Faculdade de Tecnologia SENAC RS
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Relatório de Projeto Parcial

Aluno : Camila San Martin Ayres
Professor Orientador: Rafael Jeffman
Semestre: 2011/2

Porto Alegre, 12 de setembro de 2011.

Sumário

1. Identificação

IDENTIFICAÇÃO DO ALUNO

ALUNO: Camila San Martin Ayres
ENDEREÇO RESIDENCIAL:Rua Riachuelo 925/403
BAIRRO: Centro CEP: 90010-270
CIDADE: Porto Alegre UF: RS
TELEFONE(S): (51) 91434791 / 3226-6987
E-MAIL(S): smayres@gmail.com

IDENTIFICAÇÃO DO ORIENTADOR

PROFESSOR: Rafael Jeffman
E-MAIL(S): rafael.jeffman@gmail.com

2. Histórico de Versões

	Versão	Autor	Descrição	Data
1		Camila Ayres	Rascunho do Relatório	01/06/2011
2		Camila Ayres	Melhorias	04/06/2011
3		Camila Ayres	Protótipo TCCI	07/06/2011
4		Camila Ayres	Diagramas	08/06/2011
5		Camila Ayres	Revisão Final de Texto	09/06/2011
6		Camila Ayres	Versão Final	10/06/2011

3. Projeto

3.1. Titulo do Projeto

Refatoração: adaptando o Umbrello para QGraphicsView.

3.2. Professor Orientador

Rafael Jeffman

3.3. Apresentação Geral do Projeto

Software Livre e Open Source (FLOSS – Free Libre Open Source Software) descrevem o conceito de software com licença *copyleft*[1]. Comparado aos Copyright, copyleft dá a quem recebe o software direitos adicionais[2]. Este trabalho fará uso do termo "Software Livre" para abranger todos os aspectos de FLOSS que enfatizam a filosofia e os aspectos do desenvolvimento colaborativo, bem como o aspecto legal e prático[3].

A licença copyleft bem como a GNU Public License garantem principalmente que: "A liberdade de redistribuir cópias deve incluir formas binárias ou executáveis do programa, bem como código-fonte, para ambas as versões modificadas." (GNU, 2010) O código-fonte, uma vez publicado sob a licença GNU Public License, estará sempre disponível sob essa licença, incluindo qualquer alteração realizada nesse. Embora as razões por trás da criação de licenças como copyleft e a GPL têm sido principalmente ideológica (STALLMAN, 2010), o termo mais comercial[4] Open Source tem impulsionado a GPL e licenças similares a tornarem-se dominante no mundo da TI (Tecnologia da Informação).

A filosofia do Software Livre, tem tido um impacto muito além do mundo da TI em si. Software Livre pode ser visto como um exemplo de *Crowd Sourcing*[5] - um processo de produção aberto a colaboração dos usuários e o público interessado em geral - mas é muito mais antigo que esse conceito. Um exemplo disso são sites como Wikipedia, mídias sociais e colaborativas como *Facebook*, *Youtube* e *Flickr*

---

1 http://www.gnu.org/copyleft/
2 http://www.gnu.org/philosophy/free-sw.html
3 http://www.gnu.org/philosophy/open-source-misses-the-point.html
4 http://www.gnu.org/philosophy/open-source-misses-the-point.html e http://www.opensource.org/osd.html
5 http://crowdsourcing.typepad.com

O KDE[6] é uma comunidade de Software Livre que tem desenvolvido e entregue Software Livre e *Open Source* há mais de 15 anos. De acordo com a maioria das métricas, é a segunda maior comunidade no mundo, apenas atrás da comunidade do Kernel do Linux. Entre os seus produtos estão uma variedade de aplicações, para a computação móvel, *desktops* e uma plataforma de desenvolvimento onde os aplicativos são projetados e desenvolvidos. KDE desenvolve software principalmente para Linux, mas seus produtos também estão disponíveis para Windows, Mac OS X e um número crescente de dispositivos móveis. A comunidade é atualmente constituída por cerca de 800 a 1000 desenvolvedores ativos[7] e um número semelhante de contribuidores para arte gráfica, tradução, desenvolvimento de sites e outras tarefas dentro do projeto.

Os aplicativos desenvolvidos pelo KDE são escritos em C++, linguagem de programação projetada e implementada por Bjarne Stroustrup[8], utilizam Qt[9], um framework multiplatforma sob a licença LGPL (*Lesser General Public License*); e KDE-Libs, bibliotecas de software em C++/Qt que compõem o *KDE Developer Platform* ou KDE SDK, exigida por todos os aplicativos do KDE *Software Compilation* (KDE SC).

O Umbrello é a ferramenta de modelagem UML (*Unified Modeling Language*) distribuída com as ferramentas de desenvolvimento do KDE. O Umbrello foi escrito inicialmente em C++ para sistemas Unix pelo estudante universitário Paul Hensgen[10]. Em 2002, Jonathan Riddell[11] assumiu o projeto refatorando o código e incluindo novas funcionalidades. No mesmo ano, o Umbrello foi integrado as aplicações do KDE utilizando o framework Qt. Em 2008, Gopala Krishna[12], começou a adaptação do limitado Framework QCanvas do Qt 3 para o QGraphicsView do Qt 4.

---

6 http://kde.org
7 http://www.kde.org/announcements/4.3
8 http://www2.research.att.com/~bs
9 http://qt.nokia.com
10 http://phensgen.users.sourceforge.net/
11 http://jriddell.org
12 http://krishnaggk.blogspot.com

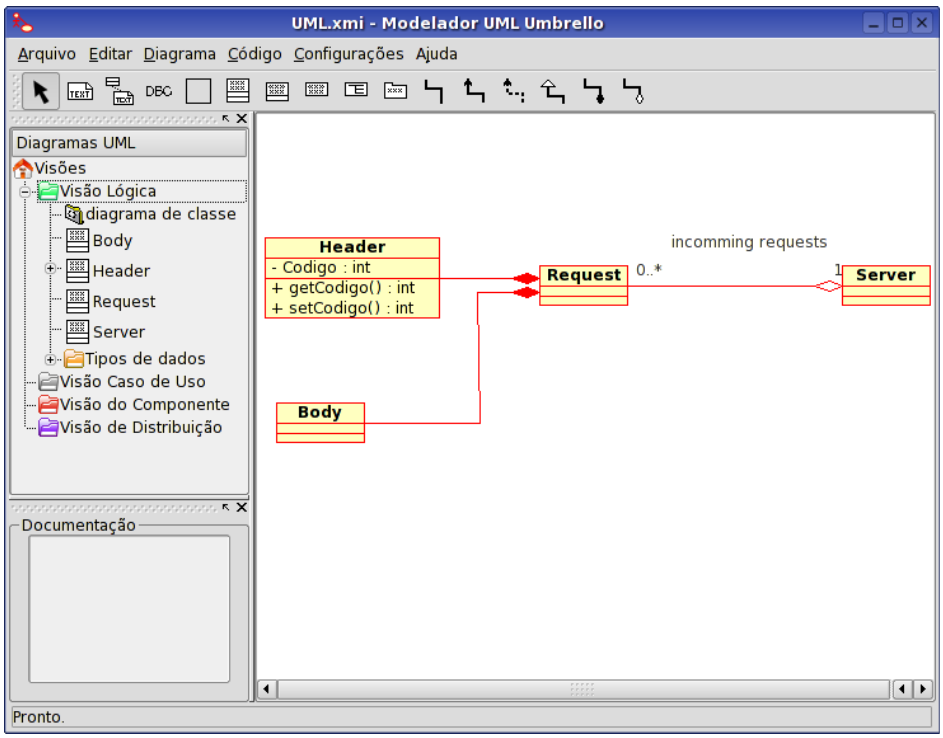

Imagem 1: Umbrello no KDE 3, utilizando QCanvas

Hoje o Umbrello tem uma versão para o KDE 4.6, mas a versão com a adaptação iniciada ppor Krishna, está instável, como consequência, não foi integrada a versão atual do Umbrello. O Qt 4 alcançou um nível maior de estabilidade com melhorias como suporte a arquitetura *ModelView* e um novo *build system* modular. O Framework Graphics View surgiu na versão 4.2 e fornece recursos para manusear e interagir com elementos gráficos 2D, e um widget para a visualização dos itens[13].

