UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CLEBER ANTONIO GUGEL MACHADO

QOC EDITOR

Projeto Final de Graduação

Marcelo Soares Pimenta
Orientador

Porto Alegre – RS, Julho de 2013.

"A questão sobre se os computadores sabem ou não pensar é equivalente à questão sobre se os submarinos sabem ou não nadar."

E. W. Dijkstra

"O poder da Matemática reside na evasão a todo o pensamento supérfluo e na prodigiosa economia de operações mentais."

Ernst Mach

"A imaginação é mais importante do que o conhecimento."

Albert Einstein

"Alan Turing criou a computação, Bill Gates colocou um computador em cada casa, mas foi Mark Zuckerberg com sua rede social de massa quem conseguiu dar um real sentido para tudo isso."

Cleber Antonio Gugel Machado

Este trabalho é dedicado a Deus,
pelas inúmeras oportunidades nesta vida,
e as pessoas para quem ele for útil.

AGRADECIMENTOS

À minha esposa Josoela, por tanto apoio e por dividir sua vida comigo.

Aos meus Pais pela educação e pelo esforço para me dar essa oportunidade.

Aos meus irmãos por terem ajudado e pelo carinho de sempre.

À minha família pela compreensão de minhas ausências.

Aos meus amigos pelo carinho e pela torcida.

Ao meu orientador, Prof. Marcelo Soares Pimenta pelo conhecimento transmitido, pela atenção, confiança, dedicação e amizade.

Aos Professores de quem fui aluno pelo bom ensino que recebi.

Aos Professores e Técnicos do Instituto de Informática da UFRGS pela ótima estrutura que me proporcionaram usar.

Aos colegas pela parceira nos estudos e por terem me escolhido representante discente e presidente do diretório acadêmico.

Ao povo brasileiro que com seus impostos financia a Universidade Pública gratuita.

Aos matemáticos por terem criado as bases para a computação.

Aos físicos e engenheiros por terem criado as tecnologias que usam a computação.

RESUMO

No processo de desenvolvimento de software, uma importante tarefa é a documentação interna (para a equipe) do projeto do software para propósitos de futuras manutenções e aprimoramentos. Porém capturar e registar de forma que informações possam ser utilizadas no futuro é uma tarefa complexa e onerosa. As pesquisas em Design Rationale visam criar modelos eficientes para documentação do projeto de software.

Design Rationale (DR) é o processo de documentação de tomada as decisões, sistematizando as razões que deram suporte a cada decisão, incluindo suas justificativas e alternativas consideradas. O Design Rationale tem grande potencial para ser uma tecnologia que agregue valor ao processo de desenvolvimento de software. No entanto, Design Rationale ainda não é utilizado adequadamente por empresas em casos reais e, raramente, há casos de captura de informações, fornecendo pouca oportunidade para investigar o problema da captura do Design Rationale na prática.

O QOC Editor é um gerador de diagramas QOC, que permite aos usuários gerenciar o conhecimento relacionado ao processo de tomada de decisão em projetos de software. Através dessa ferramenta, podemos representar, de forma simples e intuitiva, os problemas e alternativas ponderadas antes de cada decisão, reduzir a perda de capital intelectual quando pessoas saem do projeto, propiciar a outros projetos o reuso de experiências positivas e prevenir a repetição de problemas já vivenciados anteriormente, entre outras vantagens.

Neste trabalho foram resumidos os principais conceitos de DR, assim como resumidos os principais modelos de DR, dentre os quais o QOC. Em seguida, foram apresentadas as características do QOC Editor e descritos aspectos de sua modelagem e implementação.

Espera-se que com esta ferramenta e com o suporte que ela oferece, mais desenvolvedores adotem a ideia de modelar DR.

ABSTRACT

In the process of software development, an important task is the internal documentation (for the team) design software for the purpose of future maintenance and enhancements. But capturing and recording so that information can be used in the future is complex and costly. Research in Design Rationale aim to create efficient models to project documentation software.

Design Rationale (DR) is the process of documentation of the decisions taken by systematizing the reasons that supported every decision, including its justification and alternatives considered. The Design Rationale has great potential to be a technology that adds value to the process of software development. However, Design Rationale is still not properly used by companies in real cases and, rarely, there are cases of information capture, providing little opportunity to investigate the problem of capturing Design Rationale in practice.

The QOC Editor is a QOC diagram generator, which allows users to manage knowledge related to the process of decision making in software projects. Through this tool, we can represent, in a simple and intuitive, problems and alternatives considered before each decision, reduce the loss of intellectual capital when people leave the project, provide the other projects the reuse of best practices and prevent the recurrence of problems have previously experienced, among other advantages.

In this work were summarized the main concepts of DR as well as a summary of the main models of DR, among which the QOC. Then we presented the characteristics of the QOC Editor and described aspects of modeling and implementation.

It is expected that this tool and support it provides more developers adopt the idea of modeling DR.

SUMÁRIO

LISTA DE ABREVIAÇÕES

ADO.NET - ActiveX Data Objects (conjunto de classes definidas pela .NET)
BCL - Base Class Library (Biblioteca de Classes Básicas)
CLR - Common Language Runtime
CLS - Common Language Specification
CSS - Cascading Style Sheets (Folhas de Estilo Encadeadas)
DD - Documentador de Decisões
DME - Device Modeling Environment
DR - Design Rationale
DRL - Decision Representation Language (Linguagem de Representação de Decisões)
DW - Data Warehouse
EDN - Electronic Design Notebook
ES - Engenharia de Software
gIBIS - Graphical Issue Based Information System
GUI - Graphical User Interface (Interface Gráfica do Usuário)
HTML - HyperText Markup Language (Linguagem de Marcação de Hipertexto)
IBIS - Issue Based Information System (Sistema de Informação baseado em Questões)
IDE - Integrated Development Environment (Ambiente Integrado de Desenvolvimento
JIT - Just-In-Time (compilação em tempo de execução)
JPG - Joint Photographic Experts Group
KBDS - Knowledge-Based Design System
LP - Linguagem de Programação
MDR - Modelo de Decisões e Raciocínios
MSIL - Microsoft Intermediate Language (linguagem de código intermediário)
PHI - Procedural Hierarchy of Issues (Hierarquia Procedural de Questões)
PHP - Hypertext Preprocessor (Pré Processador de Hipertexto)
PNG - Portable Network Graphics (formato de arquivo de imagem)
QOC - Questions, Options and Criteria (Questões, Opções e Critérios)
RUP - Rational Unified Process
SQL - Structured Query Language (Linguagem de Consulta Estruturada)
TI - Tecnologia da Informação
UML - Unified Modeling Language (Linguagem de Modelagem Unificada)
XML - eXtensible Markup Language (Linguagem de Marcação Extensível)
XPDL - XML Process Definition Language (Linguagem XML de Definição de Processos)

LISTA DE FIGURAS

LISTA DE TABELAS

1.0 INTRODUÇÃO

No processo de desenvolvimento de software, são definidas as funcionalidades do software e como essas funcionalidades serão implementadas. Para isso, muitas vezes, são utilizados modelos para registrar e documentar seus requisitos, sua especificação e o projeto e implementação.

No entanto, frequentemente, desejamos registrar as decisões relacionadas ao projeto de software, ou seja, não somente quais decisões são tomadas mas também, e principalmente, quais são os critérios destas decisões. Este registro, em teoria, contém as informações suficientes e necessárias para o completo entendimento do software, para a reutilização das experiências adquiridas e a recuperação do processo de tomada de decisão.

De maneira geral, apenas as decisões finais a respeito do projeto são documentadas. Design Rationale é o nome dado ao registro das razões (Rationale), os argumentos pró ou contra as decisões tomadas durante o projeto (design) e é usualmente abreviado como DR. DR consiste das informações adicionais aos documentos padrões em um processo de desenvolvimento de software, facilitando sua compreensão, manutenção e reuso. O desenvolvimento de mecanismos que facilitem a captura e representação de Design Rationale durante a elaboração de artefatos de software é ainda um desafio.

Este trabalho é um esforço para auxiliar a minimizar esta dificuldade de registro, definindo e implementando uma Ferramenta para Edição e Manipulação de um Modelo de Design Rationale (DR).

Neste capítulo inicial, apresentaremos o contexto do problema, o objetivo do trabalho, a justificativa e a organização da dissertação.

1.1 Contextualização

Ninguém imaginava que o software se tornaria um elemento tão importante para o mundo e que teria a capacidade de manipular a informação. Como muitos elementos computacionais, sofreu mudanças e continua sofrendo até hoje. Este crescimento computacional levou à criação de sistemas perfeitos e também problemas para quem desenvolve softwares complexos. As preocupações dos engenheiros de software para desenvolverem softwares sem defeitos e entregarem estes produtos no tempo marcado, leva à aplicação da disciplina de engenharia de software. Com o crescimento desse segmento, muitas empresas passaram a ter mais especialistas em TI, onde cada um tem sua responsabilidade no desenvolvimento de software,

diferentemente do que acontecia antigamente, onde um único profissional de software trabalhava sozinho em uma sala [PRESSMAN, 2006].

A Engenharia de Software é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de software. Isso vai desde os estágios iniciais de especificação de um sistema até a manutenção para que esse mesmo software sobreviva ao longo do tempo [SOMMERVILLE, 2003].

A Engenharia de Software evoluiu significativamente nas últimas décadas, procurando estabelecer técnicas, critérios, métodos e ferramentas para a produção de software, em consequência da crescente utilização de sistemas baseados em computação em praticamente todas as áreas da atividade humana. Isto levou a uma crescente demanda por qualidade e produtividade, tanto do ponto de vista do processo de produção como do ponto de vista dos produtos gerados.

Atualmente, a indústria de software representa uma área estratégica para o desenvolvimento industrial de qualquer economia, pois o software está sendo cada vez mais incorporado em uma imensa gama de produtos, além de constituir uma peça fundamental na estrutura organizacional de muitas empresas, condicionando sua eficiência produtiva.

Engenharia de software é a criação e a utilização de sólidos princípios de engenharia, a fim de obter softwares econômicos que sejam confiáveis e que trabalhem de forma eficiente em máquinas reais. Pode ser definida como uma aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software; isto é, a aplicação da engenharia ao software [PRESSMAN, 2006].

Segundo Pressman, o trabalho associado à engenharia de software pode ser categorizado em fases genéricas independente da área de aplicação, tamanho do projeto e complexidade. As fases necessariamente focam no que (definição), como (desenvolvimento) e modificações (manutenção).

Pressman define que a Engenharia de Software é uma tecnologia em 3 camadas: processos, métodos e ferramentas, sendo que a base de todas essas camadas é o foco na qualidade do software desenvolvido:

- Processos: constituem os elos de ligação que mantém juntos os métodos e as ferramentas e possibilita o desenvolvimento racional e oportuno do software de computador. É o alicerce da engenharia de software.
- Métodos: proporciona os detalhes de "como fazer" para construir o software;

- Ferramentas: proporcionam apoio automatizado ou semi-automatizado aos métodos (ferramentas CASE combinam software, hardware e um banco de dados).

O alicerce da engenharia de software é a camada de processo. O processo de engenharia de software é o adesivo que mantém unidas as camadas de tecnologias e que permite o desenvolvimento racional e oportuno de softwares de computador. O processo define um arcabouço que deve ser estabelecido para a efetiva utilização da tecnologia de engenharia de software. Os processos de software formam a base para o controle gerencial de projetos de software e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalhos (modelos, documentos, dados, relatórios, formulários etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas [PRESSMAN, 2006].

