Avaliação de Ferramentas para Desenvolvimento Orientado a Objetos com UML

Alexandre Nunes Costa[1], Vera M. B. Werneck[1], Marcio Francisco Campos[2]
[1]*UERJ - Universidade do Estado do Rio de Janeiro*
Departamento de Informática e Ciência da Computação
vera@ime.uerj.br
[2] *FAETEC/IST-RIO - Instituto Superior de Tecnologia do Rio de Janeiro*
camposmf@gmail.com

Resumo

A orientação a objetos é uma realidade e veio como solução para construir sistemas computacionais. O desenvolvimento de sistemas com qualidade tem a modelagem como uma tarefa complexa sendo preponderante o uso de um método e ferramentas CASE (Computer Aided Software Engineering) que apóiem a construções dos diferentes diagramas e artefatos. UML (Unified Modeling Language) é a linguagem padrão adotada para orientação a objetos. Neste contexto este trabalho apresenta uma avaliação de quatro ferramentas CASE gratuitas que estão disponíveis no mercado, a fim de descobrir a qualidade destes softwares e o quanto eles podem ajudar na criação rápida e eficiente de softwares utilizando a UML.

1. Introdução

Construir sistemas computacionais não é uma tarefa fácil, sendo um dos temas centrais a representação do sistema nos diversos níveis de abstração. Nesse aspecto, existem algumas abordagens para a modelagem: paradigma estruturado, orientado a objetos entre outros. Em qualquer um destes paradigmas um problema é o da padronização, o fato de não existir uma notação padrão e realmente eficaz que possa abranger qualquer tipo de aplicação.

As metodologias estruturadas existentes possuem suas próprias notações (símbolos usados para projetar modelos orientados a objetos), processos e ferramentas. Assim, a escolha de um método a ser utilizado se torna uma decisão importante, e leva a discussões e debates sobre qual o melhor método, o mais avançado e adequado para ser utilizado em um projeto específico.

Com a junção das três mais conceituadas metodologias (Booch de Grady, OOSE de Jacobson e o OMT de Rumbaugh) de modelagem orientadas a objetos (OO) criou-se uma notação padrão UML (Unified Modeling Language), aproveitando o que havia de melhor em cada uma delas adicionando conceitos e visões da linguagem. Assim, a UML é uma padronização de modelagem orientada a objetos que procura abranger a modelagem de qualquer sistema, de uma forma correta, consistente e compreensível.

As ferramentas CASE podem ser definidas como sistema computacional composto de ferramentas que suportam a automação do processo de desenvolvimento de software e permite o uso efetivo dos princípios e práticas gerais de engenharia de software. Normalmente, essas ferramentas apóiam alguns métodos ou linguagens de métodos.

Com a difusão da Orientação a objetos e do UML surgiram no mercado várias ferramentas que apóiam o desenvolvimento com UML, sendo algumas gratuitas. Com esse foco surgiu esse estudo que tem como objetivo fornecer uma visão geral e avaliação de algumas ferramentas livres disponíveis para a UML e assim facilitar o uso da UML principalmente a nível acadêmico.

Este artigo está dividido em cinco seções. Na seção 2 é apresentada uma visão geral do UML com seus elementos, visões e diagramas. A metodologia de avaliação é apresentada na seção 3. A seguir na seção 4 é fornecida uma visão geral das ferramentas Dia, ArgoUML, Umbrello e Jude que serão avaliadas na seção 5. Na seção 6, descreve-se as conclusões deste trabalho.

2. UML

UML é uma linguagem padrão para a elaboração da estrutura de projetos de software. Ela pode ser utilizada para visualizar, especificar, construir e documentar sistemas complexos de software. A UML é adequada para modelar Sistemas de Informação Corporativos, aplicações baseadas em Web, e Sistemas Complexos de Tempo Real [1].

A UML não é um método, pois, geralmente, um método possui uma linguagem de modelagem e um processo [2]. A UML é apenas uma linguagem, assim, esta pode ser um componente de um método voltado para o desenvolvimento de software. A UML é independente de processo, mas pode ser usada em um processo orientado a casos de uso, centrado na arquitetura, iterativo e incremental [1].

Vários métodos de análise e projeto orientados a objetos surgiram no final dos anos 80 e início dos anos 90, sendo que estes métodos eram muito semelhantes,

diferenciando-se apenas em pequenos detalhes. Algumas das principais metodologias populares nos anos 90 foram: Booch, OMT (Object Modelling Technique), OOSE/Objectory.

Cada um destes métodos possui sua própria notação (seus próprios símbolos para representar modelos orientados a objetos), processos (que atividades são desenvolvidas em diferentes partes do desenvolvimento), e ferramentas (as ferramentas CASE que suportavam cada uma destas notações e processos).

Diante desta diversidade de conceitos, "os três amigos", Grady Booch, James Rumbaugh e Ivar Jacobson [1] decidiram criar uma Linguagem de Modelagem Unificada. Eles disponibilizaram inúmeras versões preliminares da UML para a comunidade de desenvolvedores que ajudou com novas idéias, melhorando ainda mais a linguagem. Assim os metodologistas Rumbaugh e Booch unificaram seus métodos com o intuito de propor a todos os outros que adotassem esta metodologia unificada. Esta ação foi aceita após muita discussão com os outros metodologistas. Assim, surgiu a versão 0.8 do Método Unificado de Rumbaugh e Booch.