13 http://doc.qt.nokia.com

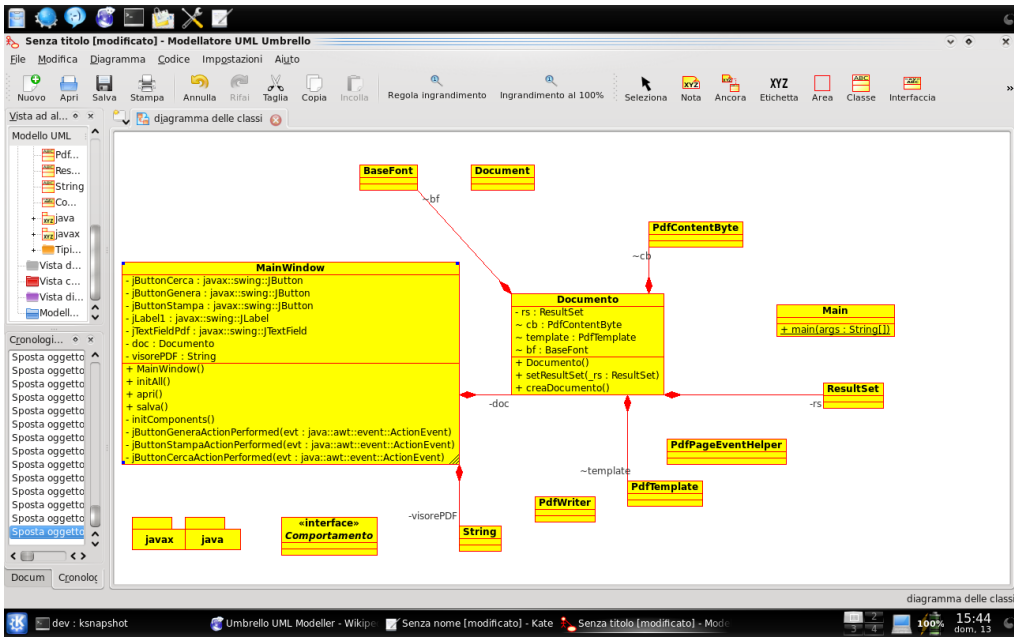

Imagem 2: Umbrello no KDE 4.1, utilizando Q3Canvas

Além de ser um editor de diagramas UML eficiente, o Umbrello ainda conta com funcionalidades pouco encontradas em ferramentas open source e software livre da mesma funcionalidade:

- Geração automática de código para linguagens como: Ada, C++, C#, Java, perl, PHP, Python, Ruby.
- Importação de classes e de projetos nas linguagens: Ada, C++, IDL, Java, Pascal e Python.
- Assistência para refatoração de código.

3.4. Definição do Problema

O Umbrello é a única ferramenta de modelagem UML presente no KDE SDK e amplamente utilizado por alguns projetos[14] do próprio KDE.

A necessidade de uma ferramente como o Umbrello é visível com a importância da UML para a Engenharia de Software[15] e o aumento do uso do Software Livre em outras áreas (GROOT, KÜGLER, ADAMS e GOUSIOS 2006).

Em "*The State of Open Source*" da empresa de pesquisas Garner predisse:

14 http://uml.sourceforge.net/users.php
15 http://www.uml.org/uml_success_stories/index.htm

"Em 2012, mais de 90 por cento das empresas irá usar código aberto de forma direta ou incorporado," (JUDGE, 2008).

A empresa de pesquisa IDC, escreveu em "*Linux in the New Economy*" (2009):

"IDC projeta que o crescimento do setor durante o período 2008-2013 de crescimento anual composta mercado global está projetado para software relacionados ao Linux vai liderar o de recuperação pós-recessão, com uma taxa (CAGR) de 23,6%. (...) Em comparação, o aumentar em 5% 2008-2013 CAGR ". (GILLEN, 2009)

Atualmente o Umbrello utilliza a classe Q3Canvas – a mesma classe QCanvas, renomeada apenas durante o processo de adaptação - que faz parte do Qt3Support, biblioteca que foi desenvolvida para facilitar a adaptação do Qt 3 para Qt 4, e que não devem estar no código de produção, mas apenas no processo intermediário, devendo ser imediatamente substituítas por uma classe equivalente do Qt 4. Nesse caso, o QCanvas deve ser substituído para a classe QGraphicsView[16]. Para cada classe do framework QCanvas existe uma equivalente no QGraphicsView.

O principal trabalho na adaptação do software é verificar as melhores opções existentes no novo framework para substituir os antigos métodos, sem alterar a forma como o Umbrello trabalha. Entendendo como o antigo framework QCanvas funciona e as melhorias desenvolvidas no QGraphicsView torna possível realizar uma otimização do código, ou seja, uma refatoração – atividade que conceitualmente não muda visualmente o software, mas sim como o *back-end*[17] funciona, o que implica em uma mudanças nos recursos que o software utiliza para exibir os diagramas na tela. Tais alterações são transparentes ao usuário.

16 http://doc.qt.nokia.com/4.7/q3canvas.html
17 http://www.businessdictionary.com/definition/back-end-application.html

3.5. Objetivos

3.5.1. Objetivo Geral

O objetivo do projeto é realizar uma adaptação da principal funcionalidade do software – a diagramação UML – mantendo as funcionalidades secundárias, como a importação e exportação de código, e ao final garantir uma fiel transição de versão de framework através de ferramentas de testes automatizados.

3.5.2. Objetivos Específico

. Eliminar o uso do QCanvas no Umbrello.

. Criar uma cobertura de testes para as principais classes.

4. Análise de Tecnologias/Ferramentas

As ferramentas foram selecionadas baseadas nas especificações do projeto já existente. Estas ferramentas são amplamente usadas e testadas, o que oferece mais estabilidade.

4.1. C++

Linguagem de programação em que o Umbrello foi desenvolvido.

4.2. Kdevelop 4.3

IDE para C/C++ e outras linguagens de programação. É baseado no KDE *Developer Platform*, no KDE e nas bibliotecas do Qt.

4.3. SVN (Subversion)

Sistema de controle de versão de software utilizado pelos projetos do KDE. Sendo usado desde o início do projeto.

4.4. Qt

Framework multiplataforma para C++, base do KDE e também utilizado pelo Umbrello.

4.5. QtTestLibs

Framework para desenvolvimento de testes unitários em Qt. Os testes serão escritos na última fase do projeto.