A massificação do produto de software levou a um crescente número de tecnologias e modelos de processo para engenharia de software que são um vasto conjunto de possibilidades de solução, mas que representam também um desafio: “a ênfase em tecnologias e modelos de processo obscurece o fato de que a engenharia de software é primariamente uma atividade baseada em pessoas e que o sucesso de um projeto ou produto é contingente das decisões feitas durante a engenharia” [DUTOIT, 2006].

O Design Rationale (DR) documenta o processo de tomada as decisões, sistematizando as razões que deram suporte a cada decisão, incluindo suas justificativas e alternativas consideradas [LEE, 1996].

Abordagens de Design Rationale buscam manter, de forma estruturada e organizada, as decisões-chave tomadas durante o desenvolvimento de um projeto, para que estas possam ser úteis especialmente em atividades futuras.

Design Rationale busca documentar, além da decisão final, também as razões por trás de cada decisão, incluindo as justificativas, as alternativas consideradas e os argumentos que levaram à determinada decisão.

Há vários modelos e notações para DR e a utilizada neste trabalho foi o QOC (Questions, Options and Criteria). QOC é baseado na formulação de questões, as quais são uma importante dimensão do espaço de projeto, representando os principais problemas que devem ser considerados, os quais compõem o espaço de alternativas. Critérios são objetivos positivos que servem como base para a avaliação e escolha entre as várias opções analisadas. Opções são as possíveis respostas às questões. Para a representação desta técnica, podemos ter relações positivas (PRO) entre um critério

e uma opção OU relações negativas (CONTRA) entre opção e critério, ou seja, a opção não é suportada pelo critério [MACLEAN, 1991].

Dentro da representação desta técnica, linhas contínuas representam relações positivas entre um critério e uma opção, enquanto linhas pontilhadas indicam uma relação negativa, ou seja, a opção não é suportada pelo critério.

1.2 Objetivo

O objetivo do trabalho é a definição e implementação de um editor de modelos QOC (Questions, Options, Criteria), um dos modelos de DR mais utilizados e que serve para estruturar e documentar as decisões e justificativas dos membros da equipe de desenvolvimento de software.

1.3 Justificativa

Durante o processo de desenvolvimento de software, uma importante tarefa é a documentação interna (para a equipe) do projeto do software para propósitos de futuras manutenções e aprimoramentos.

Na falta de uma documentação formal completa, as informações sobre decisões de projeto necessárias para posterior manutenção do software ficam subentendidas no código fonte, ofuscadas por detalhes de implementação, tornando a manutenção do software problemática.

Normalmente, a documentação padronizada de projetos contém a descrição do projeto final, isto é, o resultado final das decisões que foram tomadas, mas não contém o porquê da decisão tomada, nem as alternativas que existiam no momento da decisão. O projeto de um software sofre várias alterações durante o seu ciclo de desenvolvimento, tanto para fazer correções, adaptar-se a mudanças de plataforma ou incorporar novos requisitos ou funções. Porém, se a manutenção de software não for feita com a documentação adequada se torna muito complexa e custosa. Por isso, o Design Rationale é especialmente importante para projetos de software, já que fornece um auxílio para a compreensão das decisões de projeto tomadas e dos argumentos (pró e contra) do raciocínio subjacente à decisão.

O Design Rationale é uma documentação explícita das razões por trás das decisões feitas ao projetar um sistema através de cada um dos seus artefatos. Como definido por WR Kunz e Rittel Horst, em 1970: “DR visa proporcionar a argumentação

baseada em estrutura para o processo político, colaborativa de resolução de problemas complexos" [KUNZ,1970].

Uma das dificuldades em adotar algum modelo de Design Rationale é a captura das informações. O armazenamento de todas as decisões tomadas, bem como as rejeitadas, pode consumir muito tempo e ter um alto custo. No geral, quanto mais o processo de captura for intrusivo, maior a resistência encontrada. Além do tempo extra que será necessário a esta atividade, ela pode desviar o foco do projetista de sua tarefa usual. Por outro lado, não se pode esperar que o projetista saiba expressar estas informações sem o auxílio de uma ferramenta que o guie, pois é uma tarefa cognitiva de difícil execução [GRUBER, 1991].

Existem poucas ferramentas com suporte adequado para o DR que sejam fortemente baseadas no armazenamento e recuperação da informação, bem como na reutilização com o uso eficaz das notações [LACAZE, 2010].

1.4 Estrutura

Este texto está estruturado como segue. Após esta introdução (cap. 1), apresentaremos inicialmente no cap. 2 a definição e os conceitos básicos de Design Rationale. A seguir, descreveremos os modelos de DR mais relevantes e faremos uma comparação entre eles. Ainda no cap. 2 apresentaremos uma ferramenta de edição gráfica para o modelo QOC, a única que encontramos descrita na literatura.

No capitulo 3, definiremos nossa proposta de ferramenta para QOC - denominado QOC Editor - através da enumeração de seus requisitos e de um conjunto de funcionalidades básicas.

No capítulo 4, mostraremos o uso do QOC Editor através de exemplos da sua aplicação em alguns cenários hipotéticos e descreveremos o comportamento interno do aplicativo.

2.0 DESIGN RATIONALE

Inicialmente desenvolvido por Kunz e Rittel na década de 70, o Design Rationale é um conjunto de técnicas de documentação que visa registrar de forma explícita as razões por trás de decisões tomadas no processo desenvolvimento de um sistema ou artefato de software. Desde então, o Design Rationale tem sido usado em vários domínios de aplicações. Na década de 80, engenheiros de software começaram a adaptar as primeiras abordagens as suas necessidades no projeto de software. Atualmente, estas abordagens não chegam a oferecer suporte nem à metade da totalidade das atividades do projeto de software [DUTOIT, 2006].

O projeto de software envolve a captura de conhecimento que soluciona um problema, a adoção de uma arquitetura de solução, a definição de um conjunto de elementos (módulos ou classes) e a realização de sua transformação em um código escrito em uma linguagem fonte. Para atingir este objetivo, o projetista de software deve mapear os objetos do domínio do problema e suas relações em estruturas e funcionalidades durante a análise de requisitos dentro do contexto e das restrições da arquitetura, de forma a tornar possível a construção do software.

Dependendo do tamanho e da complexidade do software, seu projeto pode envolver a manipulação de uma grande quantidade de informações relacionadas ao problema, à tecnologia utilizada no projeto, à lógica de negócio necessária ao software ou ainda a outras atividades ou dependências do projeto. Essas informações são usadas para tomar as decisões de projeto durante o desenvolvimento do sistema.

A UML é uma linguagem gráfica para a visualização, especificação, construção e documentação dos artefatos de sistemas com uso intensivo de software. Pretendendo ser uma relação de documentação e expressões, comportamentos e ideias numa notação que é fácil de aprender e eficiente para se escrever. A UML fornece uma forma padrão de documentar o sistema, cobrindo a parte conceitual, os processos de negócios, as funções do sistema e, também, a parte mais concreta: como escrever as classes em uma linguagem específica, esquema de banco de dados e componentes reutilizáveis de software [BOOCH, 1999].

No desenvolvimento dos Diagramas de UML registram-se apenas as informações relativas às decisões finais de determinada fase, pois sem uma ferramenta adequada se torna muito onerosa a tarefa de documentar cada uma das decisões com todas as alternativas levantadas.

Diferentemente do processo padrão de documentação tradicional, que consiste na descrição do resultado final de um projeto, a do Design Rationale busca documentar, além da decisão final, também as razões por trás de cada decisão,

incluindo as justificativas, as alternativas consideradas e os argumentos que levaram à determinada decisão [MORAN, 1996].

Documentando de maneira adequada as decisões com todas as alternativas levantadas, leva a um melhor conhecimento do projeto, facilita sua integração, mudanças ou adaptações, evita que uma análise seja feita várias vezes e, mesmo após sua conclusão, traz uma maior facilidade de manutenção do sistema. Assim, fica evidente a importância da captura destas informações durante o processo de desenvolvimento de software.

Considerando a grande quantidade de informações analisada em projetos de software e o volume de decisões tomadas ao longo do seu desenvolvimento, é necessária uma ferramenta que armazena as informações de forma eficiente para documentar e manter estas decisões e os motivos pelos quais estas foram tomadas. Esse tipo de ferramenta deve implementar um modelo de Design Rationale.

2.1 Representação das Informações de Design Rationale

Há vários tipos distintos de representação do Design Rationale: obviamente, a forma de representação influencia diretamente na forma de captura das informações, como veremos a seguir.

2.1.1 Representação Informal

Representações são classificadas como informais quando elas capturam informações na forma gerada pelo projetista durante o projeto, ao invés de exigir que uma nova estrutura seja utilizada. Por essa razão, são facilmente entendidas pelas pessoas, mas não podem ser facilmente utilizadas pelo computador. As justificativas de projeto (rationales) são capturadas em uma forma não estruturada [BURGE, 2005]. Existem vários exemplos de representações informais, entre elas, gravações de áudio ou de vídeo, fotos, anotações, relatórios, e-mails, descrição de experimentos, etc.

2.1.2 Representação Formal

Representações são classificadas como formais quando elas capturam informações usando uma notação formal. Uma abordagem formal permite a utilização dos dados pelo computador, mas nem sempre apresenta as informações em uma forma que as pessoas possam entender facilmente. Além disso, requer que os dados

sejam inseridos no sistema no formato especificado de entrada do sistema [CONKLIN, 1995].

Entre os exemplos de representações formais estão as regras embutidas em um sistema especialista [CONKLIN, 1995] e um sistema de aprendizado de máquina que utiliza rastreamentos de solução de problemas passados para solucionar problemas futuros.

2.1.3 Representação Semiformal

Uma Representação Semiformal permite a compreensão de informação tanto por humanos quanto por computadores, busca oferecer certo poder computacional de uma abordagem formal, mas em um formato que é entendido pelas pessoas. São usadas tipicamente como linguagem de argumentação.

Quanto mais formal for a representação, mais serviços o sistema poderá oferecer. No entanto, mais complexa e demorada se torna a tarefa de captura Rationale [LEE, 1987].

2.2 Principais Abordagens

O processo de Design Rationale envolve três aspectos principais: a captura, a representação e o uso de Design Rationale. A obtenção de informações, visando à técnica do Design Rationale, ou seja, a captura e recuperação das informações necessárias podem ser realizadas de várias maneiras.

Geralmente, o esquema de representação determina os métodos usados para capturar e recuperar o Design Rationale, possibilitando sua utilização no design de novos artefatos. No entanto, a utilização de uma determinada abordagem não exclui a existência de outras, ou seja, é possível utilizar, em um mesmo sistema, alternativas distintas de abordagens ao Design Rationale.

A obtenção e recuperação das informações de raciocínio são baseadas em quatro abordagens não exclusivas entre si. A seguir, apresentaremos as principais abordagens de Design Rationale conhecidas [BURGE, 1998].

2.2.1 Baseada em Argumentação

Design rationale baseado em argumentação é uma abordagem essencialmente representacional e que usa um formato gráfico semiformal para organizar a estrutura de argumentos. O design rationale é essencialmente utilizado para representar os argumentos que definem o design [GARCIA, 1993].

Estes argumentos consistem em questões levantadas, alternativas para a resolução destas questões e argumentos prós e contras para cada alternativa. É formado por nós e relacionamentos. Oferece uma maneira de manter consistência nas tomadas de decisão, acompanhar decisões e disseminar o conhecimento do raciocínio do projeto com outras pessoas [HU, 2000].

2.2.2 Baseada em História

O design rationale baseado em história consiste em uma sequência de eventos que ocorreram enquanto o design foi desenvolvido [GARCIA, 1993]. Trata-se do histórico do projeto, ou seja, da sequência de eventos que ocorreram durante o projeto. Esta informação pode ser guardada como entradas do documento de projeto ou como um arquivo de mensagens de e- mails, entre outros [BURGE, 1998].