A seguir, a Rational Software comprou a Objectory (ferramenta CASE), e Jacobson uniu-se à equipe. Logo após, o método passou a se chamar UML, como é conhecido atualmente. Os outros metodologistas não aceitaram que somente a UML formasse a metodologia unificada, assim, a OMG, de forma a estabelecer um padrão, abriu solicitação para o envio de propostas para uma modelagem unificada. A Rational enviou, então, sua versão 1.0 da UML. Após um período de discussões, a OMG adotou a versão 1.1 como seu padrão oficial. Depois disso, surgiram as versões 1.2 (melhorias na aparência), a versão 1.3, 1.4 e 1.5 (melhorias mais significativas) [2]. Atualmente a UML está na versão 2.0.

Assim os objetivos da UML são: a modelagem de sistemas (não apenas de software) usando os conceitos da orientação a objetos; estabelecer uma união fazendo com que métodos conceituais sejam também executáveis; criar uma linguagem de modelagem usável tanto pelo homem quanto pela máquina.

A UML tem sido dominante, como a linguagem de modelagem comum usada nas indústrias. Ela é totalmente baseada em conceitos e padrões extensivamente utilizados, provenientes das metodologias, existentes anteriormente como mostra a Figura 1, e também é muito bem documentada com toda a especificação da semântica da linguagem representada em meta-modelo.

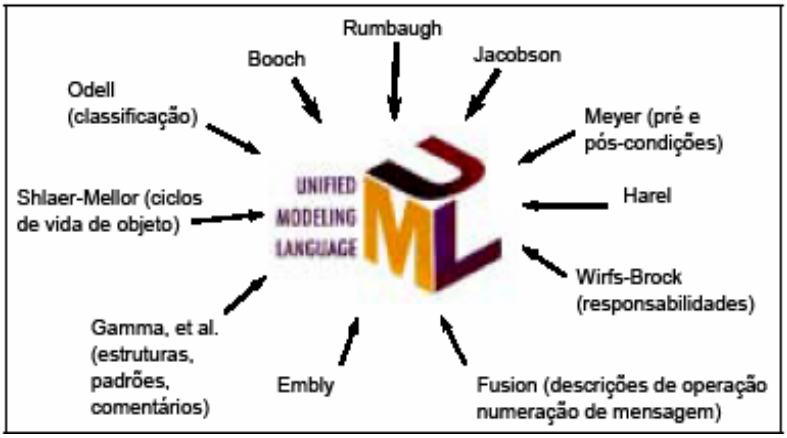

Figura 1: Contribuições para a UML. Fonte: www.omg.org (2004)

2.1. Visão Geral da UML

Dentre as cinco fases do desenvolvimento de sistemas em UML, as fases de análise de requisitos, análise e projeto utilizam-se em seu desenvolvimento cinco tipos de visões, treze tipos de diagramas e vários modelos de elementos que serão utilizados na criação dos diagramas e mecanismos gerais que todos em conjunto especificam e exemplificam a definição do sistema, tanto a definição no que diz respeito à funcionalidade estática e dinâmica do desenvolvimento de um sistema.

As partes que compõem a UML são Visões, Modelos de Elementos, Mecanismos Gerais e Diagramas.

As Visões mostram diferentes aspectos do sistema que está sendo modelado. A visão não é um gráfico, mas uma abstração consistindo em uma série de diagramas. Definindo um número de visões, cada uma mostrará aspectos particulares do sistema, dando enfoque a ângulos e níveis de abstrações diferentes e uma figura completa do sistema poderá ser construída. As visões também podem servir de ligação entre a linguagem de modelagem e o método/processo de desenvolvimento escolhido.

Os conceitos usados nos diagramas são Modelos de Elementos que representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamentos entre classes incluindo associações, dependências e heranças.

Os Mecanismos Gerais provêm comentários suplementares, informações, ou semântica sobre os elementos que compõem os modelos; eles provêm também mecanismos de extensão para adaptar ou estender a UML para um método/processo, organização ou usuário específico.

Os Diagramas são os gráficos que descrevem o conteúdo em uma visão. UML possui treze tipos de diagramas que são usados em combinação para prover todas as visões do sistema.

Um sistema é composto por diversos aspectos: funcional (que é sua estrutura estática e suas interações dinâmicas), não funcional (requisitos de tempo, confiabilidade, desenvolvimento, etc.) e aspectos organizacionais (organização do trabalho, mapeamento dos módulos de código, etc.). Desta forma o sistema é descrito em um certo número de visões, cada uma representando uma projeção da descrição completa e mostrando aspectos particulares do sistema.

Cada visão é descrita por um número de diagramas que contém informações que dão ênfase aos aspectos particulares do sistema. Existem em alguns casos uma certa sobreposição entre os diagramas, o que significa que um deste pode fazer parte de mais de uma visão. Os diagramas que compõem as visões contêm os modelos de elementos do sistema. As visões que compõem um sistema são [1]:

- Visão de casos de uso: Descreve a funcionalidade do sistema desempenhada pelos atores externos do sistema (usuários). A visão de casos de uso é central, já que seu conteúdo é base do desenvolvimento das outras visões do sistema. Essa visão é montada sobre os diagramas de casos de uso e eventualmente diagramas de atividade.
- Visão Lógica: Descreve como a funcionalidade do sistema será implementada. É feita principalmente pelos analistas e desenvolvedores. Em contraste com a visão use-case, a visão lógica observa e estuda o sistema internamente. Ela descreve e especifica a estrutura estática do sistema (classes, objetos, e relacionamentos) e as colaborações dinâmicas quando os objetos enviarem mensagens uns para os outros para realizarem as funções do sistema. Propriedades como persistência e concorrência são definidas nesta fase, bem como as interfaces e as estruturas de classes. A estrutura estática é descrita pelos diagramas de classes e objetos. A modelagem dinâmica é descrita pelos diagramas de estado, seqüência, colaboração e atividade.
- Visão de Componentes: É uma descrição da implementação dos módulos e suas dependências. É principalmente executado por desenvolvedores, e consiste nos componentes dos diagramas.
- Visão de concorrência: Trata a divisão do sistema em processos e processadores. Este aspecto, que é uma propriedade não funcional do sistema, permite uma melhor utilização do ambiente onde o sistema se encontrará, se o mesmo possui execuções paralelas, e se existe dentro do sistema um gerenciamento de eventos assíncronos. Uma vez dividido o sistema em linhas de execução de processos concorrentes (*threads*), esta visão de concorrência deverá mostrar como se dá a comunicação e a concorrência destes *threads*. A visão de concorrência é suportada pelos diagramas dinâmicos, que são os diagramas de estado, seqüência, colaboração e atividade, e pelos diagramas de implementação, que são os diagramas de componente e execução.
- Visão de Organização: Finalmente, a visão de organização mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si. Esta visão será executada pelos desenvolvedores, integradores e testadores, e será representada pelo diagrama de execução.