4.6. Squish Community Edition - IDE 4.1.0-SNAPSHOT

Software para testes automatizados de interface da empresa Froglogic[18]. Será utilizado a medida que a refatoração resultar em um minimo de funcionalidades testáveis.

18 http://www.froglogic.com

5. Descrição da Solução

O Umbrello utiliza recursos gráficos que permitem ao usuário desenhar diagramas 2D, essa é a sua principal função quanto software para diagramação UML. Logo, o projeto propõe a adaptação dos recursos do QCanvas - cerca de 30 classes são dependentes desses recursos - para o QGraphicsView utilizando técnicas de refatoração de código concluindo o projeto com ferramentas de testes para a garantir a continuidade das funcionalidade originais do software.

O Framework Graphics View tem uma abordagem baseada na programação model-view. Varias visualizações (QGraphicsView) podem conter única cena (QGraphicsScene), e a cena pode conter itens (QGraphicsItem) de diferentes formas geométricas.

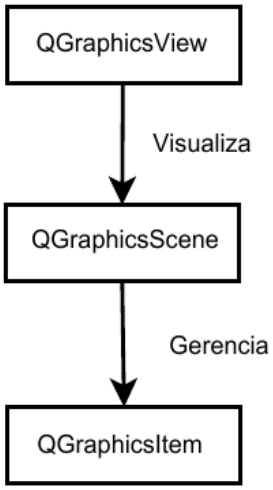

Diagrama 2: GraphicsView

Essa arquitetura é bastante similar a do QCanvas, mas o QGraphicsItem, equivalente ao Q3CanvasItem é mais poderoso e mais fácil de usar. Por exemplo, o QGraphicsView implementa o arrastar e soltar (*drag and drop*) de um item em uma cena, funcionalidade inexistente no Qcanvas:

> "Uma das mais notáveis melhorias na nova API (QGraphicsView) é o sistema de coordenadas. Agora podemos usar coordenadas em números reais para

adicionais um nível elevado de precisão a capacidade de visualização do QGraphicsView, especialmente quando utilizamos os zoom em uma cena. Isso simplifica ao ter que lidar com dads de fontes externas, como topologia, que geralmente usam coordenadas reais." (HANSSEN, 2006)

No QCanvas, a geometria dos items eram relativa ao canvas - a cena onde os items são adicionados. No QGraphicsView é possível criar items compostos, items dentro de items e redimensioná-los de forma independente ou de acordo com o seu item pai, independente da cena, como era no antigo framework.

As melhorias propostas com a adaptação do Umbrello são na sua maioria relacionadas a perfomance e otimização de código (HANSSEN, 2006), pois o Framework Graphics View utiliza os melhores recursos do Qt4 que a partir da versão 4.2 utiliza BSP *tree*[19] (*Binary Space Partitioning Tree*) com um método de indexação que acelera a busca dos itens na cena. (SLETTA, 2010)

Processo de Refatoração e Testes proposto como solução para o Projeto:

. Análise:

Processo de análise do problema e desenvolvimento de diagramas UML conforme a necessidade para a ajudar na resolução. É nesse momento que decide-se pela melhor técnica de refatoração para determinada classe e quais os testes que devem ser aplicados.

. Desenvolvimento (Refatoração):

Refatorar é uma técnica amplamente utilizada na engenharia de software, que consiste em alterar um código existente com o objetivo de obter um código legível, de fácil manutenção. Para garantir que o software mantenha o seu comportamento após as devidas melhorias na sua estrutura, aplica-se então testes automatizados. (FOWLER, 2002)

.Testes de Funcionalidade:

São os testes unitários – testes específicos para cada cada funcionalidade do software e classe do código - e testes automatizados – testes executados por ferramentas desenvolvidas para tal objetivo, que garantem o correto funcionamento do software após a refatoração.

. *Bug Fixing:* Correção dos erros encontrados após os testes.

19 http://web.cs.wpi.edu/~matt/courses/cs563/talks/bsp/bsp.html

6. Abordagem de Desenvolvimento

A metodologia adotada é uma variação do *Scrum*[20], que contará com a organização de *sprints* e pequenas entregas. De acordo com a solução proposta, o ciclo do *Scrum* será da seguinte maneira:

Ciclo															
Dias	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Tarefas	Análise				Desenvolvimento						Testes			*Bug Fixing*	

As alterações feitas no projeto estarão sendo versionadas com o controlador de versões SVN, logo, todo o histórico das alterações estará seguro e atualizado em svn+ssh://ayres@svn.kde.org/home/kde/branches/work/soc-umbrello-2011/.

O KDE funciona, como muitos projetos de Software Livre, de forma descentralizada, que é freqüentemente chamada de *Bazaar* (RAYMOND, 2001), modelo de desenvolvimento ao contrário do mais tradicional *Cathedrall* (RAYMOND, 2001). Em suma, no modelo *Cathedral* (RAYMOND, 2001), uma ou mais arquitetos projetam o produto, enquanto os desenvolvedores, sem muita autonomia, executam seus planos.

Em contraste, no estilo *Bazaar*, pequenas equipes de trabalho são semi-independentes dos componentes de todo o projeto. Qualidade e capacidade de resposta das equipes de desenvolvedores determina a composição final do produto. A tomada de decisão é feito em público através de uma combinação de discussão meritocrática e do desenvolvimento de soluções concorrentes. Os desenvolvedores com um longo histórico na comunidade, muitas vezes exercem uma influência considerável e a maioria das aplicações tem um mantenedor que toma decisões. No entanto, devido à natureza voluntária, na maior parte do trabalho (mesmo os pagos por empresas costumam agir muito independente), não é possível dizer às pessoas o que fazer. Logo, o Umbrello tem uma equipe responsável, atualmente pequena em comparação com outros projetos do KDE, com um mantenedor, Jonathan Riddell.