As informações armazenadas são suficientemente ricas para reconstruir o projeto e, consequentemente, fornecer explicações sobre ele. Neste modelo, o rationale é explícito e modelos de produto sobrecarregam as atividades do projeto, pois a documentação deve conter todas as informações e ações que aconteceram durante o projeto. O projetista acaba se distraindo com a grande quantidade de detalhes, que poderiam ser melhor tratados em um estágio mais avançado [LAKIN, 1989].

2.2.3 Baseada em Modelo de Dispositivo

As explicações da concepção seriam produzidas usando o modelo para simular o comportamento de um dispositivo, tais como Tablet PCs, lousas eletrônicas, laptops, entre outros. Contempla a obtenção e o armazenamento do modelo de comportamento do dispositivo; um modelo do próprio dispositivo é utilizado para gerar explicações textuais e gráficas sobre como e porque um dispositivo funciona de determinada maneira. Permite ao usuário visualizar o modelo e fazer perguntas sobre seu design e comportamento [GRUBER, 1990].

Utiliza um modelo para explicar o projeto e assume que estas explicações sobre os dispositivos são suficientes para todos os esclarecimentos necessários. É mais poderoso e específico ao domínio que o baseado em argumentação e histórico. É baseado em muitas das técnicas e hipóteses dos sistemas especialistas baseados em modelo de dispositivo, principalmente os de diagnóstico [GARCIA, 1993].

Esta abordagem foi mais bem sucedida para diagnóstico do que em projetos propriamente ditos. Na atividade de diagnóstico, o modelo para o sistema composto de dispositivos é considerado como completo. Por outro lado, na maioria dos projetos, algumas partes do sistema não são especificadas [GARCIA, 1993].

2.2.4 Baseada em Documentos Ativos

O Design Rationale baseado em documentos ativos é gerado com base no conhecimento previamente guardado, não apenas sobre o projeto em questão, mas sobre outros projetos. A cada decisão que é tomada, o sistema compara a decisão do usuário com a decisão que seria tomada com base no conhecimento armazenado. Se a decisão do usuário for conflitante com a recomendação do sistema, o sistema avisa ao usuário que poderá alterar sua decisão ao modificar algum critério da base de conhecimento [BURGE, 1998].

2.3 Modelos de Representação do Design Rationale

Desde a década de 80, engenheiros de software começaram a adaptar as primeiras abordagens as suas necessidades no projeto de software. Atualmente, existem diversas abordagens, sendo que a maioria difere apenas na nomenclatura dos nós e de seus relacionamentos.

Cada modelo utiliza uma nomenclatura própria para se referir a estas entidades. No entanto, os quatro modelos de DR apresentados modelam o espaço solução relacionando argumentos às alternativas de solução. IBIS e DRL possuem um nodo que define os argumentos para a escolha ou rejeição de uma alternativa. QOC e DRL possuem um nodo que indica os critérios necessários à solução de um problema. Esta característica é importante para uma melhor e mais precisa tomada de decisão [FRANCISCO, 2004], pois estes critérios indicam propriedades desejadas ou requisitos que devem ser satisfeitos.

Os modelos de DR são compostos de elementos e relacionamentos para a representação e registro das razões que motivaram a tomada de determinadas decisões. Em geral, são aplicáveis a vários contextos diferentes. Entretanto, modelos

com um maior número de elementos e alguns deles mais específicos facilitam a modelagem do seu contexto, porém a aplicabilidade destes modelos é menor que dos modelos mais simples.

Assim, à medida que aumenta a quantidade de elementos semanticamente diferentes em uma abordagem de DR, aumenta a sobrecarga cognitiva para quem pratica a modelagem de raciocínios ou quem faz uso dela. Isto impõe uma limitação sobre o poder expressivo de qualquer formalismo adotado na representação e captura de raciocínios e decisões [SHUM, 1994].

De acordo com objetivo do modelo, é classificado como descritivo ou prescritivo. No descritivo, o objetivo é somente descrever os processos de raciocínio dos projetistas. Não é feita nenhuma tentativa de alterar seu modo de pensar ou mesmo a decisão tomada. Entretanto, as informações guardadas podem melhorar processos de outras fases do ciclo de desenvolvimento de software, como, por exemplo, a implantação, manutenção ou reuso de artefatos de projeto. O Design Rationale pode também ser utilizado para transferência de conhecimento a novos integrantes da equipe [DUTOIT, 2006].

A abordagem prescritiva, por outro lado, tem o objetivo de melhorar os processos durante a fase de elaboração do projeto através do aperfeiçoamento do modo de pensar dos envolvidos. Procura-se corrigir deficiências percebidas no raciocínio de questões do projeto tornando-o mais correto, mais consistente e mais completo. Nesta abordagem também se pode criar registros de Design Rationale para serem utilizados no auxílio a processos fora da fase de elaboração do projeto. Deve-se notar também que descritivo e prescritivo não são sempre mutuamente exclusivos. Por exemplo, algumas abordagens têm a intenção principal de serem descritivas, mas também têm alguns objetivos prescritivos [DUTOIT, 2006].

A seguir descreveremos os modelos de DR mais utilizados atualmente, segundo grande parte dos pesquisadores, incluindo Stumpf (1997) e Shum (1991).

2.3.1 Issue Based Information System (IBIS)

Historicamente, o movimento de Design Rationale iniciou-se com o Issue Based Information System (IBIS), descrito por Kunz e Rittel. Tinha o intuito de fornecer suporte a grupos que eram confrontados com problemas complexos [KUNZ, 1970]. O sistema guiava a identificação, estruturação e decisão de questões levantadas em grupos de resolução de problemas e fornecia informação apropriada à discussão, criando um plano de decisão. Não estava relacionado à software, mas sim a um modo de modelar a argumentação em geral.

A Abordagem foi desenvolvida para capturar decisões de projeto durante o progresso do projeto. Nesta abordagem, possíveis soluções (“Posição”) para uma questão (“Tema”) são apresentadas e julgadas em seus prós e contras. IBIS foi a primeira representação explícita para raciocínio em um contexto de design, de onde surgiu o termo Design Rationale [SHUM, 1991].

Uma vantagem desta abordagem é de ser simples e intuitiva. A captura começa a partir da discussão de um tema, aos quais são apresentadas posições. Posições recebem argumentos de dois tipos, prós ou contras. Finalizando, uma posição pode derivar novos temas, que iniciam outros ramos com a mesma estrutura. A Figura a seguir mostra um diagrama IBIS simples:

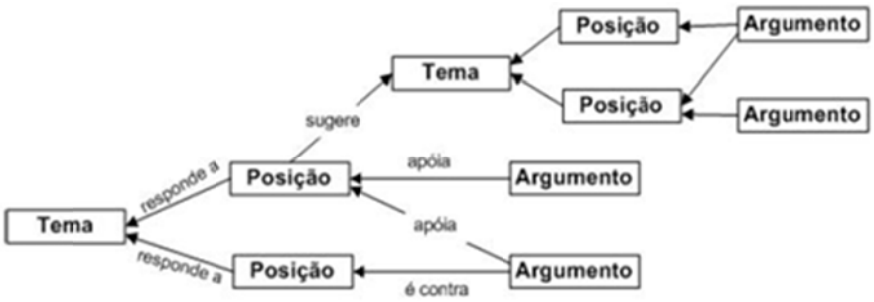

Figura 2.1 - Estrutura do modelo IBIS [KUNZ, 1970]

Uma discussão em torno do IBIS começa com o tema principal, que é a principal questão a ser respondida. Posições são obtidas para esta questão juntamente com argumentos a favor e contra as posições. Temas secundários, terciários, etc., podem ser gerados e tratados da mesma maneira, até que uma solução seja encontrada.

Após a solução ter sido encontrada, os temas são colocados em um diagrama chamado de mapa de temas, que representa suas relações. Várias outras relações entre temas são reconhecidas na técnica original do IBIS como, por exemplo, ‘mais geral que’, ‘similar a’, ‘substituto’, ‘sucessor temporal de’ e ‘sucessor lógico de’.

A grande vantagem do modelo IBIS é que o Design Rationale pode ser capturado de maneira informal em um estágio inicial do processo, assegurando que as questões do projeto estão entendidas. É considerado um modelo de discussão muito poderoso, podendo também ser utilizado por usuários que trabalham sozinhos. Eles afirmam que a organização de ideias os ajuda a prestar mais atenção às partes mais complexas e críticas do problema, além de ajudar a identificar a falta de consistência dos pensamentos mais rapidamente.

O modelo IBIS é considerado complexo o suficiente para ser capaz de lidar com diversos tipos de problemas, e simples o suficiente para ser prático no cenário de captura. A simplicidade do IBIS é percebida mesmo para aqueles que estão na fase de aprendizagem. A maioria dos comentários em uma discussão analítica são perguntas e respostas, portanto, é possível mapear grande parte dessas interações utilizando apenas estes dois elementos desta abordagem.

Para os usuários que trabalham em equipe, a estrutura utilizada na discussão é muito útil e serve para mostrar opiniões pessoais, sinalizações de concordância e tornam as suposições e definições mais claras [CONKLIN, 1988].

Entretanto, por ser um modelo simples, com apenas três tipos de nós, muitas informações acabam sendo representadas em um único nó [MONK, 1995]. O nó ARGUMENTO, por exemplo, armazena também requisitos, restrições e objetivos.

Além disso, por não ter uma representação hierárquica, gera uma rede complexa quando armazena muitos nós [FRANCISCO, 2004]. Consequentemente, o desempenho da busca de informações fica comprometido. O modelo IBIS não pode fazer uso de uma exploração estruturada dos assuntos. O controle fica muito mais nas mãos do projetista e demais pessoas envolvidas que devem investigar maneiras factíveis de resolver cada problema.

Por um lado, IBIS é considerado intuitivo. Por outro lado, é restrito pelo contexto local e pela falta de objetivos explícitos e de resultados da argumentação [NGUYEN, 1998].

O modelo IBIS tem o intuito de fornecer suporte a grupos que eram confrontados com problemas complexos [KUNZ, 1970], sendo classificado como modelo prescritivo, detalhado para fornecer informações a outras fases do ciclo de desenvolvimento do artefato.

2.3.2 O Modelo Potts and Bruns (PB)

Apresentado em 1988 por Potts & Bruns, esse modelo tem o propósito de representar o processo de deliberações que conduz a um dado design e os designs intermediários que resultam neste processo. No modelo Potts and Bruns, a história de design é formada pela rede de designs intermediários representados pelos artefatos (especificações ou documentos de design), que são derivados uns dos outros através dos nós de deliberação baseados em IBIS, representados como temas, alternativas e justificativas. Os artefatos produzidos dependem do método de design de software que está sendo apoiado.

A grande diferença entre o novo modelo e o modelo IBIS é a caracterização da associação dos temas discutidos aos artefatos relacionados. Em outras palavras, os elementos desta abordagem não são exclusivamente de raciocínios, caracterizando-se como uma abordagem híbrida. Cria-se assim uma história cognitiva com associação entre os elementos do artefato relacionados. Outra diferença para o IBIS é que, ao invés de ter elementos de argumentação para representar prós e contras, esta abordagem representa a argumentação em um único elemento, chamado de "Justificativa". Este elemento agrega o raciocínio que levou a sugestão de uma determinada solução.