A UML define um modelo e uma notação. O modelo auxilia na visualização do sistema, permite especificar a estrutura e comportamento do sistema, proporciona um guia para construção do sistema e documenta as decisões tomadas [1]. A notação é a visão obtida por meio de modelos e diagramas do sistema, pois define o modo de representar itens e conceitos. Suas notações são mais intuitivas do que definidas formalmente, mas, apesar disto, são consideradas valiosas [2].

Os diagramas auxiliam o desenvolvedor no momento da análise e no projeto. O usuário está interessado em um software fácil de usar e que execute o que se propõe a fazer. Assim, a UML é importante para ajudar o programador a escrever o código, sendo também muito importante para facilitar a comunicação entre desenvolvedor e usuário. Por meio dela, é possível transmitir alguns conceitos de forma clara, sem a imprecisão inerente a uma linguagem natural, nem o detalhamento e precisão do código (linguagem que, provavelmente, o usuário não conhece) [2].

A UML possibilita uma visão geral do sistema, sem que o desenvolvedor use muito tempo para analisar um diagrama, pois o diagrama vai salientar apenas os detalhes relevantes do projeto. Esta característica é muito importante para uma equipe que trabalha em grandes projetos, pois facilita a comunicação entre a equipe e o entendimento do sistema [2].

UML define treze tipos de diagramas e os diagramas estão divididos em dois grupos: seis diagramas que representam a estrutura estática da aplicação e sete que representam diferentes aspectos da conduta dinâmica da aplicação.

Os Diagramas Estáticos são: Diagrama de Classes, Diagrama de Estruturas Compostas, Diagrama de Objetos, Diagrama de Componentes, Diagrama de Instalação e Diagrama de Pacotes.

Os Diagramas Dinâmicos são: Diagrama de Casos de Uso, Diagrama de Atividades, Diagrama de Máquina de Estados, Diagrama de Seqüências, Diagrama de Comunicação, Diagrama de Visão Geral de Iteração e Diagrama de Sincronização.

Os diagramas de Estruturas Compostas, de Visão Geral de Iteração e Diagrama de Sincronização são novidades na UML 2.0.

O Diagrama de Casos de Uso exibe ações do sistema e atores (tipo especial de classe) e seus relacionamentos. Este diagrama abrange a visão estática do sistema, e é importante para organizar e modelar os comportamentos do sistema [1].

O Diagrama de Classes mostra o conjunto de classes, interfaces, colaborações e relacionamento. Geralmente são usados para modelar sistemas Orientados a Objeto, abrangendo uma visão estática do sistema [1]. Este diagrama descreve os tipos de objetos e de relacionamentos estático do sistema. Além disso, o diagrama de classes exibe atributos e operações de uma classe [2].

O Diagrama de Objetos exibe objetos e seus relacionamentos, simbolizando retratos estáticos das instâncias de itens que constam nos diagramas de classes. Este diagrama modela instâncias das classes do sistema em certa etapa e momento de execução. É similar ao diagrama de classes, pois, utiliza quase todas as notações usadas no diagrama de classes, seu intuito é mostrar as instâncias das classes de um sistema em determinado momento de execução desse sistema [2], [3].

O Diagrama de Componentes exibe as organizações e dependências que existem em um conjunto de componentes. Abrange a visão estática de um sistema e tem relação com os diagramas de classes, já que os componentes são mapeados para uma ou mais classes, interfaces ou colaborações [1]. Este diagrama exibe os tipos de elementos que farão parte do sistema, identificando as dependências entre esses componentes.

O Diagrama de Instalação mostra como os nós de processamento estão configurados em tempo de execução e componentes nele existentes. Este diagrama exibe a visão estática da arquitetura do sistema. Está relacionado aos diagramas de componentes, pois, em geral, um nó inclui um ou mais componentes [1]. Este diagrama exibe o relacionamento físico entre componentes de software e de hardware no sistema implementado. Este diagrama mostra como os componentes e objetos se movimentam em um sistema distribuído [2]. Representa a configuração e arquitetura de um sistema, com a possibilidade de representar componentes de hardware também. Segundo Fowler [2], este diagrama pode representar a estrutura da plataforma em que o sistema funcionará e qualquer dispositivo como um servidor, um terminal, etc.

O Diagrama de Atividade exibe o fluxo de uma atividade para outra no sistema. Abrange a visão dinâmica do sistema e é importante para modelar a função de um sistema, enfatizando o fluxo de controle entre os objetos. Descreve as ações de um estado, como, por exemplo, a execução de um método de uma classe. Este diagrama descreve em que seqüência as atividades irão ocorrer; permite modelar atividades condicionais e em paralelo. Ele se assemelha ao diagrama de estados. Uma das diferenças é quando existe comportamento condicional, este é delineado por Desvios e Intercalações [1].

