas possuem um custo relativamente alto, face a realidade do ambiente acadêmico. Não obstante isto, como estratégia de marketing, as empresas desenvolvedoras de tais ferramentas disponibilizam através da Internet, versões de demonstração, as quais possuem prazo de validade que normalmente não ultrapassa os 30 dias.

Neste contexto, colocam-se algumas questões importantes: a primeira diz respeito a posição da Universidade como entidade com fins de formação acadêmica. Não é razoável que os alunos

sejam formados em uma ferramenta específica, mas, que adquiram conhecimento sobre a filosofia que norteia as várias implementações da proposta em questão.

A Segunda colocação é mais operacional, e refere-se ao fato de que, tais ferramentas possuem um tamanho (em bytes) considerável, o que implica que a cada x dias haja necessidade de alocação de tempo para novamente realizar o procedimento de download e re-instalação do software. Esta implicação poderia ser refutada com o argumento de que o processo de download poderia ser disparado automaticamente de x em x dias para os servidores a Universidade. Porém, se analisado do ponto de vista do aluno, que pode vir a necessitar de uma cópia do referido software em sua casa, tendo em vista realizar suas tarefas, a questão pode ser considerada mais complexa.

Uma terceira questão, porém não menos importante, refere-se ao fato de que, as ferramentas comerciais, de maneira geral, apresentam um grau de complexidade e abrangência desnecessários no escopo do ambiente acadêmico.

E finalmente, a possibilidade de agregação de novas facilidades desenvolvidas in-house (tais como : módulo de análise de verbos e substantivos, módulo de métricas de qualidade, etc.) que venham a incorporar novas facilidades motivaram o início de um processo de desenvolvimento de um protótipo de uma ferramenta CASE que viesse a implementar um sub-conjunto dos diagramas propostos pela UML, a qual será brevemente descrita neste trabalho.

Antes da apresentação do protótipo própriamente dito, far-se-á uma breve apresentação da proposta UML, tendo em vista situar o leitor dentro do contexto do projeto.

2. Fundamentação Teórica

Com o aumento gradativo da complexidade dos sistemas, envolvendo os seus requisitos, sua funcionalidade e seu gerenciamento, surge a necessidade de encontrar novos paradigmas para construção de software, os quais facilitam a especificação e manutenção dos sistemas. Orientação a objetos é um paradigma que se propõe a desempenhar esta tarefa.

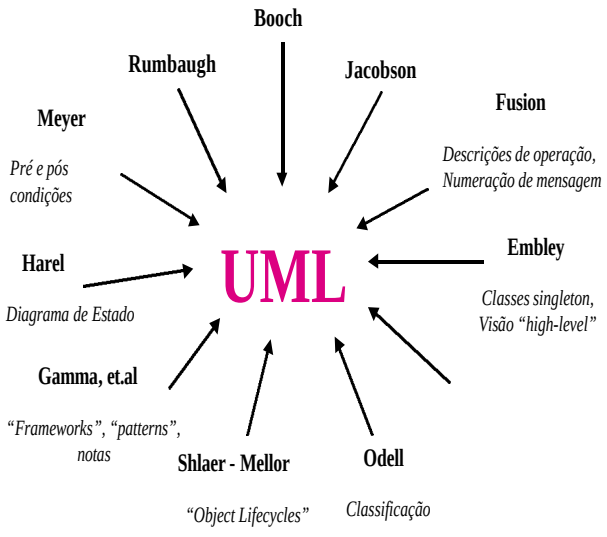

Figura 1 – Convergência de tecnologias para UML

Um dos fatores críticos encontrados nas atuais incursões do desenvolvimento de sistemas orientados a objetos, é a falta de definição de uma linguagem metodológica. Sem a definição da metodologia, corre-se o risco de fracassos em projetos por uma clara falta de padronização[7]. As técnicas orientadas a objetos tem por objetivo simplificar o projeto de sistemas complexos, pois o sistema pode ser visto como uma coleção de objetos, os quais podem ainda ser reaproveitados em projetos futuros [11]. Baseados neste paradigma foram disponibilizadas no mercado várias metodologias para a análise e projeto de sistemas orientados a objetos, tais

como: Booch, OOSE, OMT e UML.