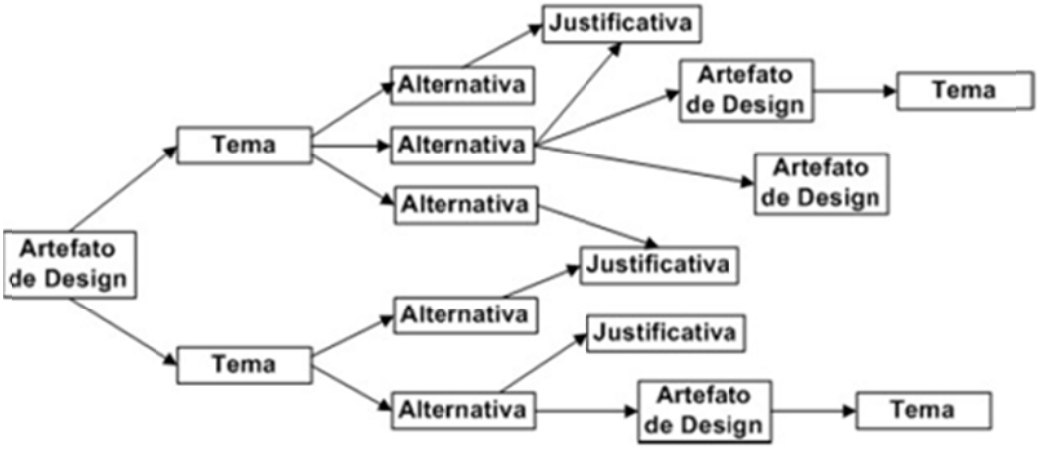

Figura 2.2 - Estrutura do modelo Potts and Bruns [POTTS, 1998]

Um aspecto importante a ser destacado nesta abordagem (Figura 2.2) é a associação do artefato tanto na deliberação do tema discutido quanto na derivação de alternativas de design. Dessa forma, artefatos modificados ou novos artefatos gerados são acomodados em seu vocabulário.

A ideia principal do modelo Potts and Bruns é a integração de entidades dos métodos de design existentes com a deliberação de design baseada em IBIS. Esta integração é a diferença chave entre este modelo e os outros esquemas de representação para Design Rationale. Nela, as entidades do modelo genérico de Potts & Bruns são refinadas para acomodar o vocabulário de um método de design particular para derivar novos artefatos. Por exemplo, uma nova entidade, específica de um método de design, é incorporada ao modelo IBIS dando origem a um novo tema na deliberação do design. Assim, o modelo Potts and Bruns pode ser integrado a diferentes métodos de design.

Potts & Bruns apresentaram um exemplo do design de um formatador de texto usando o método de Liskov & Guttag (1986) para ilustrar essa integração. Neste exemplo, uma nova entidade tarefa, específica do método de Liskov & Guttag, é

incorporada ao modelo IBIS dando origem a novos temas na deliberação do design do formatador de texto. As transformações dos artefatos através dos nós de deliberação foram representadas usando o sistema hipertexto generalizado PlaneText (Gullichsen et al., 1986), cuja estrutura era verificada por regras Prolog que "conheciam" a sintaxe do método de Liskov & Guttag.

O modelo PB tem o propósito de representar o processo de deliberações que conduz a um dado design, e os designs intermediários que resultam neste processo [POTTS, 1988], portanto, é classificado como modelo descritivo.

2.3.3 Decision Representation Language (DRL)

A DRL (Decision Representation Language) foi inicialmente desenvolvida como uma linguagem para representar o processo de tomada de decisão, sendo que, posteriormente, foi estendida para a representação de Design Rationale [LEE, 1991].

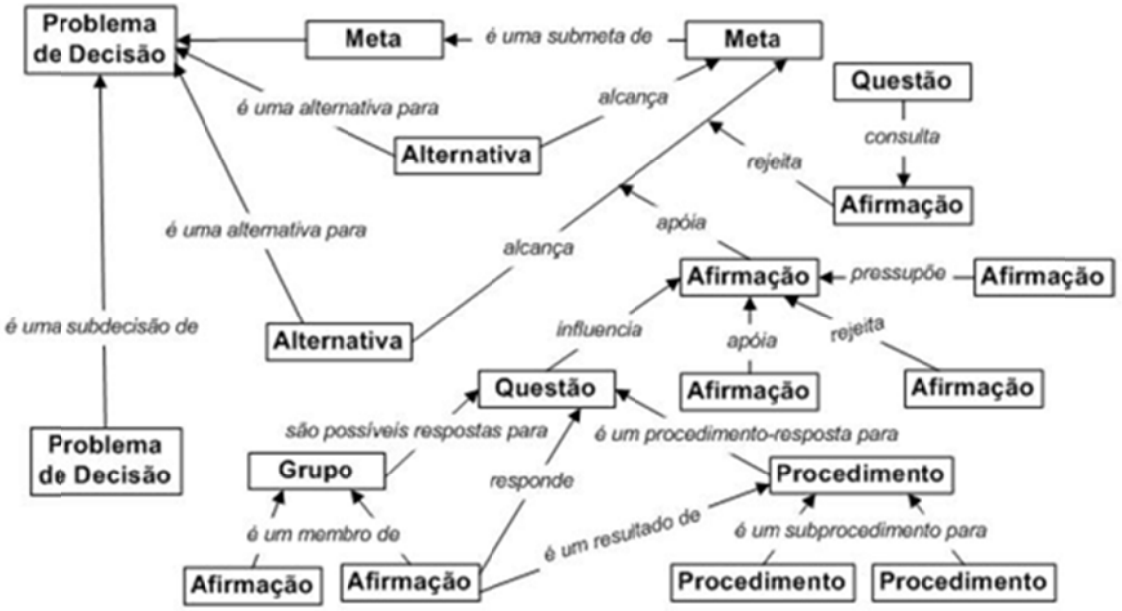

Figura 2.3 - Estrutura do modelo DRL [LEE, 1991]

A DRL é uma extensão tanto do Potts and Bruns quanto do IBIS, mais completa e expressiva. O foco desta linguagem de representação está em gerenciar o peso para uma decisão de design particular de uma maneira consistente e confiável. Os elementos fundamentais do modelo DRL são metas, alternativas e afirmações. As alternativas representam as opções de escolha. As metas especificam as propriedades da situação ideal e as afirmações constituem argumentos relevantes para a tomada de decisão.

A representação deve suportar um certo número de tarefas de projeto, tais como responder questões sobre o progresso do projeto, as alternativas geradas, as avaliações que levaram à escolha de determinadas alternativas e a possível transferência de conhecimento a projetos futuros ou outras pessoas. DRL foi desenvolvido para suportar todas estas questões. Sua ênfase é gerenciar os elementos qualitativos das tomadas de decisão e do gerenciamento de suas dependências. Portanto, é mais um sistema de gerenciamento de Design Rationale [LEE, 1991].

A DRL não inclui deliberações sobre como gerar alternativas de projeto, pois sua ênfase é no gerenciamento de elementos qualitativos da tomada de decisão e gerenciamento de dependências, sendo, desta forma, mais do que um sistema de representação da tomada de decisão [STUMPF, 1998].

Este método esgota a avaliação de alternativas através da referência de objetivos explícitos que capturam os objetivos do processo de projeto, ao invés de se concentrar na exploração do espaço de projeto. Este método não beneficia o processo de exploração de um problema mas, sim, o gerenciamento do peso de uma decisão em um dado projeto, de forma correta e consistente [LEE, 1991].

O modelo DRL foi desenvolvido como uma linguagem para representar o processo de tomada de decisão, sendo que, posteriormente, foi estendido para a representação de design rationale [LEE, 1991], sendo classificado como modelo descritivo.

2.3.4 Questions, Options and Criteria (QOC)

O Modelo QOC (Questions, Options and Criteria) é basicamente um modelo hierárquico onde Questões referem-se às principais decisões a serem tomadas no decorrer do desenvolvimento; Opções referem-se a possíveis alternativas a considerar para cada decisão; e Critérios referem-se a razões para adoção (critério positivo) ou rejeição (critério negativo) de cada uma das alternativas enumeradas em Opções.

O Modelo QOC é uma notação semiformal, baseada em argumentação, criada para explicar porque um design do artefato foi escolhido dentre um espaço de possibilidades. Esta abordagem está focada nos três conceitos básicos indicados em seu nome: (Q) questões identificam os principais temas para estruturar o espaço de alternativas; (O) opções fornecem as possíveis respostas a estas questões, e (C) critérios formam as bases para avaliação e escolha entre as opções [MACLEAN, 1991].

Este modelo destaca a exploração de um espaço de alternativas do projeto, o espaço do projeto e as escolhas realizadas. É uma abordagem que utiliza notação

semiformal baseada em argumentação para representar, sistematicamente, visões do espaço de projeto. O espaço de projeto é um conjunto de relacionamentos ou dimensões conceituais, usado para comparar projetos e soluções alternativas de instanciar estes conceitos [SHUM, 1991]. Segundo Shum, o espaço de projeto nunca poderá ser representado em sua totalidade, pois podem surgir questões até um nível infinito de detalhamento e de numerosas perspectivas [SHUM, 1996].

O modelo apresenta uma estrutura baseada no processo de como as alternativas são geradas e avaliadas. Esta organização possibilita a representação explícita de um espaço estruturado de alternativas de projeto e as considerações que levaram a sua escolha. Dentro das opções do espaço de projeto há possíveis respostas às perguntas [JARCZYK, 1992].

O modelo desenvolvido inclui três tipos de nodos: QUESTÕES, OPÇÕES e CRITÉRIOS. As QUESTÕES são os problemas chave a resolver, as OPÇÕES são as alternativas levantadas para resolver os problemas identificados e os CRITÉRIOS justificam as opções existentes. Além destes elementos existem também os julgamentos (assessments), que são os relacionamentos entre OPÇÕES e CRITÉRIOS. Em sua forma mais simples, os julgamentos podem possuir dois tipos de relacionamentos: SUPORTA ou OPÕE-SE A. Os ARGUMENTOS são usados para conduzir a discussão sobre a situação destas entidades e seus relacionamentos. A Figura 4 mostra a estrutura geral do modelo QOC.

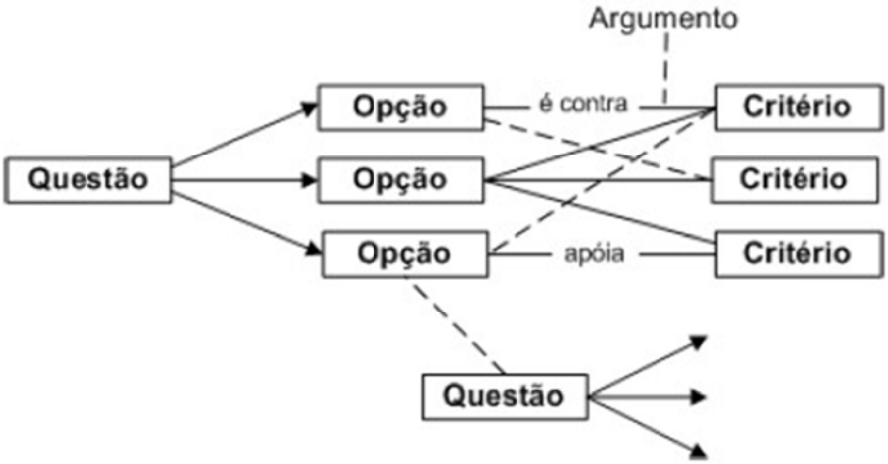

Figura 2.4 - Estrutura da notação QOC [MACLEAN, 1991]

Em resumo, o modelo visa identificar problemas chave (Questões) e levantar e justificar (via Critérios) alternativas de projeto (Opções). Uma diferença do QOC em relação ao IBIS é o armazenamento dos critérios, possibilitando a captura explícita das restrições do projeto e facilitando a resolução do problema.