Um Diagrama de Seqüências é um diagrama de interação cuja ênfase está na ordenação temporal das mensagens [1]. No diagrama de seqüências pode-se ver o uso de um objeto, desde sua criação até sua destruição. Isto é representado por uma linha de vida do objeto. Objetos podem trocar mensagens, ou efetuar uma auto chamada (uma mensagem que um objeto manda para si mesmo).

O Diagrama de Comunicação é um diagrama de interação com ênfase na organização estrutural dos objetos que enviam e recebem mensagens [1]. Diagramas de Comunicação e de Seqüências são isomórficos, ou seja, pode-se transformar um diagrama de um tipo em um diagrama de outro tipo [1].

Um pacote é uma construção de agrupamento que permite pegar qualquer construção na UML e agrupar seus elementos em unidades de nível mais alto. Normalmente os pacotes são compostos de classes ou de outros pacotes de modo a obter uma estrutura hierárquica Os Diagramas de Pacote representam um mecanismo de agrupamento em tempo de compilação sendo útil para mostrar as dependências e controles [2].

O Diagrama de Estruturas Compostas apresenta uma novidade da UML 2.0 que é poder decompor uma classe em uma estrutura interna permitindo que um objeto complexo seja dividido em partes. A distinção entre pacotes e estruturas compostas é que o primeiro um agrupamento em tempo de compilação enquanto que as estruturas compostas são em tempo de execução. Parte dessa notação é usada no Digrama de Componentes, pois mostram os componentes e suas partes [2,3].

O Diagrama de Visão Geral de Iteração são uma mistura de diagrama de atividades e diagramas de seqüências mostrando os fluxos de controle nas interações [2].

O Diagrama de Sincronização é outra forma de diagrama de interação com foco nas restrições de temporização para um único ou vários objetos [2].

3. Metodologia de Avaliação

Avaliar a qualidade de um produto de software é verificar, através de técnicas e atividades operacionais, quanto os requisitos são atendidos. Tais requisitos, de uma maneira geral, são a expressão das necessidades, explicitados em termos quantitativos ou qualitativos, e têm por objetivo

definir as características de um software, a fim de permitir o exame de seu entendimento [4].

A metodologia de avaliação deste trabalho utilizou a norma brasileira NBR 13596 que é uma tradução do texto-base da norma ISO/IEC 9126. Esta norma define seis Características de Qualidade de Software (Tabela 1). Essas características e subcaracterísticas se preocupam também em apresentar um modelo de qualidade de produto de software.

A qualidade em uso pode ser medida através da operação do Produto Final em condição de uso normal ou simulada, verificando-se a existência e nível das Características e Subcaracterísticas definidos na Norma ISO/IEC 9126 [5].

Tabela 1: Quadro de Características para Avaliação de Software

Característica	Subcaracterística	Pergunta chave para a subcaracterística
Funcionalidade (satisfaz as necessidades?)	Adequação	Propõe-se a fazer o que é apropriado?
	Acurácia	Faz o que foi proposto de forma correta?
	Interoperabilidade	Interage com os sistemas especificados?
	Conformidade	Está de acordo com as normas, leis, etc.?
	Segurança de acesso	Evita acesso não autorizado aos dados?
Confiabilidade (é imune a falhas?)	Maturidade	Com que freqüência apresenta falhas?
	Tolerância a falhas	Ocorrendo falhas, como ele reage?
	Recuperabilidade	É capaz de recuperar dados em caso de falha?
Usabilidade (é fácil de usar?)	Inteligibilidade	É fácil entender o conceito e a aplicação?
	Apreensibilidade	É fácil aprender a usar?
	Operacionalidade	É fácil de operar e controlar?
Eficiência (é rápido e "enxuto"?)	Tempo	Qual é o tempo de resposta, a velocidade de execução?
	Recursos	Quanto recurso usa? Durante quanto tempo?
Manutenibilidade (é fácil de modificar?)	Analisabilidade	É fácil de encontrar uma falha, quando ocorre?
	Modificabilidade	É fácil modificar e adaptar?
	Estabilidade	Há grande risco quando se faz alterações?
	Testabilidade	É fácil testar quando se faz alterações?
Portabilidade (é fácil de usar em outro ambiente?)	Adaptabilidade	É fácil adaptar a outros ambientes?
	Capacidade para ser instalado	É fácil instalar em outros ambientes?
	Conformidade	Está de acordo com padrões de portabilidade?
	Capacidade para substituir	É fácil usar para substituir outro?

Medição é o ato de aplicar as medições escolhidas ao produto de software, onde o resultado é dado em valores nas escalas das medições; Pontuação é o processo técnico de medição da qualidade, onde o nível de pontuação é determinado a partir do valor medido; Julgamento é a emissão de um juízo sobre a qualidade, no qual um conjunto de níveis pontuados é sintetizado.

Assim foram selecionadas as medidas a serem usadas no software e o método adotado nesta avaliação foi a utilização de um "checklist" onde cada componente de software, com seus respectivos atributos, é desdobrado em perguntas e itens. Com relação às perguntas incluídas no "checklist", consideramos que aquelas perguntas são as proposições lógicas sobre um atributo a ser conferido na avaliação. Cada proposição que se refira a um atributo foi a mais objetiva possível envolvendo somente uma característica de qualidade. As principais respostas para a pergunta são: "S" (Sim) para as proposições verdadeiras; "N" (Não) para as proposições falsas.

Selecionadas as medidas, partimos para a pontuação destas onde as medidas são perguntas que têm como resposta um tipo de conceito. Um quadro para cada tipo de respostas foi feito com um valor numérico. Cada atributo teve um valor numérico. Este valor foi uma Média entre zero (0) e um (1).