2.1. A Unified Modeling Language – UML

A *Unified Modeling Language* (Linguagem de Modelagem Unificada) - UML, merece destaque especial, uma vez que ela foi projetada para servir como uma linguagem de modelagem orientada a objetos, indiferente ao método de desenvolvimento, pois pode substituir - sem perda de informação - as notações dos métodos Booch, OMT e OOSE, entre outros [7]. Ela pode ser definida como uma linguagem para especificação, visualização, construção e documentação de artefatos de sistemas de *software*. Ela representa uma coleção das melhores experiências na área de modelagem de sistemas OO (orientados a objetos), as quais tem obtido sucesso na modelagem de grandes e complexos sistemas [8].

A figura 1 apresenta as principais tecnologias que convergiram para a proposta UML. Segundo [MUL97], os respectivos trabalhos dos três mais respeitáveis especialistas em modelagem OO (Grady Booch, Ivar Jacobson e Jim Rumbaugh) representavam diferentes aspectos do mesmo problema, então eles resolveram unir seus esforços para dar início a um processo de unificação.

Dentro desse esforço coletivo, ocorreram a unificação dos métodos Booch - de Grady Booch, Objectory - de Ivar Jacobson, e OMT - de Jim Rumbaugh, e a incorporação de idéias de outros metodologistas, mais notavelmente Wirfs-Brocks, Wark, Cunningham, Rubim, Harel, Gamma, Ulissides, Helm, Johnson, Meyer, Odell, Embley, Coleman, Coad, Yourdon, Shlaer, e Mellor, conforme salienta [2] .

Com a unificação desses métodos de modelagem, a UML provê a base para um comum, estável, e expressivo método de desenvolvimento orientado a objetos, sendo, consequentemente, o sucessor direto e mais compatível dos métodos que foram unificados [1]. Conforme [8], Booch, Rumbaug e Jacobson adotaram quatro metas para este processo de unificação:

a) Representar sistemas completos (em vez de somente porções de software) usando conceitos de OO;
b) Estabelecer uma ligação explícita entre conceitos e código executável;
c) Levar em conta os fatores de escala que são inerentes aos sistemas complexos e críticos;
d) Criação de uma linguagem de modelagem utilizável por ambos homens e máquinas.

Como a notação da UML foi projetada para servir como uma linguagem de modelagem OO, indiferente ao método de desenvolvimento, ela pode substituir - sem perda de informação - as notações de métodos como Booch, OMT e OOSE (*Object Oriented Software Engineering*, também chamado de *Objectory*). Melhores referências sobre o surgimento e evolução da proposta pode ser obtido em [7].

2.2. ARQUITETURA DA LINGUAGEM

Conforme [8], a UML possui uma arquitetura de metamodelo de quatro camadas, quais sejam: meta-metamodelo, metamodelo, modelo e objetos do usuário. Esta arquitetura provê a infraestrutura para definição das semânticas precisas requeridas pelos modelos complexos. A partir do momento em que a camada de metamodelo torna-se muito complexa, ela pode ser

decomposta em pacotes lógicos, os *packages* da UML, os quais permitem incrementar a modularidade da linguagem.

A camada de meta-metamodelo forma a fundamentação para a arquiterura de metamodelagem. Sua responsabilidade primária é definir a linguagem para especificação de um metamodelo. Alguns exemplos são: MetaClass, MetaAttribute e MetaOperation . Um metamodelo é uma instância de um meta-metamodelo. A responsabilidade primária da camada metamodelo é definir uma linguagem para especificação de modelos. Exemplos de metaobjetos na camada de metamodelagem são: *Class*, *Attribute*, *Operation* e *Component*. Um modelo é uma instância de um metamodelo. A responsabilidade primária da camada modelo é definir uma linguagem que descreve um domínio de informação. E finalmente, os objetos do usuário são uma instância de um modelo. A responsabilidade primária da camada dos objetos do usuário é descrever um domínio de informação específico.