20 http://www.scrumalliance.org/

7. Arquitetura do Sistema (Modelagem)

7.1. Diagrama de Classes

7.1.1. Antes

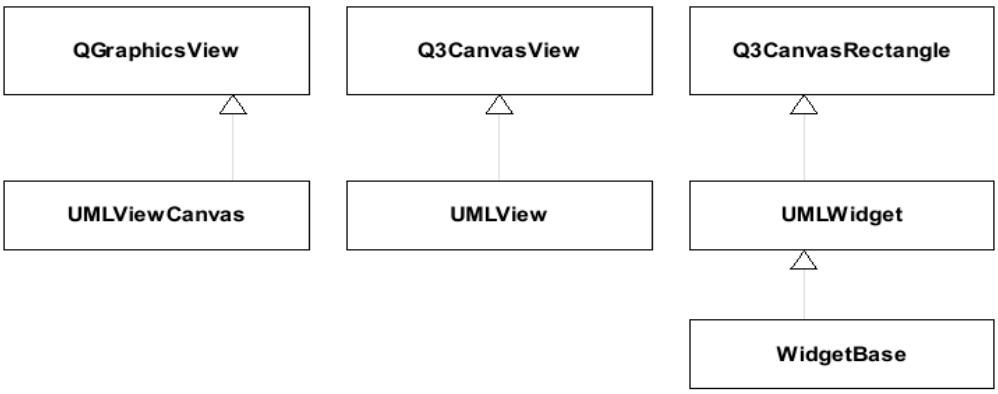

Diagrama 1: Diagrama Simplificado de Classes antes da refatoração

7.1.2. Depois

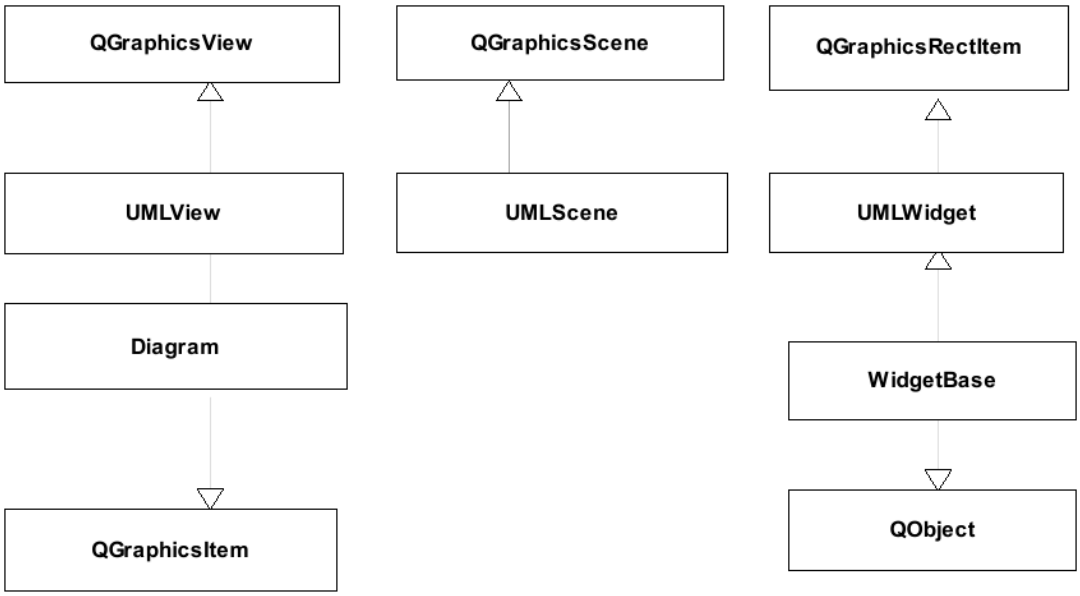

Diagrama 2: Diagrama de Classes Simplificado após refatoração

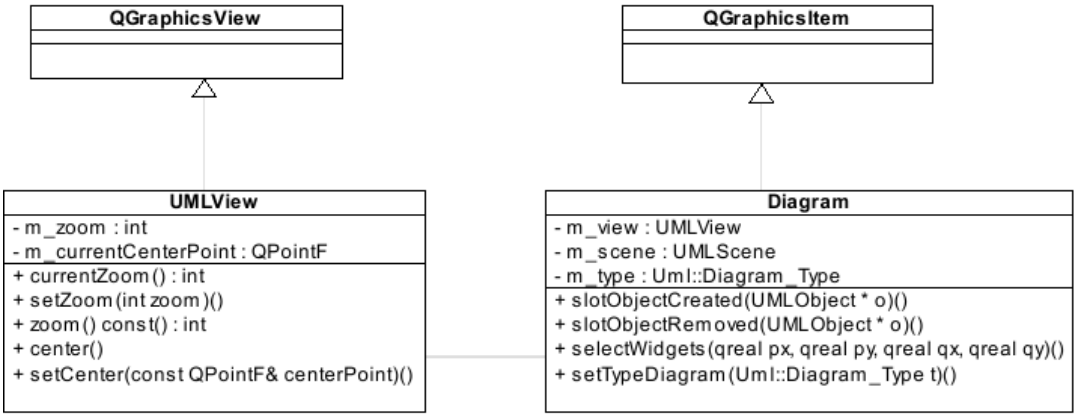

Diagrama 3: Diagrama de classes

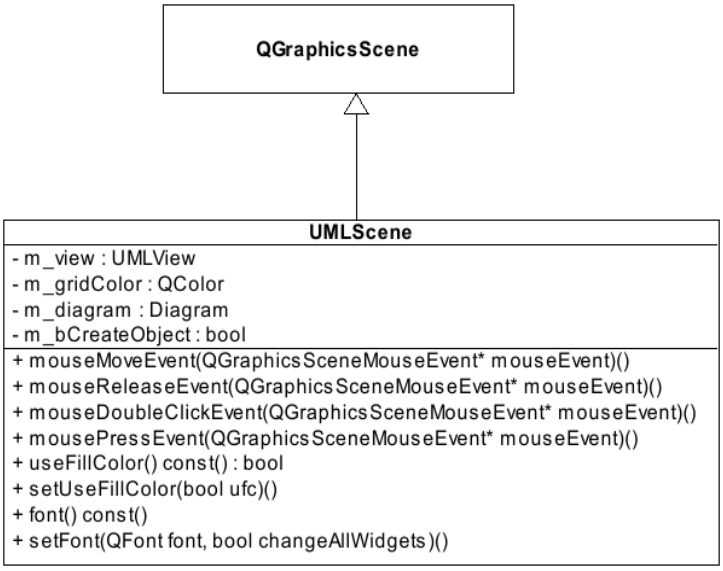

Diagrama 4: Diagrama de Classes

7.2. Diagrama de Sequência

Diagrama 5: Diagrama de Sequência

O diagrama explica a arquitetura do QGraphicsView, onde o UMLDoc cria um novo objeto UMLView (QGraphicsView), que por sua vez aloca a sua UMLScene (QGraphicsScene). O mesmo UMLDoc dispara o sinal sigObjectCreated quando o usuário adiciona um novo widget na tela, tal sinal é recebido pelo slot do Diagram, slotObjectCreated, que por sua vez cria um novo UMLWidget.