O Julgamento significa interpretar resultados medidos. É importante que todos os atributos tenham valores semelhantes, até mesmo aquele composto de vários itens, fato este que normalmente acontece quando o atributo é subdividido em itens podendo, assim, ser conferido de um modo mais claro e objetivo. Conseqüentemente, em um atributo composto por mais de um item, os valores numéricos destes itens assumirão valores proporcionais e estão entre zero e um dependendo de quantos itens ele seja composto. É necessário que a média destes itens, ou melhor, os valores numéricos estejam entre zero e um.

Após, foram realizadas as conclusões sobre qualidade e todo resultado das medidas foi tabulado para a escala (0,1), nos quais 0 (característica ausente) aponta para o pior resultado possível, considerando que 1(característica presente) aponta para o melhor. Ao final de cada avaliação, as notas são somadas. Assim são encontradas as pontuações de cada software para aquela avaliação. A quantidade de características avaliadas é a meta a ser atingida 100%.

4. Ferramentas CASE

As ferramentas Dia, ArgoUml, Unbrello e Judes foram escolhidas nesse trabalho para serem analisadas, pois são ferramentas freeware (livres), estão disponíveis para download nos sites de seus desenvolvedores e são conhecidos no mercado. A seguir uma descrição sucinta de cada ferramenta será fornecida, bem como a tela de interface dando uma visão geral da ferramenta.

4.1. Dia

Esta ferramenta está disponível em

http://www.gnome.org/projects/dia/. Dia é um software de criação de diagramas baseado no gtk+ e distribuído sob a licença GPL. Dia inspira - se no programa Windows "VISIO" da Microsoft, embora mais orientados para diagramas informais para uso ocasional. Pode ser usado para desenhar diferentes tipos de diagramas. Conta, atualmente, com objetos especiais que ajudam a desenhar entidades de relacionamento de diagramas, diagramas UML, fluxogramas, diagramas de rede, e muitos outros diagramas. Também é possível adicionar suporte para novas formas de escrita de arquivos XML simples, usando um subconjunto de SVG para desenhar a forma [6].

Este software (Figura 2) é capaz de carregar e guardar diagramas para um formato XML personalizado (gzipped por padrão, para economizar espaço), pode exportar diagramas para uma série de formatos, incluindo EPS, SVG, XFIG, WMF e PNG, e pode imprimir diagramas (incluindo aquelas que ocupam várias páginas) [6].

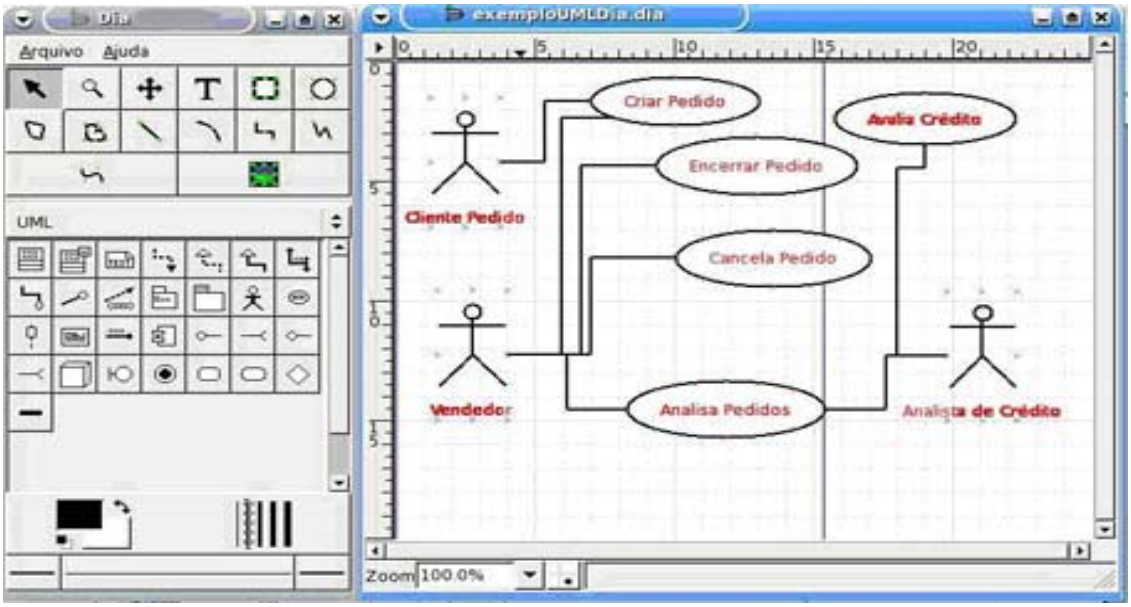

Figura 2: Interface do programa DIA. Fonte: www.gnome.org (2008)

4.2. ArgoUML

Esta ferramenta está disponível em http://argouml.tigris.org/. Foi desenvolvido originalmente por comunidade de desenvolvedores de código livre Tigris vinculada a Universidade da Califórnia, Berkeley. Atualmente, a versão 0.24 do ArgoUML executa todos os tipos de diagrama da UML 1.4 (as versões do ArgoUML anteriores a 0.20 implementava somente os diagramas da UML 1.3). É escrito em Java e pode ser executado em computadores que possuem a plataforma Java 2 da Java 1.4 ou mais recente. Ele usa o arquivo aberto formatos XMI (XML Metadata Interchange) para informações modeladas e PGML (Precision Graphics Markup Language) para informações gráficas [ArgoUML, 2008].

Sua interface (Figura 3) é bem completa o que a torna um pouco complexa de manipular. Com o ArgoUML é possível desenhar e imprimir todos os diagramas da UML 1.4, gerar declarações de classes Java, exportar documentação para páginas Web em Java, gerar arquivos gráficos (gif), com auxilio de software de terceiros e possível gerar comandos SQL, é possível, também, fazer engenharia reversa (ele fornece uma estrutura modular da engenharia reversa de classes Java) e exportar dados para o padrão XMI (baseado no formato XML) [7].