2.3.DIAGRAMAS DA UML

Para suprir o seu propósito geral, a UML disponibiliza uma série de diagramas para que seja possível levantar a especificação dos requisitos, a estrutura estática, a parte dinâmica ou comportamental, e os aspectos de implementação do sistema ou artefato do sistema [2]. Os modelos definidos pela UML para representação de sistemas são os seguintes:

Modelo de Classes (Class Model)	define a estrutura estática de classes;
Modelo de Estados (State Model)	expressa o comportamento dinâmico de objetos;
Modelo de Casos de Uso (Use Case Model)	descreve as exigências do usuário;
Modelo de Interação (Interaction Model)	representa os cenários e fluxos de mensagens;
Modelo Implementação (Implementation Model)	apresenta os dispositivos de trabalho;
Modelo de Disposição (Deployment Model)	provê detalhes sobre a distribuição de processos

Tabela 1 - Modelos definidos pela UML

Os modelos são manipulados por meio de representações gráficas que são projeções dos elementos contidas em um ou mais modelos. Podem ser construídas de diferentes perspectivas para um modelo básico - cada uma pode mostrar tudo ou parte do modelo, e cada uma possui um diagrama correspondente[8]. A UML define nove diferentes tipos de diagramas conforme apresentado na tabela 2.

Diagramas de Atividade (Activity Diagram)	representam o comportamento de uma operação e o seu funcionamento;
Diagramas de Classe (Class Diagram)	Representam a estrutura estática das classes e seus relacionamentos;
Diagramas de Colaboração (Collaboration Diagram)	São uma representação de objetos, ligações e interações;
Diagramas de Componente (Component Diagram)	Representam os componentes físicos de uma aplicação;
Diagramas de Disposição (Deployment	Representam a disposição dos componentes de hardware que compõem

Diagram)	o sistema;
Diagramas de Objeto (Object Diagram)	Representam objetos e suas relações, e correspondem a diagramas de colaboração simplificados que não representam dispersões de mensagem;
Diagramas de Seqüência (Sequence Diagram)	São uma representação temporária de objetos e as interações entre eles;
Diagramas de Estados (Statechart Diagram)	Representam o comportamento de uma classe em termos de estados;
Diagramas de Casos de Uso (Use Case Diagram)	Representam as funções de um sistema do ponto de vista do usuário.

Tabela 2: diagramas que compõem a UML

Na prática, a UML é simplesmente outra representação gráfica de um modelo semântico comum. Porém, combinando os elementos mais úteis dos métodos orientados a objetos, e estendendo a notação para cobrir aspectos novos de desenvolvimento de sistema. Em suma, os diagramas podem mostrar tudo ou parte das características dos elementos de um modelo, com um nível de detalhe satisfatório dentro do contexto de um determinado diagrama.

3. QUALIDADE DE SOFTWARE

A cada momento verifica-se o esforço da comunidade de software e pesquisadores da área no sentido de desenvolver produtos de software de melhor qualidade. Já é consenso que, uma especificação de requisitos de baixa qualidade conduz a erros de projeto que somente muito tarde no ciclo de desenvolvimento serão descobertos.

Embora haja um número significativo de vantagens no uso de linguagens de especificação formais, seu efetivo emprego não é uma prática comum. Como os requisitos que o contratante espera do desenvolvedor devem ser entendidos por ambas as partes, as especificações , na maioria das vezes, é realizada em linguagem natural. O uso de linguagem natural para descrever sistemas complexos e dinâmicos apresenta pelo menos 3 problemas sérios, quais sejam: ambigüidade, imprecisão e inconsistência.

A importância em produzir documentos de especificação mais corretos tem levado a indústria de software a produzir um número significativo de ferramentas para apoio a criação e gerenciamento de documentos de especificação de requisitos. Poucas entretanto, atuam na área de avaliação de qualidade do documento em si.

Em função desta constatação, o Software Assurance Technology Center (SATC) do Goddard Space Flight Center (GSFC-NASA) iniciou o desenvolvimento de uma ferramenta que deve ser utilizada para análise de especificação de requisitos em linguagem natural.[13] Basicamente a ferramenta analisa o documento buscando expressões em linguagem natural que foram previamente classificadas como indicadores de qualidade. O relatório produzido pela ferramenta é utilizado para identificar indicativos de uma descrição que poderá causar problemas.