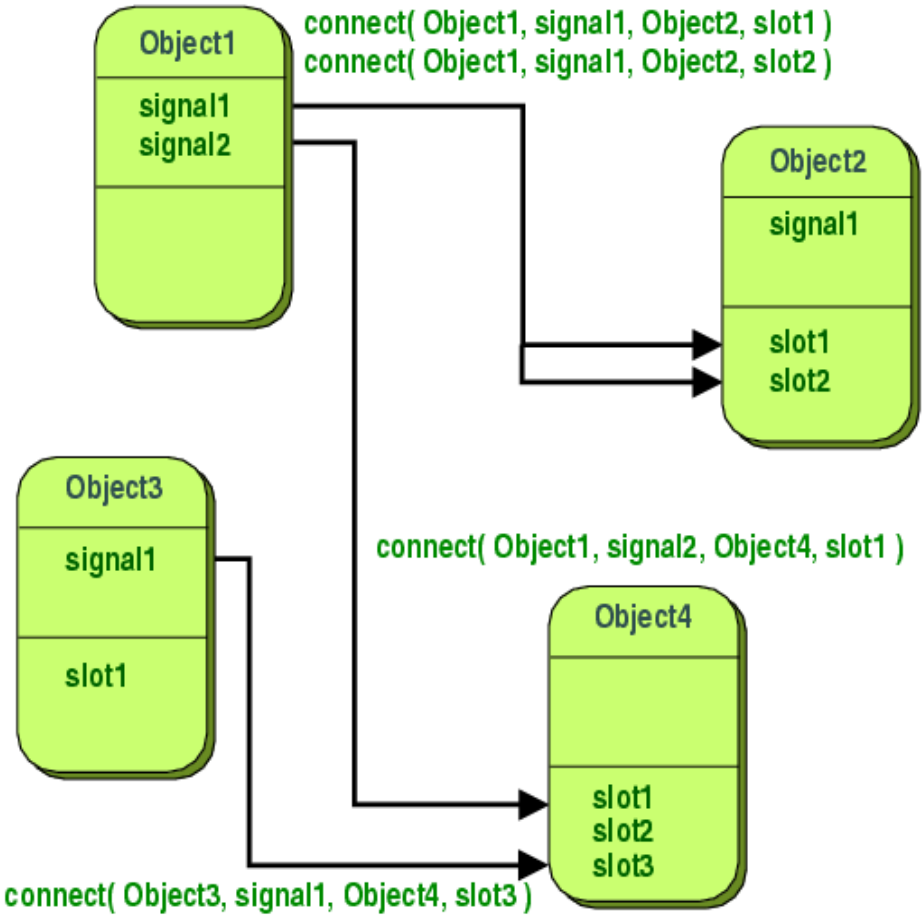

Imagem 3: Signals e slots - comunicação entre objetos no Qt

O Qt gerencia a comunicação entre objetos da aplicação através de *signals* e *slots*[21]. Um *signal* é emitido quando determinado evento ocorre e é tratado por um *slot* especificado no código da aplicação. Um slot nada mais é do que uma função que é chamada em resposta a um sinal em particular.

21 http://doc.qt.nokia.com/4.7/signalsandslots.html
Fonte Imagem 1: http://doc.qt.nokia.com/4.7/images/abstract-connections.png

8. Aspectos de Implementação do Protótipo

8.1. Funcionalidades a serem implementadas

. Eliminar o uso do QCanvas no Umbrello.

. Criar uma cobertura de testes para as principais classes.

8.2. Protótipo

Funcionalidades (classes refatoradas):

. UMLView

. UMLViewCanvas

. WidgetBase

. UMLWidget

. UMLObject

. BoxWidget

. EntityWidget

TCCI: . Eliminar o uso do QCanvas no Umbrello.

O primeiro sprint de desenvolvimento foi basicamente um teste em cima do cronograma planejado. As primeiras alterações foram um exemplo de como não refatorar um software, pois foi contra ao que é previsto como uma boa prática de refatoração e testes, alterações pequenas e devidamente planejadas para que seja possível encontrar os erros mais rapidamente.

> "Refatorar siginifica alterar os programas em pequenos passos. Se você cometer um erro, é fácil de encontrá-lo." (FOWLER, 2001)

Ao começar a adaptar as primeiras classes planejadas, encontrou-se a primeira diferença entre QCanvas e QGraphicsView: o sistema de coordenadas. No QCanvas as coordenadas são em números inteiros e no QGraphicsView em números reais. Ao começar as primeiras alterações notou-se que seria necessário alterar em mais métodos

do que o planejado, o que gerou muitos erros, sem o devido controle do funcionamento do software e das classes afetadas pela alteração. Para o segundo sprint, realizou-se uma análise melhor, decidindo-se tratar das coordenadas em um próximo sprint, pois é possível continuar usando as coordendas em tipo inteiro.

A melhor maneira de trabalhar encontrada para realizar as alterações por partes foi manter as classes antigas e criar as novas classes em conjunto. Assim, incluir as chamadas para os novos métodos através diretivas de pré-processamento em C++ nos arquivos antigos mantendo o funcionamento dos dois e testando o funcionamento dos novos métodos.

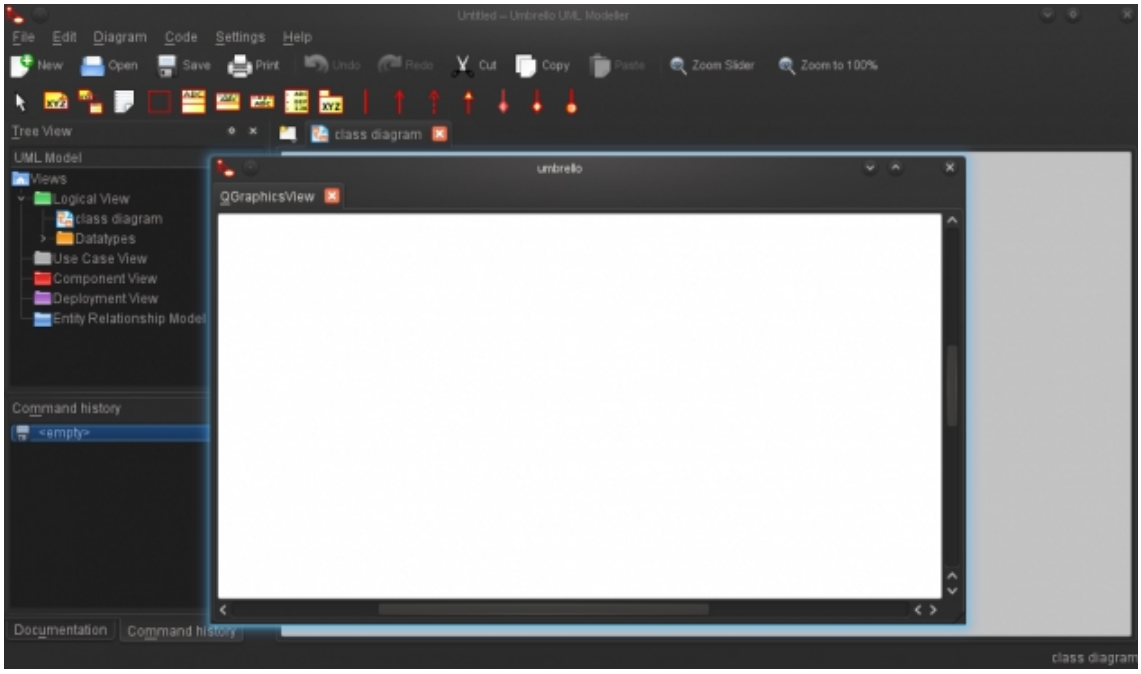

Imagem 4: Resultado obtido ao manter o antigo QCanvas (janela atrás) funcionando com o novo QGraphicsView (janela na frente).

Diretivas de pré-processamento[22] são linhas incluídas no código, precedidas pelo sinal #, logo não são consideradas declarações da linguagem de programação. Quando o pré-processador encontra essa diretiva, ele substitui qualquer ocorrência dessa declaração no resto do código. O pré-processador é executado antes da compilação, assim essa substituição é realizada antes de qualquer código ser de fato gerado pelas declarações da linguagem de programação propriamente dita.

Por exemplo, a classe UMLDoc utiliza a classe UMLView, que foi refatorada. Logo, no header umldoc.h foi realizada a declaração:

22 http://www.cplusplus.com/doc/tutorial/preprocessor/

```
//diretiva de pre-processamento
#define SOC2011 1
```

Na declaração dos métodos, foi utlizada uma declaração condicional, que permite ao pré-processador incluir ou descartar parte do código de acordo com a diretiva declarada anteriormente, nesse caso, a diretiva possui o valor "1", representando um valor verdadeiro:

```
//metodos utilizando o antigo UMLView
void addView(UMLView *view);
void removeView(UMLView *view , bool enforceOneView=true );

//metodos utilizando o novo UMLView
#ifdef SOC2011
void addView(QGV::UMLView *view);
void removeView(QGV::UMLView *view, bool enforceOneView=true );
#endif
```

A técnica de refatoração, conforme Fowler (2001, p. 65) explica, mais utilizada foi a de dividir algumas classes que continham métodos que poderiam facilmente serem gerenciados por outra classe. A nova classe Diagram são alguns dos métodos da classe UMLView que antes gerenciavam as configurações dos diagramas. Com isso temos um código de mais fácil manutenção e correção de erros.