O ArgoUML possui integração com o AndroMDA. O AndroMDA é um framework open source baseado em MDA (Model Driven Architecture). Ele utiliza modelos UML gerados por ferramentas CASE (padrão XMI) e uma série de plugins, chamados de cartuchos (cartridges) para realizar a geração de componentes customizados, ou seja, o código fonte do sistema [6].

4.3. Umbrello

Esta ferramenta está disponível em http://uml.sourceforge.net/. Umbrello é uma ferramenta gratuita de UML.

Ele é parte do KDE Desktop Environment, mas funciona também em outros ambientes. Umbrello tem uma interface limpa e consistente (Figura 4). Ele manipula todos os tipos de diagramas UML. Pode importar C + +, IDL, Pascal / Delphi, Ada, Python, Java, Perl, bem como código (através da utilização de uma ferramenta externa) e exportar para várias linguagens de programação. O arquivo de formato utilizado é baseado em XMI [8].

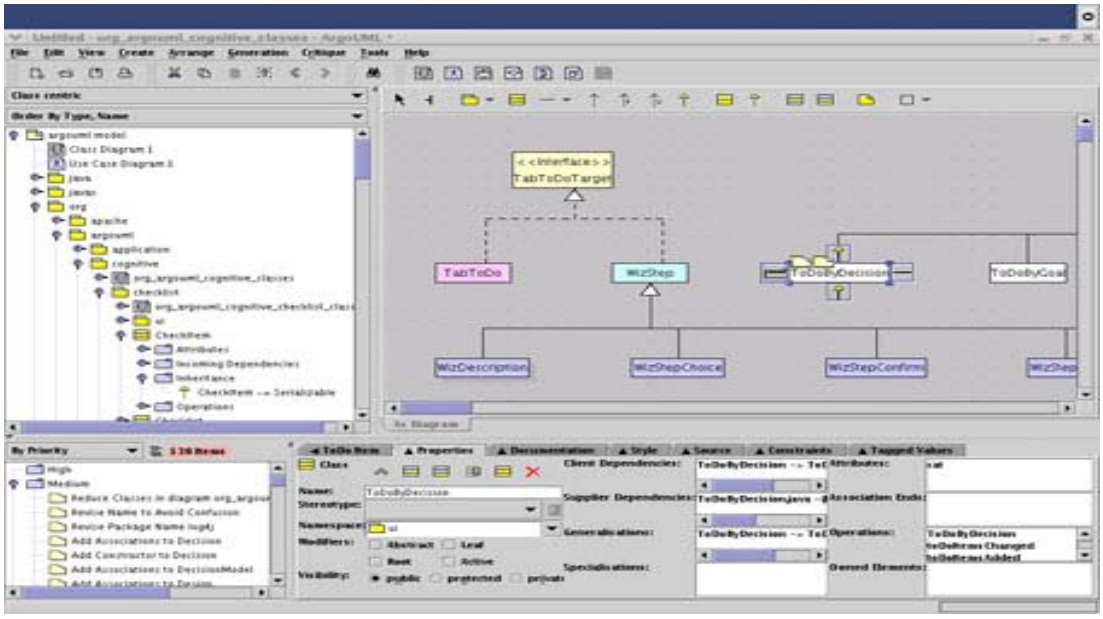

Figura 3: Interface do programa ArgoUML. Fonte: argouml.tigris.org (2008)

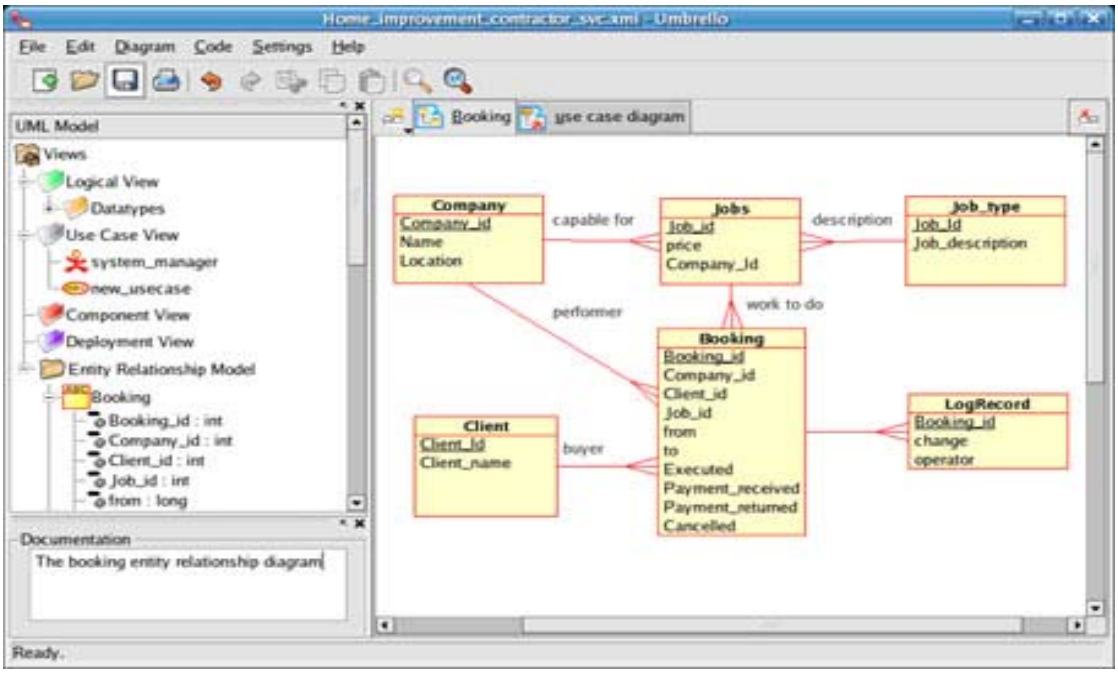

Figura 4: Interface do programa Umbrello. Fonte: http://upload.wikimedia.org (2008)