As principais características de um documento de especificação de requisitos são: ser completo, consistente, correto, ser passível de modificações, classificável, ser passível de verificação e não ambíguo. Apesar dos atributos de qualidade serem subjetivos, existem alguns aspectos da documentação que podem ser mensurados. Por exemplo, o tamanho que é uma primitiva utilizada em um grande número de métricas, pode ser diretamente aplicado: o

tamanho de um documento pode ser totalizado para número de páginas, número de linhas, número de parágrafos, etc. Estes contadores fornecem um indicativo de como o documento de especificação está organizado. Também é possível contar a ocorrência de palavras e expressões específicas, as quais sinalizam pontos fracos ou fortes na descrição[14].

O trabalho do SATC iniciou pela compilação de uma lista de atributos de qualidade que esperava-se os documentos de especificação de requisitos viessem a apresentar. Esta compilação realizou-se a partir de uma série de documentos de especificação de sistemas desenvolvidos pela NASA e obtidos na biblioteca da SATC. O próximo passo foi agrupar àqueles aspectos que poderiam ser objetivamente e quantitativamente mensurados. Nove categorias de indicadores de qualidade foram estabelecidas neste estudo. Estas categorias estão divididas em 2 grupos: aquelas relacionadas a análise de comandos específicos, quais sejam : imperativas, continuações, diretivas, opções e frases “fracas”; e aquelas relacionadas a totalização do documento, quais sejam: tamanho, estrutura do texto, profundidade da especificação (em termos de itemização) e legibilidade. A partir daí, foi construído um protótipo de software para automaticamente mensurar os atributos identificados anteriormente. Uma descrição mais detalhada do projeto pode ser obtida em [3,4,5,6,9,10].

Um aspecto importante enfatizado no trabalho de [12], refere-se ao fato de que, a ferramenta produzida pelo SATC, não verifica a correção da especificação em termos de produto final, mas permite a produção de documentos de especificação que utilizam uma linguagem mais apropriada a especificação de requisitos.

As questões acima relatadas relacionam-se ao contexto do presente trabalho a partir de uma das conclusões apresentadas por Wilson [12]: “ quem escreve documentos de especificação de requisitos deveria ser ensinado a como escrever estruturas de linguagem simples e diretas e como fielmente seguir um enfoque disciplinado na criação destes documentos. O documento não tem que ser interessante, mas passível de entendimento”.

O que difere o protótipo de que trata o presente trabalho, das demais ferramentas disponíveis é que ele agrega um passo a mais que interliga as descrições em linguagem natural dos casos de uso aos demais diagramas através de um passo de compilação de verbos e substantivos. A partir dos estudos acima descritos, iniciou-se o desenvolvimento de um novo módulo a ser incorporado ao protótipo, o qual pretende expandir o passo de identificação de verbos e substantivos com o de contabilização dos atributos de qualidade acima referidos.

4. CARACTERIZAÇÃO DO DOMÍNIO DA APLICAÇÃO

O enfoque adotado para implementação do protótipo de uma ferramenta CASE que suportasse a proposta UML consistiu primeiramente de uma análise do escopo da proposta UML, objetivando identificar-se as principais áreas cobertas pela mesma. Nesta fase identificou-se 3 grandes áreas básicas, as quais consistem de: editor de casos de uso, analisador de verbos e substantivos, um editor de diagrama de classes e, finalmente, um editor de diagramas de seqüência.

Os casos de uso, conforme apresentado anteriormente, permitem a definição do escopo de um projeto; o analisador de verbos e substantivos permite que sejam identificados, a partir das descrições dos casos de uso, elementos candidatos a: classes, atributos e métodos, facilitando o processo inicial de criação de um diagrama de classes; o diagrama inicial deve ser passível de alterações (inclusões/exclusões) tarefa esta de responsabilidade do editor do diagrama de classes e, finalmente, o editor de diagramas de seqüência permite o detalhamento, no tempo, da troca de mensagens entre as classes identificadas previamente. Esta divisão pode ser visualizada na figura 2.