> "Se você vê a estrutura do mesmo código em mais de um lugar, você pode ter certeza que seu programa vai ser melhor se você encontrar uma forma de unificá-los." (FOWLER, 2001)

. UMLView (QGraphicsView)

A UMLView herdava anteriormente de Q3CanvasView. Nessa classe, além da adaptação para QGraphicsView, foi necessário mudar como o zoom era tratado. Nas últimas versões do Qt, ao invés de QMatrix[23], utiliza-se QTransform[24]. QTransform difere QMatrix (obsoleto), em que é uma verdadeiro matriz 3x3, permitindo transformações de perspectiva.

23 http://doc.qt.nokia.com/latest/qtransform.html
24 http://doc.qt.nokia.com/latest/qtransform.html

. UMLViewCanvas (QGraphicsScene)

A UMLViewCanvas é a nova UMLScene, herdando de Q3Canvas. Trata de todos os eventos ocorridos na cena. A classe mais próxima de QCanvas para a adaptação ao Framework QGraphicsView é a QGraphicsScene. A nova classe UMLScene trata de todas as configurações do diagrama (um Diagrama de Classe, um de Sequência ou um outro diagrama qualquer da UML) em cena: cor da linha, fonte e alinhamento.

. Diagram (herda de QObject e QGraphicsItem)

Nova classe extraída a partir da UMLScene, que passa a gerenciar os widgets na UMLScene. Os métodos dessa classe tratam dos widgets pertencentes a determinado diagrama em cena. Armazena as informações referentes a posição, tipo de diagrama e a lista de widgets pertencentes ao diagrama.

. WidgetBase (herda de QObject)

Classe base para o UMLWidget.

. UMLObject

Essa classe é a versão não gráfica do UMLWidget. Estes são criados e mantidos na classe UMLDoc. Essa classe contém toda a informação genérica necessária para todos os objetos UML.

. UMLWidget (herda de WidgetBase e QGraphicsRectItem)

Classe base para quase todos os elementos gráficos. A UMLWidget herdava anteriormente de Q3CanvasRectangle. A classe mais próxima para a adaptação ao Framework QGraphicsView é a QgraphicsRectItem. Essa classe mantém as informações referentes a posição de cada widget em determinado diagrama, como a posição, o tamanho e o tipo.

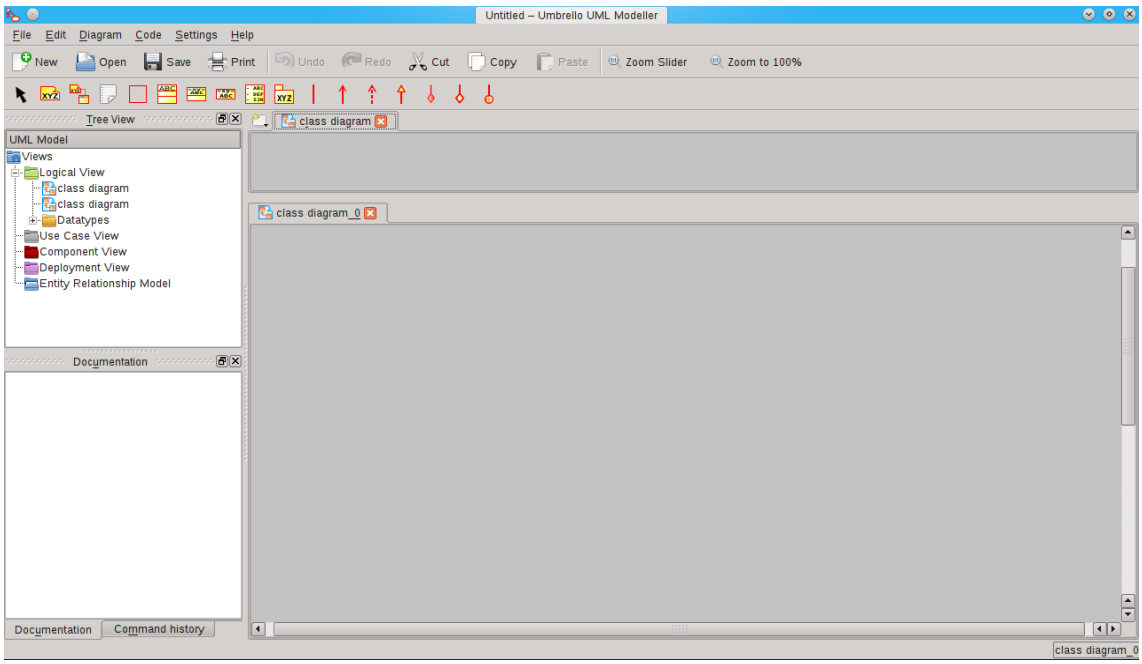

Imagem 5: Umbrello funcionando com o novo QGraphicsView (aba abaixo) em abas.

. BoxWidget (herda de UMLWidget)

. EntityWidget (herda de UMLWidget)

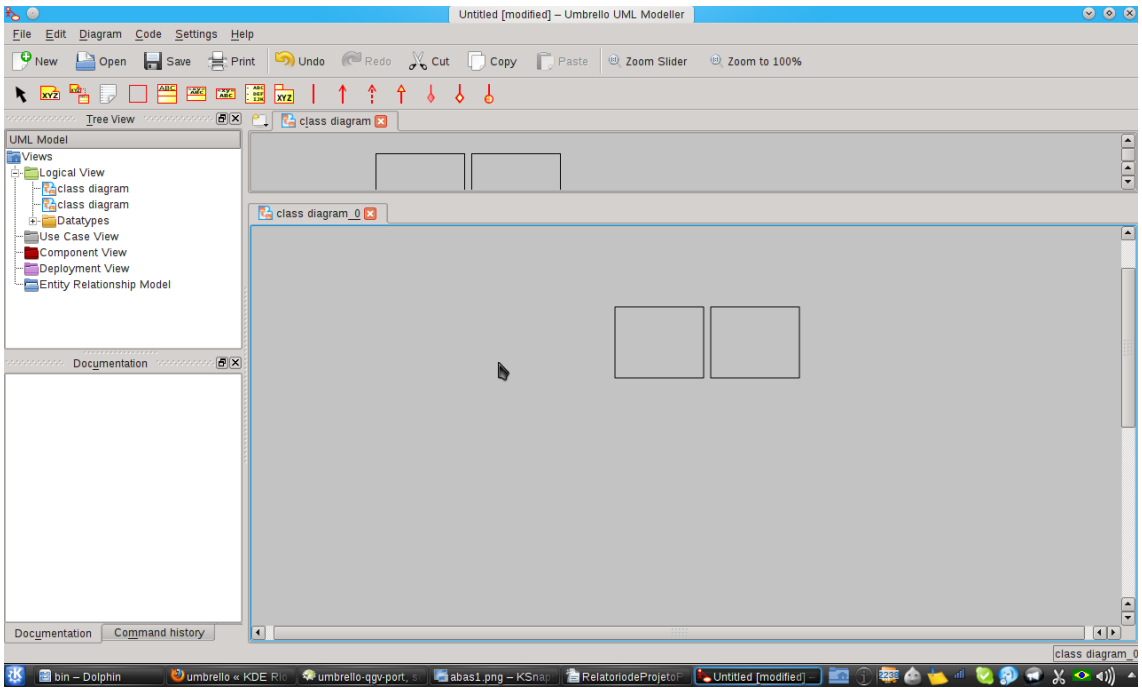

Imagem 6: BoxWidget no antigo QCanvas (aba acima) e no QGraphicsView (aba abaixo).