4.4. JUDE

Esta ferramenta está disponível em http://jude.change-vision.com/jude-web/index.html. JUDE é a sigla para Java and UML Developers Environment (Ambiente de Desenvolvimento Java e UML). O JUDE possui duas versões: Professional (versão comercial) e Community (versão livre, Figura 5), ambas são ferramentas case criadas pela empresa japonesa Change Vision. O JUDE Community oferece as seguintes funcionalidades [9]:

- Suporte a UML 1.4 (e parte da UML 2.0 somente na versão comercial);
- Diagramas de Classes, Casos de Uso, Seqüências, Colaboração, Estados, Atividades, Implantação e Componentes;
- Gera código Java a partir da modelagem;
- Importa Códigos Java para criar modelos.

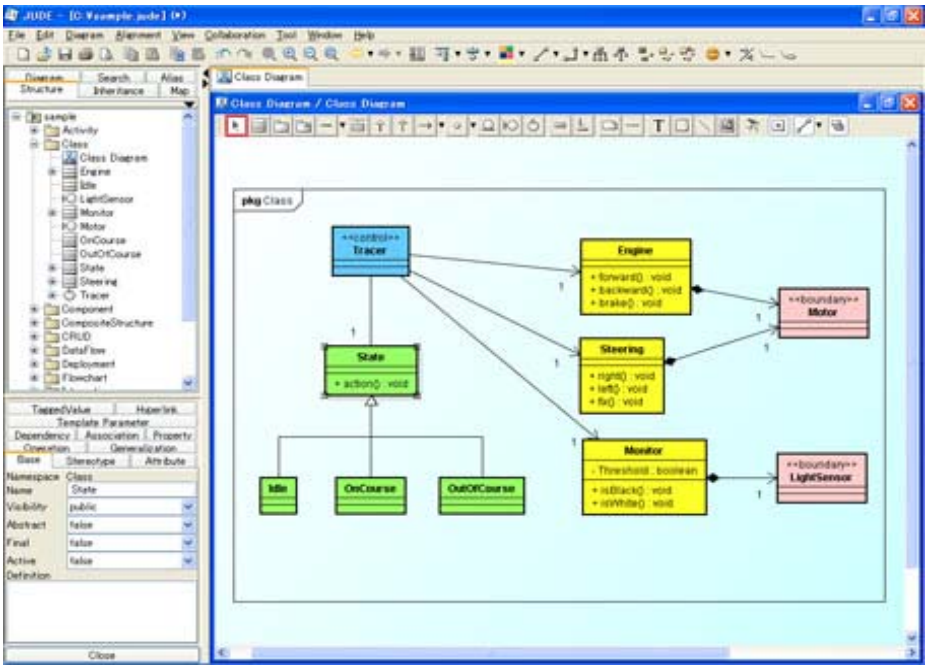

Figura 5: Tela do programa JUDE Fonte: jude.change-vision.com

5. Avaliação das Ferramentas

Para se realizar as avaliações, os softwares foram baixados e instalados num desktop com as seguintes configurações: Athlon XP 2600+, 1GB de RAM, rodando Windows XP Professional.

Foram executados testes de acordo com as características e subcaracterísticas nos softwares com base no desenvolvimento de um pequeno caso que serviu de padrão para todas as testar todas as ferramentas. O resultado do teste responde a pergunta chave e é a pontuação que o software recebe no quesito de acordo com a nossa análise e opinião pessoal sobre o produto.

5.1. Funcionalidade

A tabela 2 mostra o resultado do teste de funcionalidade e pode se destacar que nas subcaracteristicas:

- Adequação: todos os softwares fazem o propósito pelo qual eles foram elaborados, ou seja, os diagramas da UML.
- Acurácia: todos os softwares fazem corretamente os diagramas da UML.
- Interoperabilidade: todos os softwares interagem com os sistemas especificados pelo fabricante.
- Conformidade: o Dia não pontuou, pois os diagramas da UML não estão separados, estão todos juntos numa única aba.
- Segurança de acesso: O Umbrello não pontuou, pois nos testes feitos era possível alterar dados de um arquivo em uso por ele.

Tabela 2: Quadro de resultados de avaliação de funcionalidade

Subcaracterística	Pergunta chave para a subcaracterística	Dia	ArgoUML	Umbrello	Jude
Adequação	Propõe-se a fazer o que é apropriado?	Sim	Sim	Sim	Sim
Acurácia	Faz o que foi proposto de forma correta?	Sim	Sim	Sim	Sim
Interoperabilidade	Interage com os sistemas especificados?	Sim	Sim	Sim	Sim
Conformidade	Está de acordo com as normas, leis, etc.?	Não	Sim	Sim	Sim
Segurança de acesso	Evita acesso não autorizado aos dados?	Sim	Sim	Não	Sim
Total Parcial		4	5	4	5

5.2. Usabilidade

A tabela 3 mostra o resultado do teste de usabilidade e pode se destacar que nas subcaracteristicas:

- Inteligibilidade: é fácil entender conceito e a aplicação em todos os softwares.
- Apreensibilidade: é fácil aprender a usar todos os softwares.

- Operacionalidade: o ArgoUML não pontuou, pois por ser em Java era preciso um conhecimento maior sobre JAVA para ser operado.

Tabela 3: Quadro de resultados de avaliação de usabilidade.