Segundo [HU, 2000], QOC é simples de aprender e utilizar e há um número crescente de projetos de pesquisa no assunto. Outra vantagem destacada pelos autores é a relativa facilidade de criar um QOC para a engenharia reversa de parte do sistema e preservá-la para uso futuro.

O modelo QOC é fundamentalmente descritivo, visto que sua principal finalidade no sistema é criar uma representação das decisões dos projetistas que seja suficientemente para representar as decisões suas justificativas e alternativas consideradas.

2.3.5 Comparação entre os modelos

Atualmente, existem diversos modelos de DR que permitam registrar explicitamente as estruturas de raciocínio, a maioria deles diferem apenas na nomenclatura e na semântica dos nodos e de seus relacionamentos.

Os quatro modelos apresentados são baseados em argumentação e possuem notação semiformal. Eles diferem na extensão de características que almejam capturar e nas funções que suportam o modo como estes modelos são utilizados em uma representação.

A tabela 2.1 apresenta um comparativo das características dos quatro modelos vistas no capítulo anterior.

Tabela 2.1 - IBIS, PB, DRL e QOC: Características principais.

	IBIS	PB	DRL	QOC
Ano de criação	1970	1988	1991	1991
Criador	Kunz e Rittel	Potts & Bruns	Lee e Lai	McLean
Tipo	Argumentativo	Argumentativo	Argumentativo	Argumentativo
Representação	Semiformal	Semiformal	Semiformal	Semiformal
Objetivo	Prescritivo	Descritivo	Descritivo	Descritivo

A Tabela 2.2 apresenta a nomenclatura utilizada pelos autores de cada modelo para nomear os Elementos de Representação e Ligação de cada modelo estudado.

Tabela 2.2 - IBIS, PB, DRL e QOC: Elementos de Representação e Ligação

	IBIS	PB	DRL	QOC
Elementos de Representação	• Tema • Posição • Argumento	• Artefato de Designe • Tema • Alternativa • Justificativa	• Problema de Decisão • Meta • Alternativa • Questão • Grupo • Afirmação • Procedimento	• Questão • Critérios • Opções
Elementos de Ligação	• Responde a • Sugere • Prós • Contras	• Relação	• Influência, • Alcança, • Apoia, • Consulta, • É um membro de, • É um resultado, • É um subprocedimento para, • É um procedimento-resposta para, • É uma alternativa para, • É uma subdecisão de, • É uma submeta de, • Pressupõe, • Rejeita • Responde, • São possíveis respostas para	• Relação • Prós • Contras

A Tabela 2.3 apresenta a nomenclatura de cada modelo para nomear os elementos relativos ao Espaço do problema inicial, Espaço para subproblemas, Espaço para possíveis soluções, Espaço para argumentos e Espaço da decisão.

Tabela 2.3 - IBIS, PB, DRL e QOC: Nomenclatura

	IBIS	PB	DRL	QOC
Espaço do problema inicial	Tema	Artefato de Designe	Problema de Decisão	Questão
Espaço para subproblemas	Tema	Tema	Questão	Questão
Espaço para possíveis soluções	Posição	Alternativa	• Alternativa • Procedimento • Meta • Grupo	Opção
Espaço para argumentos	Argumento	Justificativa	Afirmação	Critério
Espaço da decisão	Posição Escolhida	Alternativa	Objetivo	Opção Escolhida

Apesar de ser possível representar estes julgamentos no modelo IBIS, através dos argumentos, IBIS não possui uma representação explícita dos critérios como elementos do modelo. [NGUYEN, 2000] ainda ressaltam que cada alternativa é avaliada isoladamente por seus próprios argumentos.

Não há um conjunto comum de argumentos a todas as alternativas. Outra limitação do modelo IBIS é a geração de uma rede complexa, quando armazena muitos nós, dificultando a busca de dados [FRANCISCO, 2004].

Um aspecto importante a ser destacado do modelo PB possuir elemento para representar objetos do domínio do problema, permitindo a associação do artefato tanto na deliberação do tema discutido quanto na derivação de alternativas de design. Dessa forma, artefatos modificados ou novos artefatos gerados são acomodados em seu vocabulário.

O modelo DRL é bem mais completo, envolvendo um número de nodos e relacionamentos bem maior que os outros modelos. Um dos objetivos foi o aumento

da expressividade e funcionalidade. Lee e Lai (1996) argumentam que o DRL é mais expressivo que os outros modelos de argumentação, porque atende a uma faixa mais ampla de questões que podem surgir em variadas fases do ciclo de vida de um artefato. Por outro lado, para contemplar estas características, houve um aumento da complexidade [LOURIDAS, 2000].

Para [DUTOIT, 2006] duas características, em resumo, distinguem QOC de IBIS. A primeira refere-se à abrangência das questões. Enquanto IBIS pode referir-se a qualquer tópico de projeto, as questões do modelo QOC tratam exclusivamente de características do artefato que está sendo projetado.

Por isso, as questões do modelo QOC sempre têm possíveis respostas (OPÇÕES), que descrevem as propriedades do artefato projetado. O segundo fator que distingue os dois modelos é que o QOC utiliza julgamentos para indicar se as alternativas cumprem cada critério definido.

Os modelos IBIS, PB e DRL possuem um número maior de tipos de Nodos e de relacionamentos, tornando a modelagem mais precisa, mas seu entendimento mais complexo.

Tabela 2.4 - IBIS, PB, DRL e QOC: Objetivos e complexidade.

	IBIS	**PB**	**DRL**	**QOC**
Numero de Elementos	7	4 do Modelo + Artefatos de Designe Derivados	22	5
Precisão da Modelagem	Mais precisa	Mais precisa	Mais precisa	Menos precisa
Complexidade da Modelagem	Mais complexa	Mais complexa	Mais complexa	Mais Simples

A Tabela 2.5 mostra um resumo das vantagens e limitações de cada modelo. A melhor escolha dependerá da aplicação e objetivos específicos do ambiente analisado.

Tabela 2.5 - IBIS, PB, DRL e QOC: Vantagens e limitações.

	IBIS	PB	DRL	QOC
Pontos fortes	• Modelo Simples • Auxílio às primeiras tarefas do processo • Trata questões sobre qualquer tópico do projeto	• Modelo Simples • Abordagem híbrida suporte a representação de Artefato de Designe do domínio do problema;	• Representação expressiva; • Captura de relações complexas	• Modelo Simples • Mostra explicitamente os critérios
Limita-Coes	• Não possui nodo representando os critérios • Rede complexa quando armazena muitos nós	• Pode ser integrado a diferentes métodos de design	• Maior complexidade para entendimento	• Trata apenas questões sobre o artefato • Auxílio apenas a outras fases do projeto
Análise	Preterido por não ter elemento para representar os critérios	Preterido pela complexidade da modelagem	Preterido pela complexidade da modelagem	Foi o modelo escolhido

Por outro lado, QOC possui um número maior de tipos de Nodos e não modela as informações tão precisamente quanto os outros modelos estudados, mas é uma abordagem mais simples para a modelagem, mais fácil de ser entendido e por essa razão foi o modelo escolhido para este trabalho.

2.4 Ferramentas de QOC

Como explicado no Capítulo 1, a principal dificuldade em adotar algum modelo de Design Rationale é a captura das informações. Não se pode esperar que o projetista saiba expressar estas informações sem o auxílio de uma ferramenta que o guie [GRUBER, 1991].

Existem na literatura algumas propostas de extensões do modelo QOC para atender alguns domínios de aplicação, tais como: TEAM [LACAZE, 2005], DQN [BRAMWELL, 1995], QUIDS [HORDIJK, 2004] e QOC-E [VINCENT, 2012].

Porém, não há nenhuma ferramenta adequada para Design Rationale que faça uso eficaz da notação para suportar as atividades exigentes, tais como armazenamento e recuperação da informação e sua reutilização [LACAZE, 2010].

A seguir veremos a Ferramenta DREAM que é a única ferramenta disponível para o modelo QOC.

2.4.1 Ferramenta DREAM

A ferramenta DREAM (Design Rationale Environment for Argumentation and Modelling) é uma ferramenta de edição gráfica de diagramas de QOC usando a notação TEAM.

A DREAM implementa o modelo proposto por Farenc Christelle e Palanque Philippe, em artigo publicado em 1999, que propõe uma notação ferramenta de suporte para lidar com problemas de modelagem e cobertura de requisitos durante o processo de desenvolvimento de sistemas interativos.

A notação TEAM (Traceability, Exploration and Analysis Mode), proposta por [LACAZE, 2005], é uma extensão do modelo QOC a fim de lidar com as especificidades dos sistemas interativos críticas de segurança e, também, para resolver alguns problemas de no uso do modelo de QOC. A justificativa para estender QOC é baseado, principalmente, em sua simplicidade e legibilidade, que o torna compreensível pela maioria dos atores envolvidos no projeto de sistemas interativos, tais como: designers gráficos, programadores, clientes, autoridades de certificação, etc. [LACAZE, 2006]

Uma das extensões propostas, a inclusão das entidades “modelos de tarefas” e “cenários”, tratam explicitamente dos modelos de tarefas e dos cenários que correspondem aos caminhos de execução nestes modelos. A análise e modelagem de tarefa são elementos críticos em métodos de design centrados no usuário, comumente usados na engenharia de sistemas interativos. Outras extensões, visando à usabilidade da notação, foram implementadas. Por exemplo, a representação dos requisitos expressos pelos usuários como “fatores” e a definição de métricas nos elos entre opções e critérios. [LACAZE, 2006]

A ferramenta DREAM foi desenvolvida pela equipe composta por Xavier Lacaze, Eric Barboni e Célia Martinie, além de outras pessoas que contribuíram para o projeto através de sugestões, correções ou auxiliando na documentação. Foi utilizada a linguagem Java para plataforma desktop.

A ferramenta DREAM permite para a edição, análise e exploração de diagramas do modelo QOC usando a notação TEAM. DREAM apoia o gerenciamento das equipes de trabalho e seções, tratando as decisões de acordo com as pessoas responsáveis por elas e, também, de acordo com o momento no qual essas decisões foram tomadas. A exploração de modelos permite a reutilização de elementos em outros projetos.