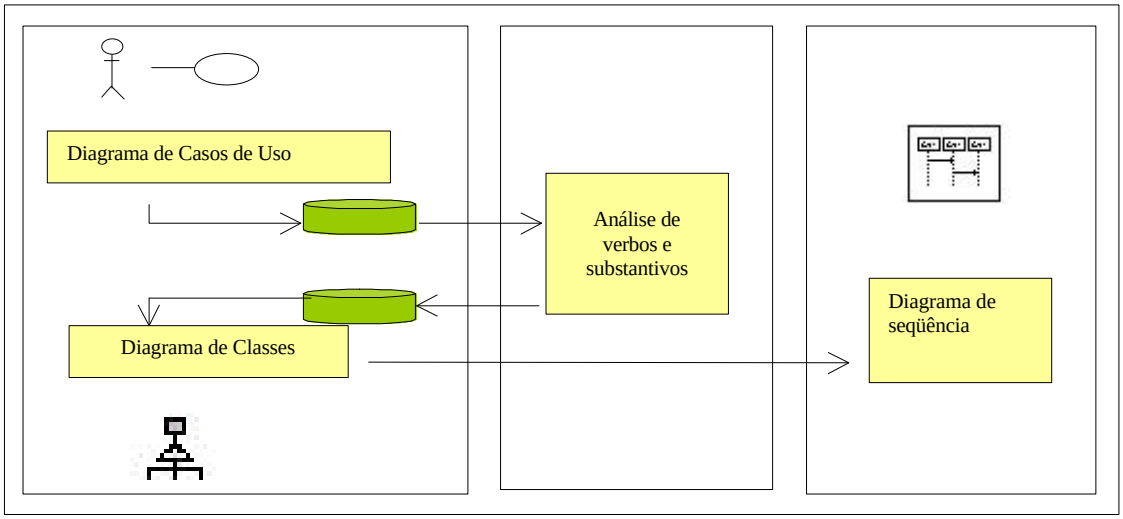

Figura 2 - Escopo dos trabalhos de conclusão de curso

Uma vez definido o escopo do projeto, identificou-se 3 alunos em fase de trabalho de conclusão de curso, aos quais coube o processo de estudo, definição dos sub-protótipos e implementação dos mesmos. Tendo em vista a complexidade do assunto, cada protótipo foi implementado na linguagem de programação e no ambiente de desenvolvimento em que o aluno tivesse maior domínio. Assim sendo, obteve-se os seguintes resultados:

a) um protótipo de editor de diagramas de use-cases e de diagramas de classes desenvolvido em ambiente Delphi da empresa Inprise Inc;
b) um protótipo de analisador de verbos e substantivos desenvolvido em java, utilizando o pacote jdk1.1.5 da empresa Sun Microsystems;
c) um protótipo de editor de diagramas de seqüência, desenvolvido em ambiente Visual Basic, da empresa Microsoft Corporation.

Deve-se observar que, não estabeleceu-se uma linguagem padrão para a implementação em função das restrições de tempo que os alunos possuem para desenvolver o trabalho de conclusão de curso.

4.1. DIAGRAMA DE CASOS DE USO

A figura 3 apresenta a tela do editor de casos de uso. Conforme pode-se verificar, a notação suportada pelo editor é aquela definida na proposta UML, ou seja: é possível a definição de atores e cenários. Para cada cenário, descrevem-se as atividades desenvolvidas naquele contexto. Além disso, o protótipo suporta os 3 tipos de relacionamentos entre os elementos de um diagrama deste tipo, quais sejam: *comunicates*, *uses* e *extends*. Sendo uma ferramenta acadêmica, ela dispõe de uma facilidade de help que orienta o usuário no processo de descrição de um cenário.