9. Validação

9.1. Estratégia

Os testes e validação fazem parte do processo de adaptação do software. Além do uso do QtTestLibs, framework para testes unitários em Qt, planejado para o final do projeto, inclui-se no processo um software específico para testes de interface, o Squish Community Edition.

Utilizando o Squish para gravar testes:

Com o Squish é possível automatizar os testes: o software a ser testado é executado através de uma VNC (Virtual Network Computing), ao executar os passos que devem ser testados, o Squish cria os scripts de teste, em Javascript, Perl, Python ou Tcl, mas também é possível editar e criar os scripts de acordo com a necessidade. Com os scripts gravados, pode-se executá-los quantas vezes forem necessárias para testes, ao invés de ter que fazer o processo manualmente para cada classe que deseja-se testar.

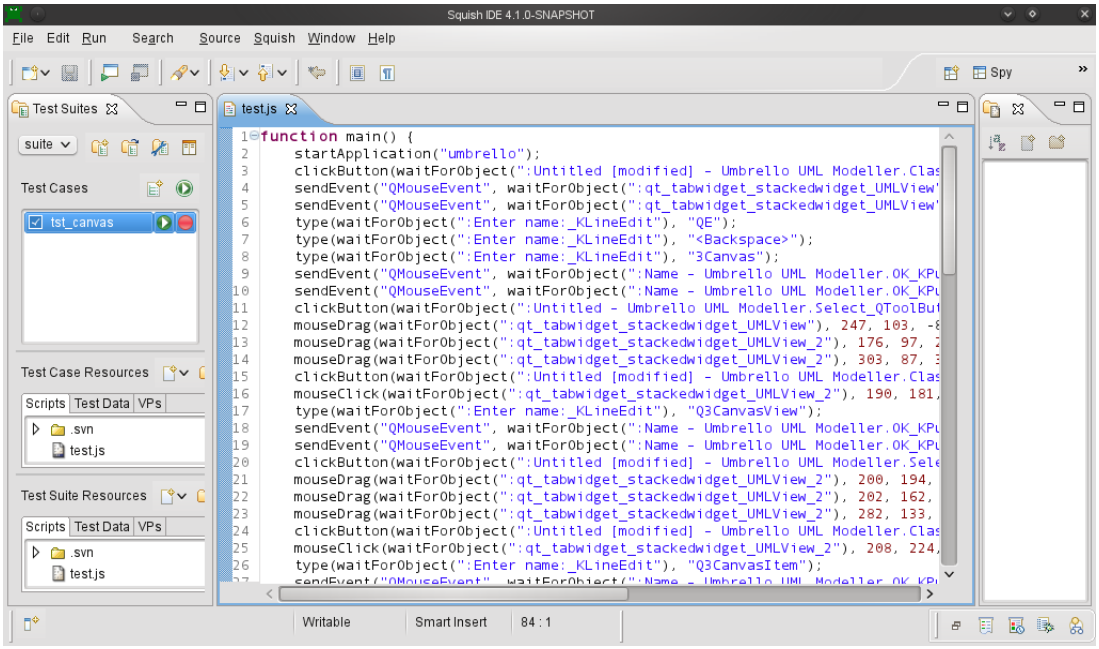

Imagem 7: Exemplo de Javascript para testes gerado pelo Squish

A única desvantagem é que a medida que se grava os testes, o sistema armazena uma tabela com os objetos da aplicação, mas se o objeto não estiver na tabela ainda, o teste não é executado por completo e é necessário incluir o objeto ou editar a tabela no manualmente, ou executar a aplicação no modo Spy ou ainda simplesmente gravar testes onde usa-se os objetos necessários no teste. Os objetos são elemento do software, como um menu, uma campo para edição de texto ou mesmo um reta no diagrama.

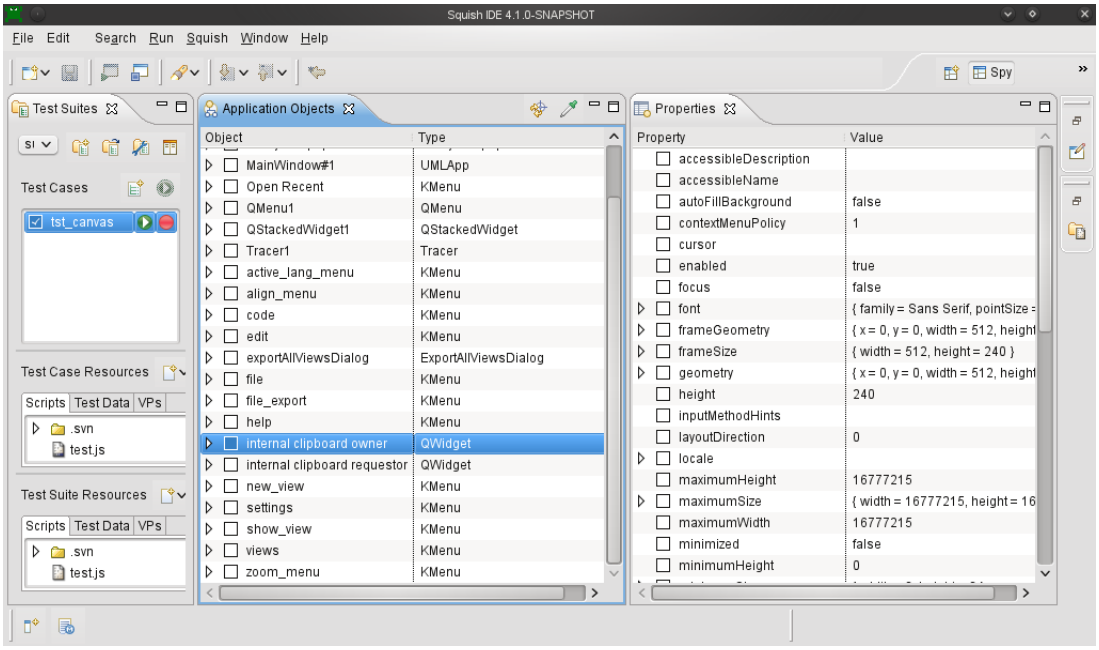

Imagem 8: Tabela de objetos da aplicação

A partir do Squish é possível obter os pontos onde a aplicaćão falha em tempo de execução e gerar relatórios a partir de um conjunto de testes executados.

10. Cronograma

O cronograma deste TCC foi todo baseado em *Sprints* de 15 dias, seguindo o ciclo explicado na abordagem de desenvolvimento do projeto, com datas programadas e concluídas. Pode-se também visualizar neste cronograma, separada por *sprints*, o produto de cada tarefa. As datas de entrega de relatórios estão destacadas separadamente dos *Sprints*.