Subcaracterística	Pergunta chave para a subcaracterística	Dia	ArgoUML	Umbrello	Jude
Inteligibilidade	É fácil entender o conceito e a aplicação?	Sim	Sim	Sim	Sim
Apreensibilidade	É fácil aprender a usar?	Sim	Sim	Sim	Sim
Operacionalidade	É fácil de operar e controlar?	Sim	Não	Sim	Sim
Total Parcial		3	2	3	3

5.3. Eficiência

A tabela 4 mostra o resultado do teste de eficiência e pode se descrever que nas subcaracteristicas avaliadas:

- Tempo: Todos foram extremamente velozes durante a execução.
- Recursos: todos ocupavam pouquíssimos recursos da máquina durante a execução.

Tabela 4: Quadro de resultados de avaliação de eficiência.

Subcaracterística	Pergunta chave para a subcaracterística	Dia	ArgoUML	Umbrello	Jude
Tempo	Qual é o tempo de resposta, a velocidade de execução?	Sim	Sim	Sim	Sim
Recursos	Quanto recurso usa? Durante quanto tempo?	Sim	Sim	Sim	Sim
Total Parcial		2	2	2	2

5.4. Portabilidade

A tabela 5 mostra o resultado do teste de portabilidade pode se destacar que nas subcaracteristicas:

- Adaptabilidade: o ArgoUML não pontuou, pois por ser em JAVA precisa que o outro ambiente possua JAVA também.
- Capacidade para ser instalado: o ArgoUML não pontuou, pois ele não é instalado e precisa do JAVA para se executado.
- Conformidade: o ArgoUML não pontuou, pois portar ele para outro ambiente é bastante complicado, já que ele não é instalável e necessita do JAVA para ser executado.
- Capacidade para substituir: todos os softwares são facilmente substituíveis, tanto para outras versões quanto para outro software.

Tabela 5: Quadro de resultados de avaliação de portabilidade.

Subcaracterística	Pergunta chave para a subcaracterística	Dia	ArgoUML	Umbrello	Jude
Adaptabilidade	É fácil adaptar a outros ambientes?	Sim	Não	Sim	Sim
Capacidade para ser instalado	É fácil instalar em outros ambientes?	Sim	Não	Sim	Sim
Conformidade	Está de acordo com padrões de portabilidade?	Sim	Não	Sim	Sim
Capacidade para substituir	É fácil usar para substituir outro?	Sim	Sim	Sim	Sim
Total Parcial		4	1	4	4

5.5. Confiabilidade e Manutenibilidade

A confiabilidade do software não pode ser mensurada, pois, nos testes feitos nas ferramentas nenhuma apresentou falhas ou "bugs" que sejam relevantes para a avaliação.

A manutenibilidade não se aplica à avaliação, já que ela está ligada ao desenvolvimento do software.

6. Conclusão

Dentre as ferramentas avaliadas, o Jude mostrou-se ser a melhor Ferramenta Case em relação às demais, com 14 pontos em todas as características avaliadas. A seguir vem Dia e o Umbrello com 13 pontos cada. E por último o ArgoUML com apenas 10 pontos.

As diferenças de pontuação, principalmente entre o Jude, Dia e Umbrello não são significativas, mas pela experiência de uso, todos os softwares atingiram seu propósito e possuem funcionalidades semelhantes, apenas o Umbrello se mostrou pouco portável, e prejudicado frente às outras ferramentas por causa deste quesito.

Sem dúvida alguma a UML facilita às grandes empresas de desenvolvimento de software uma maior comunicação e aproveitamento dos modelos desenvolvidos pelos seus vários analistas envolvidos no processo de produção de software, já que a linguagem que será utilizada por todos será a mesma, acabando assim com qualquer problema de interpretação e mau entendimento de modelos criados por outros desenvolvedores.

Os modelos criados hoje poderão ser facilmente analisados por futuras gerações de desenvolvedores

acabando com a diversidade de tipos de nomenclaturas de modelos.

Juntamente com as avaliações feitas neste trabalho, percebe-se que não são necessários grandes investimentos para pequenos projetos. As ferramentas CASE gratuitas disponíveis, estão bastante integradas com a UML, com o modelo MDA (Model Driven Architecture – Arquitetura Orientada a Modelo, definido pela OMG[10]), possibilitam a geração de código automático, cada vez se faz mais necessária, possibilitando mais velocidade e versatilidade aos projetos.

Referências

[1] BOOCH, G. Object-Oriented Analysis and Design with Applications. Addison Wesley, 2006.

[2] FOWLER, M. & SCOTT, Kendall. UML Distilled - Applying the Standard Object Modeling Language. Massachusetts: Addison-Wesley, 2004.

[3] OBJECT MANAGEMENT GROUP - Technology Adoptions - UML Documents. http://www.omg.org/techprocess/meetings/schedule/UML_RTF.html (Abr, 2005).

[4] TSUKUMO, A.N.; et AL. Qualidade de Softare: Visões de Produto e Processo de Software, VIII CITS – Conferência Internacional de Tecnologia de Software, Curitiba/PR, junho/1997.

[5] NBR ISO/IEC 9126-1, Engenharia de Software – Qualidade de Produto - Modelo de Qualidade, Associação Brasileira de Normas Técnicas, 2003.

[6] Dia – Disponível em http://www.gnome.org/projects/dia/ em Março de 2008.

[7] ArgoUML - Disponível em http://argouml.tigris.org/ em Março de 2008.

[8] UMBRELLO - Disponível em http://uml.sourceforge.net/ em Março de 2008.

[9] JUDE - Disponível em http://jude.change-vision.com/jude-web/index.html em Março de 2008.

[10] OBJECT MANAGEMENT GROUP – OMG Model Driven Architecture – disponível em www.omg/mda em Março de 2008.