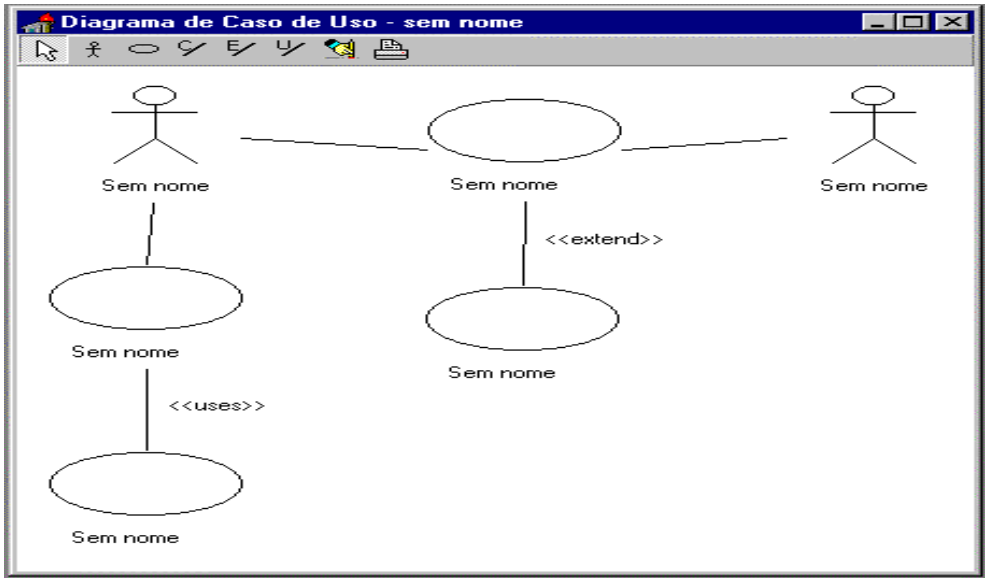

Figura 3 - Editor de diagrama de casos de uso

4.2. ANÁLISE DE VERBOS E SUBSTANTIVOS

A etapa seguinte após definidos os cenários, é a exportação das informações para o protótipo do módulo de análise de verbos e substantivos. Esta etapa é útil , uma vez que ela procura automatizar o processo de busca de verbos e substantivos (nas informações de descrição dos diagramas de casos de uso) candidatos a tornarem-se classes, atributos ou operações no diagrama de classes.

Na versão atual do protótipo, após a identificação de verbos e substantivos conhecidos, apresenta-se uma lista de palavras desconhecidas para que o usuário classifique cada palavra. Desta forma o sistema aumenta sua base de conhecimento.

Depois disso, apresenta-se a relação dos substantivos encontrados, para que o usuário analise e marque aqueles que podem ser candidatos à classe ou atributo de classe. Da mesma forma, apresenta-se uma relação dos verbos encontrados, para que o usuário identifique quais deles podem ser associados como responsabilidade de que classe, ou seja, quais verbos serão apropriados como métodos de que classes. Encerrada a fase de análise de verbos e substantivos, a base classificada é exportada para que o módulo editor de diagrama de classes possa construir o modelo a partir daquelas informações.

4.3. DIAGRAMA DE CLASSES

O editor de diagrama de classes importa as informações geradas pelo módulo analisador de verbos e substantivos e apresenta uma tela (fig 4a) para que o usuário classifique as informações.

Com base nestas informações é gerado o diagrama de classes conforme apresentado na figura 4b. Cabe salientar que este módulo suporta a criação dos seguintes elementos: classe, package, interface, classe template, bound element, relacionamentos entre classes e interfaces e entre classes e elementos de ligação, generalização, associação binária, associação binária reflexiva, composição, e finalmente de dependência.