TCC I				
Descrição da Atividade	**Produto**	**Início**	**Entrega**	**Detalhamento descritivo**
Plano de Trabalho	**Plano de Trabalho**	**18/03**	**01/04**	**Análise do Problema, definição da solução e criação do cronograma.**
Sprint 1: Classe UMLView.	Código	02/04	17/04	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 2: Classe UMLViewCanvas.	Código	18/04	02/05	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 3: Classe UMLObject.	Código	03/05	17/05	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 4: Classes UMLWidget e WidgetBase.	Código	18/05	01/06	Desenvolvimento, Testes e Bug Fix.
Relatório de Projeto	**Relatório**		**02/06**	**Entrega da versão Parcial do Relatório de Projeto Parcial para o Orientador**
Sprint 5: Classes LinePath, Circle, SubsetSymbol, SeqLineWidget, AssociationWidget, ActivityWidget, ActorWidget, ArtifactWidget e BoxWidget.	Código	02/06	16/06	Testes e Bug Fixing.
Relatório de Projeto	**Relatório**		**10/06**	**Entrega do Relatório de Projeto Parcial (versão final)**
Apresentação de Status do Projeto	**Apresentação**	**13/06**	**17/06**	**Apresentação de Status do Projeto.**

Tabela 1: Cronograma TCCI

TCC II				
Descrição da Atividade	**Produto**	**Início**	**Entrega**	**Detalhamento descritivo**
Sprint 6: Classes ComponentWidget, DatatypeWidget, EntityWidget, EnumWidget, FloatingDashLineWidget, FloatingTextWidget. NodeWidget.	Código	17/06	01/07	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 7: Classes ObjectWidget, PackageWidget, PinWidge, PreconditionWidget, RegionWidget, SignalWidget, StateWidget, UseCaseWidget and MessageWidget.	Código	02/07	17/07	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 8: Classes ForkJoinWidget, CombinedFragmentWidget and CategoryWidget, ClassifierWidget.	Código	18/07	01/08	Testes e Bug Fix.
Sprint 9: Trabalhar no suporte a melhores estereótipos.	Código	02/08	17/08	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 10: Completar suporte a Undo/Reddo.	Código	18/08	01/09	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 11: Testes e Bug Fix.	Código	02/09	16/09	Testes e Bug Fix.
Relatório de Projeto Atualizado	**Relatório**		**12/09**	**Entrega do Relatório de Projeto Atualizado.**
Sprint 12: Relatório de Andamento	Relatório	17/09	01/10	Escrita do Relatório.
Seminário de Andamento	**Apresentação**	**19/09**	**23/09**	**Apresentação do Andamento do Projeto.**
Sprint 13: Correções e melhorias no Relatório.	Relatório	02/10	16/10	Escrita do relatório.
Sprint 14: Início do desenvolvimento do artigo.	Artigo.	17/10	31/10	Escrita do artigo.

TCC II				
Descrição da Atividade	**Produto**	**Início**	**Entrega**	**Detalhamento descritivo**
Sprint 15: Desenviolvimento do Artigo final.	Artigo.	01/11	15/11	Finalizando artigo e relatório final.
Sprint 16: Finalizando artigo e relatório final.	Relatório.	16/11	30/11	Finalizando artigo e relatório final.
Relatório de Projeto	**Relatório**		**21/11**	**Entrega do Relatório Final de Projeto.**
Apresentação Final	**Apresentação**	**28/11**	**02/12**	**Apresentação da Banca Final.**
Relatório Final de Projeto Revisado	**Relatório**		**12/12**	**Entrega do Relatório Final de Projeto Revisado.**

Tabela 2: Cronograma TCCII

11. Considerações Parciais

Para o primeiro protótipo planejava-seconseguir uma visualização dos widgets em tela (os diagramas propriamente ditos), mas não é possível manter o QCanvasItem funcionando em uma QGraphicsScene. Com a base pronta, as classes até então refatoradas, vai ser possível realizar um trabalho mais rápido para concluir a adaptação do software, finalizar o suporte a Undo/Redo e escrever os testes até a data prevista de conclusão prosposta.

12. Referências Bibliográficas

RAYMOND, Eric S. **The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary**. O'Reilly & Associates, Inc. Sebastopol, CA, USA. 2001. Disponível em: <http://catb.org/~esr/writings/homesteading/cathedral-bazaar/> Acesso em: 19 mar. 2011.

GROOT, Adriaan de; KÜGLER Sebastian; ADAMS, Paul J.; GOUSIOS, Giorgo. **Call for Quality: Open Source Software Quality Observation**. European research projects, COSPA e CALIBRE. 2006. Disponível em: <http://istlab.dmst.aueb.gr/~george/pubs/2006-OSS-GKAG/paper.pdf> Acesso em: 19 mar. 2011.

JUDGE, Peter. **Gartner: Open source will quietly take over**. Abr. 2008. Disponível em: <http://www.zdnet.co.uk/news/it-strategy/2008/04/04/gartner-open-source-will-quietly-take-over-39379900/> Acesso em: 19 mar. 2011.

GILLEN, Al. **The Oportunity for Linux in the New Economy.** 2009. Disponível em: <http://www.redhat.com/f/pdf/idc-whitepaper-linux-in-new-economy.pdf> Acesso em: 19 mar. 2011.

STALLMAN, Richard. **Why Software Should Not Have Owners.** 2010. Disponível em: <http://www.gnu.org/philosophy/why-free.html> Acesso em: 19 mar. 2011.
GILLEN, Al. **The Oportunity for Linux in the New Economy.** 2009. Disponível em: <http://www.redhat.com/f/pdf/idc-whitepaper-linux-in-new-economy.pdf> Acesso em: 19 mar. 2011.

FOWLER, Martin. **Refactoring: Improving the Design of Existing Code**. 2ª ed. New York: Addison-Wesley, Fev. 2001.

BLANCHETTE, Jasmin; SUMMERFIELD, Mark. **C++ GUI Programming with Qt4**. 2ª ed. Boston: Prentice-Hall Open Source Software Development Series, Fev. 2008**.**
SUMMERFIELD, Mark**. Advanced Qt Programming: Creating Great Software with C++ and Qt 4**. 1ª ed. Boston: Prentice-Hall Open Source Software Development Series, Jul. 2010**.**

ANSSEN, Andreas Aardal. **A Better Canvas.** Qt Quarterly, 2006. Disponível em: <http://doc.trolltech.com/qq/qq19-graphicsview.html> Acesso em: 25 mar. 2011.

HANSSEN, Andreas Aardal. **A GraphicsView sneak-peek.** Maio 2006. Disponível em: <http://labs.qt.nokia.com/2006/05/01/a-graphicsview-sneak-peek/> Acesso em: 25 mar. 2011.

SLETTA, Gunna. **Qt Graphics and Performance – The Cost of Convenience.** Jan. 2010. Disponível em: <http://labs.qt.nokia.com/2010/01/11/qt-graphics-and-performance-the-cost-of-convenience/> Acesso em: 26 mar. 2011.

GNU Operating System; Free Software Fundation. **The Free Software Definition.** Nov. 2010. Disponível em: <http://www.gnu.org/philosophy/free-ww.html> Acesso em: 20 mar. 2011.

LYONS, Daniel. **Linux Rules Supercomputers**. Forbes. Mar. 2005. Disponível em: <http://www.forbes.com/2005/03/15/cz_dl_0315linux.html> Acesso em: 22 mar. 2011.

Componentes Re-utilizados

- Classes do Umbrello.
- KDE-Libs.