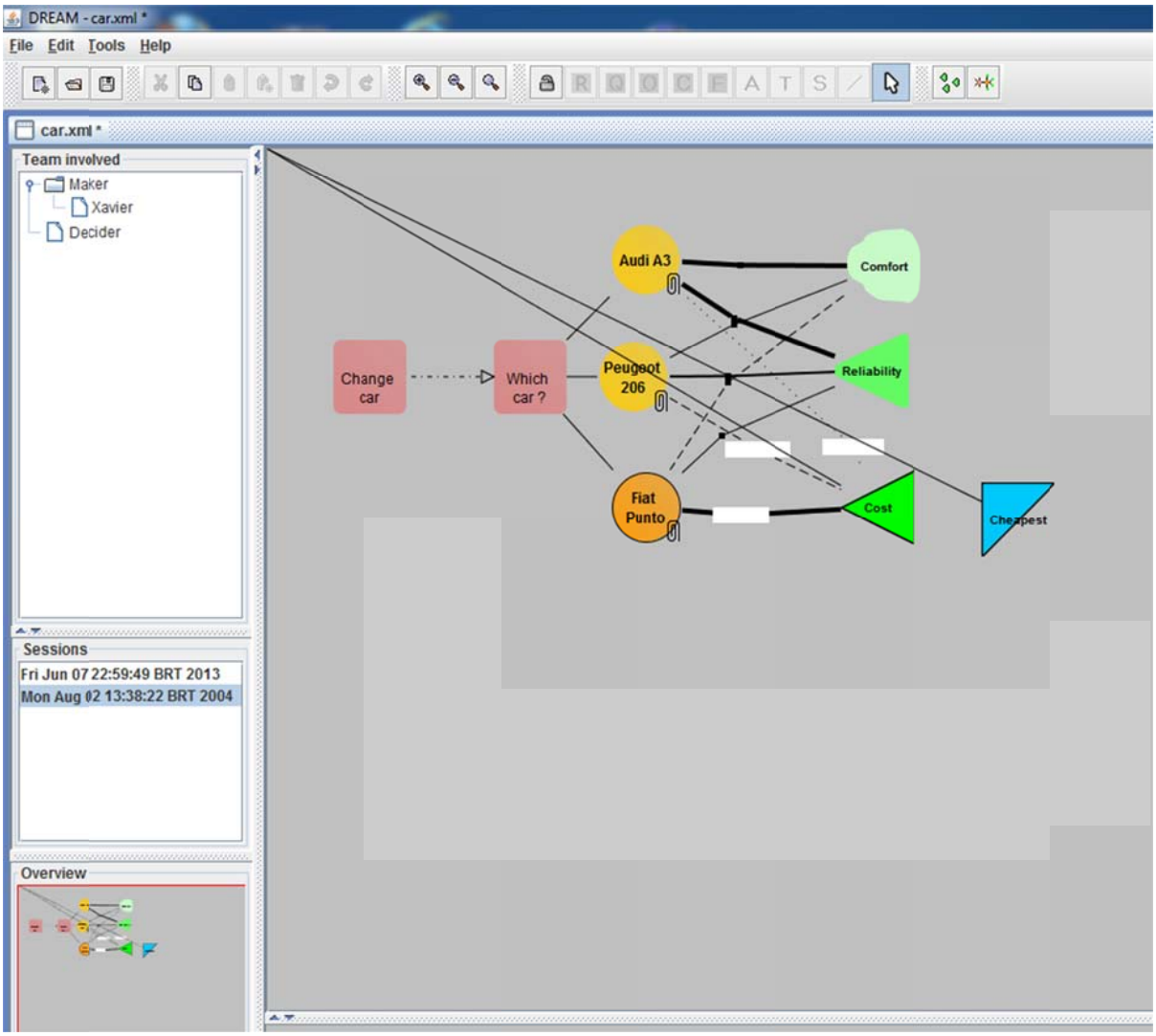

Figura 2.5 – Interface da ferramenta DREAM

A ferramenta também suporta a gestão do trabalho em equipe e sessões, apoiando assim a rastreabilidade das decisões de acordo com as pessoas que as fazem e de acordo com o momento em que essas decisões foram tomadas. Dream está disponível e pode ser baixado na seguinte página web: http://liihs.irit.fr/dream.

3. DEFININDO O QOC – EDITOR

Inicialmente, devemos definir de forma sistemática os requisitos do sistema envolvendo os futuros usuários da aplicação, de forma a ter todos os requisitos principais definidos de forma clara e objetiva.

Requisitos são objetivos ou restrições estabelecidas pelos usuários que definem as diversas propriedades do sistema. Os requisitos expressam as características e restrições do produto de software do ponto de vista de satisfação das necessidades do usuário e, em geral, independem da tecnologia empregada na construção da solução, sendo a parte mais crítica e propensa a erros no desenvolvimento de software.

A necessidade de se estabelecer os requisitos de maneira precisa é crítica para o projeto do software, já que na medida em que requisitos aumentam, também aumenta o tamanho e a complexidade do software. Os requisitos definem a arquitetura que será escolhida e exercem influência uns sobre os outros.

3.1 Levantamento dos Requisitos

O projeto da arquitetura de software é uma etapa essencial no desenvolvimento de sistemas de software, já que a arquitetura de software é a base sobre a qual funcionalidades serão concebidas e implementadas, a partir dos recursos oferecidos pela arquitetura. Entretanto, o levantamento dos requisitos do sistema antecede a etapa de projeto da arquitetura de software, pois é necessário definirmos com clareza as funcionalidades antes de projetá-las, processo conhecido como elicitação de requisitos.

A elicitação de requisitos é uma as fases iniciais do processo de desenvolvimento, onde é definido o conjunto de requisitos. Tal conjunto de requisitos é visto como uma especificação do que deveria ser implementado. Os requisitos são descrições de como o sistema deveria se comportar e contém informações do domínio da aplicação e restrições sobre a operação do sistema.

Nessa fase, o projetista de software levanta os requisitos, buscando identificar características do sistema a ser desenvolvido. Além disso, informações do domínio do problema juntamente com informações de estilos arquiteturais existentes podem ser utilizados como fontes de dados que auxiliam na identificação dos requisitos.

3.1.1 Requisitos Funcionais

Como definiu ROMAN, um requisito funcional define uma função de um sistema de software ou seu componente. Uma função é descrita como um conjunto de entradas, seu comportamento e as saídas. Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar. Requisitos comportamentais, que descrevem todos os casos em que o sistema utiliza os requisitos funcionais, são extraídos dos casos de uso.

Já [THAYER, 1990] afirma que requisito funcional é um requisito de sistema de software que especifica uma função que o sistema ou componente deve ser capaz de realizar. Estes são requisitos de software que definem o comportamento do sistema, ou seja, o processo ou transformação que componentes de software ou hardware efetuam sobre as entradas para gerar as saídas. Esses requisitos capturam as funcionalidades sob o ponto de vista do usuário.

Pela Aplicação ser um editor gráfico, requer um conjunto de funcionalidades básicas, outras funcionalidades são necessárias para atender os requisitos de usabilidade que foram definidas na fase de experiência do autor e seu orientador, como segue na Tabela 3.1.

Tabela 3.1 – Requisitos: Funcionalidade e Motivação

FUNCIONALIDADE	**MOTIVAÇÃO**
1. Modo Gráfico de edição com suporte a teclado e mouse	Aproveitar o conhecimento dos usuários do estilo de interação clássico do Windows com teclado e mouse, com interação de Manipulação Direta onde Usuário manipula diretamente a representações dos objetos.
2. Modo Lista de Gráficos	Permitir a edição do texto e posicionamento dos Nodos com mais eficiência.
3. Modo Tabela de Nodos relacionados por Relações	Identificar mais facilmente os relacionamentos entre Nodos
4. Criar Nodos representando Início, Questões, Opções e Critérios, diferenciados visualmente	Funcionalidade básica do modelo de interação de Manipulação Direta

5. Selecionar Nodos	Funcionalidade básica do modelo de interação de Manipulação Direta
6. Mover Nodos	Funcionalidade básica do modelo de interação de Manipulação Direta
7. Excluir Nodos	Funcionalidade básica do modelo de interação de Manipulação Direta
8. Criar Arcos, conforme a lógica do modelo QOC, representando Relações Positivas e Relações Negativas diferenciadas visualmente ligando os Nodos	Funcionalidade básica do modelo de interação de Manipulação Direta
9. Excluir Arcos	Funcionalidade básica do modelo de interação de Manipulação Direta
10. Inserir, Editar e Apagar o Texto nos Nodos	Funcionalidade básica do modelo de interação de Manipulação Direta
11. Marcar a opção escolhida	Funcionalidade básica do modelo de interação de Manipulação Direta
12. Salvar as Propriedades do Projeto	Armazenar os metadados do projeto
13. Pesquisa nos Nodos	Facilitar a localizar de Nodo
14. Salvar um Diagrama	Permitir o armazenamento, reedição, uso futuro dos Diagramas
15. Carregar um Diagrama salvo	Permitir o armazenamento, reedição, uso futuro dos Diagramas
16. Exportar um Diagrama para imagem	Permitir uso dos Diagramas por outros programas
17. Imprimir o Diagrama.	Imprimir os Diagramas em papel.
18. Configurar a Área de Trabalho	Permitir configurar da área de trabalho
19. Configurar a Área de	Permitir configurar da área de desenho

Desenho	
20. Configurar a Impressão	Permitir configurar da área de desenho
21. Visualizar a Impressão	Permitir Visualizar a Impressão
22. Desfazer uma ação	Permitir Desfazer uma ação
23. Refazer uma ação	Permitir Refazer uma ação
24. Copiar, Recortar e Colar Nodos ou Arcos	Permitir Copiar, Recortar e Colar Nodos ou Arcos
25. Duplicar Nodos ou Arcos	Permitir Duplicar Nodos ou Arcos
26. Alinhar Nodos Horizontalmente	Melhorar a usabilidade
27. Alinhar Nodos Verticalmente	Melhorar a usabilidade
28. Distribuir Uniformemente Nodos Horizontalmente	Melhorar a usabilidade
29. Distribuir Uniformemente Nodos Verticalmente	Melhorar a usabilidade
30. Selecionar todos os gráficos	Melhorar a usabilidade
31. Selecionar os gráficos individualmente pelo mouse	Melhorar a usabilidade
32. Exibir um Diagrama em tela cheia	Permitir a apresentação de um Diagrama em tela cheia
33. Exporta Diagrama para arquivo de imagem	Permitir uso dos Diagramas por outros programas
34. Exporta Diagrama para arquivo XML	Permitir uso dos Diagramas por outros programas
35. Exporta Diagrama para arquivo de HTML	Permitir uso dos Diagramas por outros programas

36. Importa Diagrama de arquivo XML	Importa Diagrama de arquivo XML
37. Compartilhar um Diagrama	Compartilhar um Diagrama
38. Configurar elementos Gráficos	Configurar elementos Gráficos
39. Exibir dicas de uso	Exibir dicas de uso
40. Ajuda do programa	Ajuda do programa
41. Exibir barra de status com número de Questões, número de Opções e número de Critérios, Propriedades dos Gráficos Selecionados, exibir posição do mouse	Dar um melhor feedback ao usuário

3.1.2 Requisitos de Desempenho

Requisito não funcional é aquele requisito de software que descreve não o que o sistema fará, mas como ele fará. Assim, por exemplo, têm-se requisitos de desempenho, requisitos da interface externa do sistema, restrições de projeto e atributos da qualidade. A avaliação dos requisitos não funcionais é feita, em parte, por meio de testes, enquanto que outra parte é avaliada de maneira subjetiva.

Como todo editor interativo, busca-se uma boa usabilidade através da facilidade de aprendizado do seu uso, baixo esforço para identificar, entender, aplicar e controlar as funções oferecidas [THAYER, 1990].

Desejamos que a aplicação tenha eficiência através de um bom desempenho, velocidade de execução das funções para um baixo de tempo de resposta, funcionando nos PCs Desktop disponíveis no mercado atualmente.

3.2 Arquitetura do Software

A Definição da arquitetura do software é um ponto crucial em um projeto de desenvolvimento de software, pois a arquitetura é a base para definição dos artefatos, influenciando diretamente na complexidade do projeto. Uma escolha apropriada aumenta as chances de sucesso, mas uma escola inapropriada pode acarretar um desastre.

Carnegie Mellon define a arquitetura de software: "A arquitetura de software de um sistema consiste na definição dos componentes de software, suas propriedades externas e seus relacionamentos com outros softwares. O termo também se refere à documentação da arquitetura de software do sistema. A documentação da arquitetura do software facilita: a comunicação entre os stakeholders registra as decisões iniciais acerca do projeto de alto nível e permite o reuso do projeto dos componentes e padrões entre projetos."

Assim, a arquitetura de software é o conjunto de estrutura e suas iterações e restrições que definem o software. A arquitetura é modulada em um alto nível de funcionalidades do sistema, gerenciamento e distribuição de dados para a plataforma que será usada, servindo como princípios e diretrizes para o projeto e sua evolução.