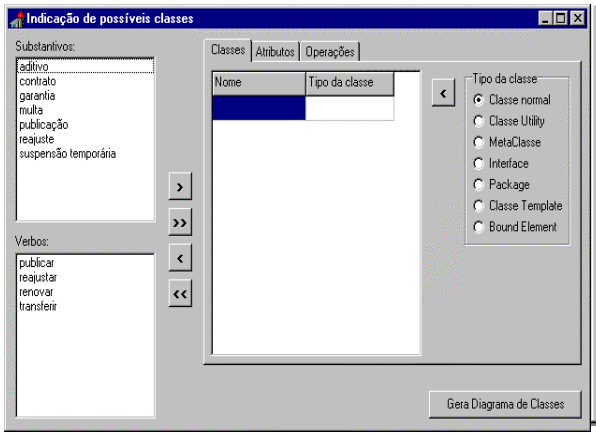

Fig.4a – Importação de verbos e substantivos

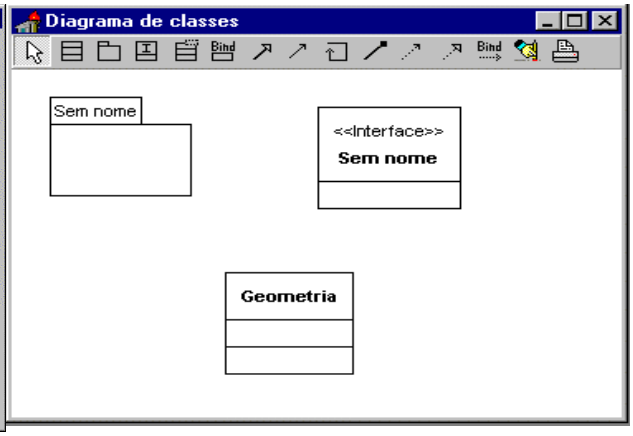

Fig.4b- Editor de diagrama de classes

4.4. DIAGRAMA DE SEQÜÊNCIA

O diagrama de seqüência tem por objetivo representar graficamente as interações entre os diversos elementos do projeto dentro de um determinado cenário. As informações geradas pelo módulo de diagrama de classes pode ser exportado para o módulo de diagrama de seqüência. Apesar disto, é possível a criação do diagrama de seqüência independentemente de já haver sido gerado o diagrama de classes (segundo a filosofia de projeto incremental, baseada na proposta UML, cada diagrama é independente de seqüência, ou seja, não é necessária a definição completa do diagrama de classes antes de iniciar-se os vários diagramas de

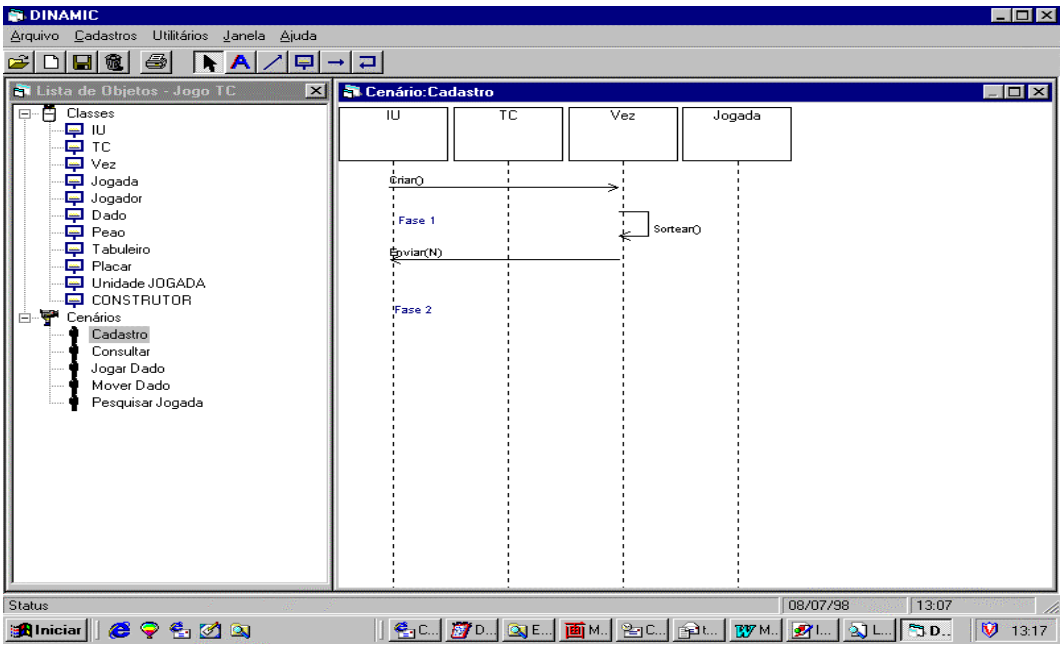

Fig 5 - Editor de Diagrama de seqüência

seqüência). A figura 5 apresenta um exemplo de diagrama de seqüência suportado por este módulo.

5. CONCLUSÃO

O presente projeto caracteriza-se por tratar de um assunto atual e de grande impacto na área. A relevância reside no fato de que todo o mercado está convergindo para a adoção da proposta UML, a qual certamente, em curto prazo, virá a ser uma das mais importantes contribuições para a área de engenharia de software e desenvolvimento do sistema.

Assim sendo, os problemas inicialmente apresentados puderam ser superados em parte (tendo em vista que o projeto ainda encontra-se em andamento) na medida em que:

a) não há vínculo com alguma ferramenta comercial específica, mas no entanto, os principais conceitos são suportados pelos módulos ora implementados;
b) não há mais necessidade de reinstalação do software a cada x dias, como é o caso de cópias de avaliação das ferramentas disponíveis no mercado;
c) o grau de complexidade e abrangência são suficientes para o escopo de um ambiente acadêmico como é o caso.
d) a possibilidade de agregação de novas facilidades desenvolvidas in-house permite a incorporação de novas facilidades o que não é possível com ferramentas comerciais;
e) a agregação de um módulo de análise de verbos e substantivos por si só, já caracteriza um avanço em termos de qualidade da ferramenta uma vez que, em algumas ferramentas comerciais é sugerido que a fase de descrição dos diagramas de casos de uso seja feita utilizando-se um editor de textos qualquer, ou seja, fora do escopo da ferramenta;
f) a agregação de um módulo de análise qualitativa das descrições de casos de uso ora em andamento vai aumentar a qualidade das especificações desenvolvidas pelos alunos, contribuindo para a formação de uma consciência no sentido de investir mais tempo na fase de especificação para evitar problemas posteriores já na fase de implementação

6. BIBLIOGRAFIA

[1] Barbieri, Carlos. **A Unificação dos Métodos de Orientação a Objetos**. Revista Developers Magazine, Rio de Janeiro: Axcel Books do Brasil Editora Ltda, n.12, p. 20-24, ago. 1997

[2] Barbieri, Carlos. **O Método de Análise OO Unified Modeling Language.** Revista Developers Magazine, Rio de Janeiro: Axcel Books do Brasil Editora Ltda, n.16, p. 32-33, dez. 1997.

[3] IEEE Std 830-1993, **Recommended Practice for Software Requirements Specifications**, December 2, 1993.

[4] Kitchenham, B.Pfleeger,S.L. **Software Quality: The elusive target**. IEEE Software, Vol.13. No.1,January 1996,pp.12-21.

[5] Lawrence,A.W. Hyatt,L.E.**Developing a Successful Metrics Program**.8th Annual Software Technology Conference. Utah - April, 1996.

[6] Lawrence,A.W. Hyatt,L.E.**Software Quality Metrics for OO Environments.** Crosstalk Jounal. April 1997.

[7] Muller, Pierre-Alain. **Instant UML**. Paris : Editions Eyrolles, 1997.

[8] Rational, Software Corporation. **Unified Modeling Language, version 1.1 1997**. Disponível em http://www.rational.com/uml (Abr 15 19:20 1998).

[9] Rosemberg,L., Hammer,T. and Shaw,J., **Testing Metrics for Requirement Quality**. 2nd Quality Week Europe. Belgium - Nov 1998

[10] Rosemberg,L., Hammer,T. and Shaw,J.,**Software Metrics and Reliability.** 9th International Symposium on Software Reliability Engineering. Germany - Nov 1998.

[11] Rumbaugh, James; BLAHA, Michael; PREMERLANI, Willian; EDDY, Frederick; LORENSEN, Willian. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro : Editora Campus, 1994.

[12] Wilson, W.M., Rosenberg,L.H and Hyatt,L.E. **Automated Analysis of Requirement Specifications**. International Conference on Software Engineering (IASTED), Boston, MA - May 1997.

[13] Wilson, W.M Rosenberg,L.H. Hyatt,L.E**. Automated Analysis of Requirement Specifications.** International Conference on Software Engineering (IASTED). Boston, MA - May 1997

[14] Wilson, W.M **.Writing Effective Requirements Specifications**. Software Technology Conference