3.2.1 Modelo de 3 Camadas

Entre os requisitos da aplicação estão a possibilidade de salvar e carregar diagramas feitos no editor, criando a necessidade de separar os dados que representam os diagramas do código do aplicativo que implementam interface e a logica do modelo QOC. Isto torna necessária a escolha de uma arquitetura em camadas.

Por outro lado, para compartilhar os diagramas entre usuários diferentes é necessário que as regras de negócio sejam separadas da camada de interface. Essa separação em camadas lógicas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente, facilitando a manutenção do sistema. Esse modelo de 3 camadas (camada de apresentação, camada de negócio e camada de dados) tornou-se a arquitetura padrão para sistemas corporativos com base na Web [MACORATTI, 2006].

As três partes de um ambiente modelo três camadas são: camada de apresentação, camada de negócio e camada de dados. Deve funcionar de maneira que o software executado em cada camada possa ser substituído sem prejuízo para o sistema. De modo que as atualizações e correções de defeitos podem ser feitas sem

prejudicar as demais camadas. Por exemplo: alterações de interface podem ser realizadas sem o comprometimento das informações contidas no banco de dados [MACORATTI, 2006].

3.2.1.1 Camada de apresentação

É a chamada GUI (Graphical User Interface) ou simplesmente interface. Esta camada interage diretamente com o usuário e é através dela que são feitas as requisições como consultas [AMBLER, 1988].

3.2.1.2 Camada de negócio

Também chamada de Lógica empresarial, Regras de negócio ou Funcionalidade. É nela que ficam as funções e regras de todo o negócio. Não existe uma interface para o usuário e seus dados são voláteis, ou seja, para que algum dado seja mantido deve ser utilizada a camada de dados [AMBLER, 1988].

3.2.1.3 Camada de Dados

A terceira camada é definida como o repositório das informações e as classes que a manipulam. Esta camada recebe as requisições da camada de negócios e seus métodos executam essas requisições em um banco de dados. Alterando o banco de dados altera apenas as classes da camada de dados e o restante das camadas não são afetados por essa alteração [AMBLER, 1988].

3.3 Arquitetura do Projeto

Com a escolha do modelo de 3 camadas temos duas boas arquiteturas possíveis para implementar o modelo: Arquitetura WEB e Arquitetura Desktop.

3.3.1 Arquitetura WEB

Em uma Arquitetura WEB, de um lado está o cliente WEB, ou browser, que solicita dados ao servidor WEB, recebe as respostas, formata a informação e a apresenta ao usuário. Do outro lado, está o servidor WEB que recebe as requisições, lê os dados (paginas HTML) do disco e as retorna para o cliente [MACORATTI, 2006].

3.3.2 Arquitetura Desktop

Na arquitetura Desktop, os programas são autossuficientes. Assim, para seu funcionamento não necessitam de um software auxiliar. Esse tipo de aplicação inclui apresentação, lógica de negócios e acesso aos dados dentro de um único programa [MOUTA, 2010] e é chamada stand-alone, porque não requer um servidor externo. Consequentemente, não existe necessidade de persistência de dados, ou os dados devem ser armazenados em um sistema de arquivos local.

3.3.3 Comparando as Arquiteturas

Para decidirmos a arquitetura, analisaremos alguns fatores considerando quais as vantagens e desvantagens dos dois ambientes para melhor embasar nossa decisão.

Tabela 3.2 – Arquiteturas Desktop e Web: Vantagens e desvantagens

	DESKTOP	**WEB**
VANTAGENS	Grande variedade de controles para interface. Controle total do posicionamento dos controles na aplicação. O Desempenho de uma interface gráfica é mais rápido em uma aplicação desktop que usa o processamento local. Integração com hardware é muito mais fácil. Maior segurança dos dados do cliente que ficam salvos na máquina-cliente.	Interface HTML reconhecida por uma grande gama de usuários já acostumados com o funcionamento dos navegadores. Desenvolvimento, manutenção e atualização centralizada da aplicação. Não há necessidade de instalação da aplicação nos clientes. Facilidade na exportação e compartilhamento dos dados entre usuários remotos. Maior escalabilidade no processamento, pois se houver necessidade de aumentar o poder de processamento, basta fazer isto no servidor.
DESVANT	Necessidade de instalação da aplicação em clientes com diferentes tipos de máquinas, com diferentes tipos de sistemas, drivers e periféricos. A integração com usuários remotos é mais sofrida.	A interface HTML que depende dos recursos do Navegador pode ser um problema, pois não há uma padronização entre os diversos navegadores e sua aplicação poderia ser exibida de uma maneira diferente dependendo do navegador.

AGENS	A manutenção e atualização de sua aplicação requer um esforço extra.	A entrada de uma grande massa de dados é prejudicada na interface HTML, pois não existe uma maneira padrão de criar máscaras de entrada de dados. A interface HTML é pobre em controles gráficos, o visual da sua aplicação é mais limitado e o posicionamento correto dos controles é complicado. A integração com outros componentes não é tão fácil com HTML. A dependência da conectividade com o servidor aumenta o tempo de resposta da aplicação.

Considerando nossos requisitos de desempenho e usabilidade, temos necessidade de uma interface complexa e pela experiência do autor com este tipo de arquitetura, optamos pela plataforma Desktop.

3.4 Ambiente para Desenvolvimento

O Ambiente Integrado para Desenvolvimento de Software - IDE é um programa que possui ambientes para desenvolvimento de software para várias Linguagens de programação(LP), oferecendo recursos semelhantes para as mesmas. As IDE reúnem ferramentas que dão apoio ao desenvolvimento de software, com o objetivo de agilizar o processo.

Atualmente, as tecnologias mais usadas para programação Desktop são as plataformas de software. A Plataforma de software é o conjunto de interfaces que formam uma camada entre a aplicação e o sistema operacional. A Máquina Virtual é o aplicativo da plataforma que compila e executa o código intermediário gerado na compilação.

As grandes vantagens do uso de plataformas são a Portabilidade, já que o Programa pode rodar em qualquer sistema operacional que suporte sua máquina virtual, e o Desempenho, já que a Máquina Virtual otimiza o código de acordo com a máquina que está executando o código.

Na teoria, quaisquer linguagens de programação suficientes são computacionalmente equivalentes. Então, para fazer a comparação temos que

escolher duas implementações práticas. Entre as linguagens de programação mais usadas atualmente estão o Java e o C#. O NetBeans Java é uma boa opção, pertence ao desenvolvedor da plataforma Java e é uma das IDE's mais usadas para a linguagem Java. Outra boa opção é o Visual C#, que é a IDE mais usada para a linguagem C#, pertence ao desenvolvedor da plataforma e já é uma IDE madura.

Do ponto de vista de Engenharia de Software, poderíamos usar 2 IDES para o desenvolvimento do QOC Editor: Visual C# e NetBeans. Ambas IDEs são ótimas soluções. NetBeans e outras IDE's Java têm se firmado em Aplicações Web e Aplicações para dispositivos móveis, e o Visual C# e outras IDE's do .Net para Aplicações Desktop em Geral, como o próprio mercado vem mostrando. Por familiaridade, optamos por usar a linguagem C# para implementar o aplicativo.

3.5 O .NET Framework

O .NET Framework é um componente integrado ao Windows que oferece suporte à criação e execução de aplicativos desktop e serviços Web. Os principais componentes do .NET Framework são o Common Language Runtime (CLR) e a biblioteca de classes do .NET Framework, que inclui ADO.NET, ASP.NET, Windows Forms e Windows Presentation Foundation (WPF), que serão visto em detalhes a seguir [FRAMEWORK, 2013].

O .NET Framework fornece um ambiente gerenciado de execução, desenvolvimento e implantação simplificados, além da integração com diversas linguagens de programação. A documentação do. NET Framework inclui uma vasta referência a bibliotecas de classes, visões gerais, diversos "passo a passo" e informações a respeito dos exemplos, compiladores e ferramentas [FRAMEWORK, 2013].

3.5.1 Arquitetura em Camadas

Como a maioria dos programas desktop desenvolvidos em C#, nosso editor de QOC tem a arquitetura em camadas mostrada na figura 3.1.

O CLR (Common Language Runtime) é o centro do Microsoft .NET Framework; ele fornece o ambiente de execução para todo o código do .NET Framework. O código executado no CLR é chamado de código gerenciado. O CLR fornece diversas funções e serviços necessários para a execução de programas, incluindo a compilação JIT (Just-In-Time), alocação e gerenciamento de memória, imposição de segurança de tipos, tratamento de exceções, gerenciamento de threads e segurança [FRAMEWORK, 2013].

A Figura 3.1 mostra a Arquitetura em Camadas do QOC Editor:

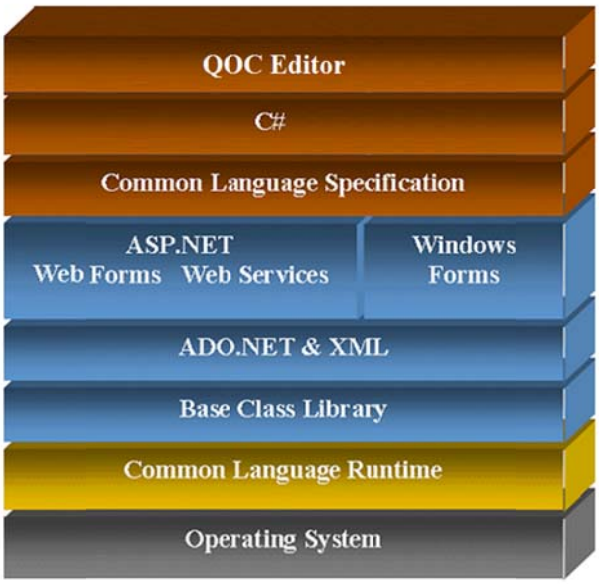

Figura 3.1 - Arquitetura em Camadas do QOC Editor

A Biblioteca de Classes Básicas (Base Class Library) do .NET Framework são as classes, as interfaces, e os tipos de valores que aceleram e otimizam o processo de desenvolvimento e fornecem acesso à funcionalidade do sistema. Os tipos do .NET Framework compõe a base na qual os aplicativos, componentes e controles do . NET são criados. O .NET Framework contém tipos que executam as seguintes funções:

- Representar tipos de dados base e exceções.
- Encapsular estruturas de dados.
- Executar E/S.
- Acessar informações sobre os tipos carregados.
- Invocar as verificações de segurança do .NET Framework.
- Fornecer acesso a dados, uma GUI detalhada do lado do cliente e uma GUI do lado do cliente controlada pelo servidor.

ADO.NET é biblioteca do .NET de manipulação de dados em componentes discretos que podem ser utilizadas separadamente ou em conjunto. Ela fornece códigos consistente de acesso aos dados básicos da biblioteca System.Data, bem como a fontes de dados expostas através de XML, Microsoft SQL Server, OLE DB e ODBC [FRAMEWORK, 2013].

O Windows.Forms é a Biblioteca que contém classes para criar aplicativos usando os componentes e demais recursos da interface avançada de usuário, disponível no sistema operacional Microsoft Windows.

Para facilitar a interoperabilidade entre as diferentes linguagens da plataforma .NET , os tipos são compatíveis com CLS e, portanto, podem ser usados por qualquer linguagem de programação cujo compilador esteja de acordo com a CLS (Common Language Specification) [FRAMEWORK, 2013].

3.5.2 Compilação, Carga E Execução

Programas desenvolvidos na plataforma .Net são compilados duas vezes [FRAMEWORK, 2013]:

- Em Tempo de Compilação: o código fonte gerado pelo programador é então compilado, gerando um código intermediário em uma linguagem chamada MSIL.
- Em Tempo de Carga do Programa: este código MSIL é novamente compilado, executando otimizações, de acordo com máquina em que vai ser utilizado, dessa vez gerando código de máquina (Assembly).

3.5.3 Linguagem C#

C# (CSharp) é uma linguagem orientada a objetos criada pela própria Microsoft e padronizada pela ECMA/ISO.

Faz parte da sua plataforma .NET e é considerada a linguagem símbolo do .NET, porque foi criada praticamente do zero para funcionar na nova plataforma, sem preocupações de compatibilidade com código existente. O compilador C# foi o primeiro a ser desenvolvido. A maior parte das classes do .NET Framework foi desenvolvida em C# [MSDN, 2013].

As principais características de C# são:

- É implementada pela IDE Gráfica do Visual Studio
- Alta Integração com o Windows
- Inspirada no C/C++
- Orientação a Objetos, em C# é orientado a componente (classes com funções básicas), permitindo decompor o programa de acordo com elementos abstratos do domínio do problema, premissa básica para atingir os Critérios de Qualidade de Software: Modularidade, Reusabildade, Extensibilidade, Compatibilidade, Continuidade.
- Usa a plataforma Framework .NET como de bibliotecas de classes ou funções.

- Proteção dos dados através três diretivas de visibilidade: public, private e protected.
- Polimorfismo: Uma mesma função (ou método) pode ser aplicada a objetos diferentes e manter a sua funcionalidade. C# implementa os quatro tipos de Polimorfismo de dados estudados:
- Sobrecarga: C# permite que um mesmo nome denote diferentes funções, de acordo com o contexto. Constitui apenas uma abreviação sintática (um pré-processamento pode atribuir nomes diferentes as diferentes funções).
- Linguagem Fortemente Tipada, é necessário sempre especificar o tipo para declarar uma variável, o que permite que a linguagem implemente com segurança os três tipos de conversão de dados estudados.
- Coersão: é a conversão implícita feita pela própria linguagem, por promoção (upcast) ou alargamento (tipos primitivos menores para maiores), por estreitamento (downcast).
- Instrução Foreach, que implementa um loop de enumeração nas classes do C#. Cada item não nulo de uma coleção é considerado um objeto dentro do loop do Foreach, permitindo buscar objetos de uma classe contidos em um objeto qualquer.
- Objetos não são liberados explicitamente, mas através de um processo de coleta de lixo (garbage collector) quando não há referências aos mesmos, prevenindo assim referências inválidas.
- Herança Simples, ou seja, em C# cada classe só pode herdar apenas outra classe e não mais do que uma. No entanto, é possível simular herança múltipla utilizando interfaces. Assim, através da herança, reduzimos o código através da sua reutilização.
- Coleções e a genéricos (generics).
- Propriedades estão disponíveis, as quais permitem que métodos sejam chamados com a mesma sintaxe de acesso a membros de dados.
- Computação Paralela: Parallel Extensions, Threads, Semáforos e Mensagens.

3.5.4 Visual C#

O Visual C# é um ambiente de desenvolvimento leve, simples e integrado, projetado para desenvolvedores interessados em criar Windows Forms, bibliotecas de classes e aplicativos baseados em console. O Visual C# oferece os recursos de produtividade do Visual Studio para linguagem C# [VISUAL C#, 2013].

Por causa da quantidade de códigos presentes no Framework, o desenvolvimento em VC# é mais rápido se comparado como outras IDE's. Por ser necessário escrever menos códigos e por existir muitas regras de sintaxe, geralmente, o programador comete menos erros de programação. Por outro lado, essa mesma

riqueza de componentes provoca um tempo de aprendizado grande, acarretando em menos mão de obra disponível e, consequentemente, mão de obra mais cara. O Editor Gráfico de Interfaces possui boa usabilidade com cerca de 75 componentes disponíveis na biblioteca System.Forms.

O Visual Studio possui muitos recursos de edição. O recurso de refactor é muito eficiente. Por exemplo, é possível alterar um nome mesmo que o nome já exista ou que o objeto correspondente ao nome tenha sido apagado. O mecanismo de debug é muito simples e eficiente, sendo possível alterar o valor de qualquer variável em qualquer ponto do código. Adicionalmente, o mecanismo de debug é muito eficiente para análises de falhas.

3.6 Projeto da Interface

Tendo definida a arquitetura do projeto e estabelecidos os requisitos e as classes, então faremos o projeto inicial da interface através de prototipação. Assim, um sistema funcional está disponível nos primeiros estágios no processo de desenvolvimento e os equívocos entre os usuários de software e desenvolvedores ou falhas de projeto podem ser detectados e sanados rapidamente.

Como as regras de negócio do sistema definidas pelo modelo QOC não requerem nenhum estudo, podemos optar por uma abordagem horizontal, a fim de obtermos rapidamente a interface do usuário quase completa sem ter o foco nas funcionalidades por trás dos botões, demonstrando superficialmente toda a interface. Este tipo de protótipo permite testar a interface como um todo.

Optamos por um estilo de interação clássico do Windows com teclado e mouse e com interação de Manipulação Direta, onde o Usuário manipula diretamente a representações visíveis de objetos, que é atualizado imediatamente com alterações visíveis (feedback).
Nosso protótipo inicial de interface será composta por:

- Barra de Menus com os menus:
    - Arquivo
    - Editar
    - Exibir
    - Ferramentas
- Barra de Ferramentas com Botões:
    - Novo
    - Abrir
    - Salvar
    - Imprimir

    - Recortar
    - Copiar
    - Colar
    - Desfazer
    - Refazer
    - Pesquisa
    - Ajuda
- Barra de Ferramentas de Edição de Gráfico com Botões:
    - Inicio
    - Questão
    - Opção
    - Critério
    - Relação Positiva
    - Reação Negativa
    - Texto
    - Mover
    - Apagar
- Barra de Ferramentas de Edição de Texto composta por:
    - Caixa de Seleção de Fonte
    - Caixa de Seleção de Tamanho da Fonte
    - Negrito
    - Itálico
    - Sublinhado
    - Cor da Fonte
    - Realce
    - Link
    - Imagem
    - Alinhar Texto a Esquerda
    - Centraliza Texto
    - Alinhar Texto a Direita
    - Justificar Texto
    - Numeração
    - Marcadores
    - Recuar Texto
    - Avançar Texto
- Área de Desenho com componentes gráficos representando os Artefatos do Modelo QOC
- Barra de Status composta pelos Rótulos:
    - Número de Questões
    - Número de Opções
    - Número de Critérios
    - Número de Relações
    - Gráfico sob Cursor Do Mouse
    - Posição X do Cursor Do Mouse
    - Posição Y do Cursor Do Mouse

A Figura 3.2 mostra o Protótipo da Interface da Aplicação:

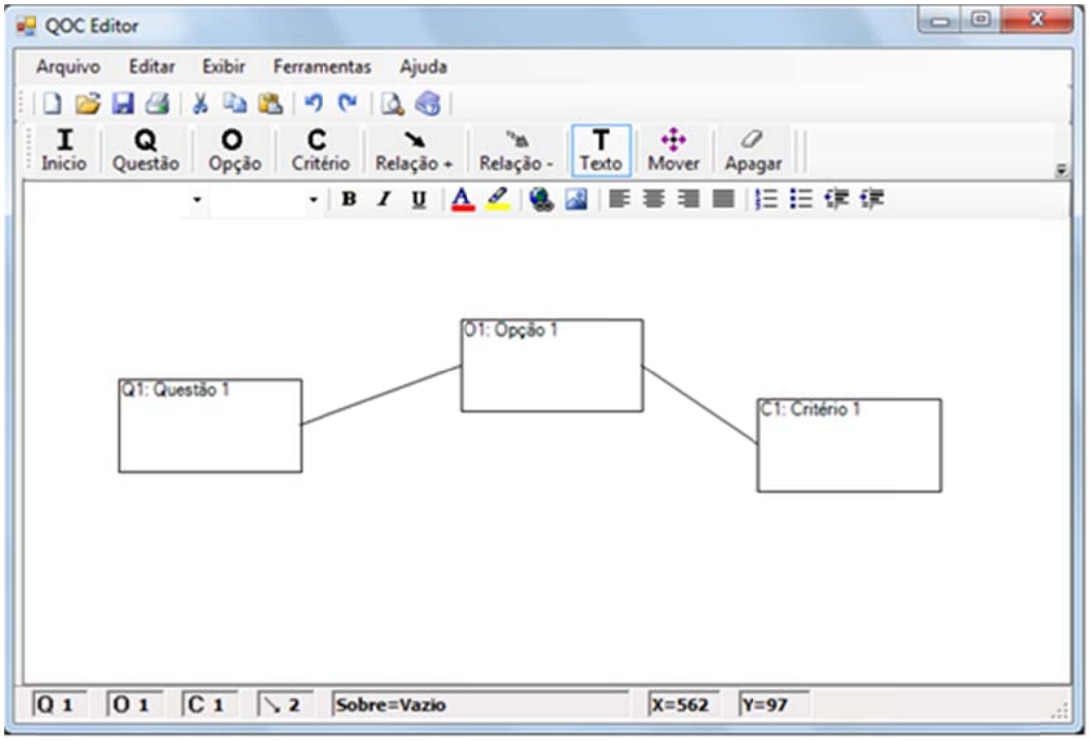

Figura 3.2 - Protótipo da Interface

3.7 Projeto do Diagrama de Classes

Um diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos. É uma modelagem muito útil para o desenvolvimento de sistemas, pois define todas as classes que o sistema necessita possuir e é a base para a construção dos diagramas de comunicação, sequência e estados [BOOCH, 1999].

Elaboramos um modelo básico de Diagrama de Classes para um primeiro protótipo funcional. Na linguagem C#, as classes modelam artefatos do domínio do problema. As classes contém métodos e atributos com diretivas de acesso: public, private e protected. Os métodos que implementam as operações. Os Atributos modelam aspectos dos objetos modelados. Os relacionamentos entre classes são mapeados diretamente do projeto para implementação. Métodos das classes de projeto são diretamente mapeados para implementação.

Nesse Diagrama de Classes, estão modeladas as classes que representam elementos do domínio do problema e que estão associadas aos requisitos básicos, e algumas classes de projeto, que representam elementos do espaço solução e que vão refletir na implementação.

Como definido na seção 3.2.1, usaremos um Modelo de 3 Camadas, com Camada de Apresentação, Camada de Negócio e Camada de Dados distintas. As principais associações estão relacionadas ao uso dos Patterns, como detalharemos na seção 3.9.

3.7.1 Classes da Camada de Apresentação

As classes que fazem parte da camada de apresentação são as seguintes:

- TProgram: É o principal ponto de entrada para a aplicação;
- FormMain: É o formulário principal do aplicativo que contém os menus e barra de ferramentas, botões, controles e a área de desenho;
- FormConfig: É o formulário que apresenta a interface de configuração da Área de Trabalho, Elementos Gráficos e Impressão;
- FormRede: É o formulário que apresenta a interface de configuração de rede;
- FormAjuda: É o formulário que apresenta informações sobre a autoria do projeto, além de algumas instruções de uso;
- Resources: Contém dados binários necessários para o aplicativo: ícones, imagens, arquivos XML, arquivos de texto, arquivos de áudio ou vídeo, tabelas.
- DrawArea: Componente que implementa a Área de Desenho;
- TCursor: Classe que cria os cursores customizados da aplicação;
- FormPropriedade: É o formulário que apresenta a interface de edição das propriedade dos Nodos e Relações.
- FormDialog: É o formulário que apresenta a interface de edição das propriedade dos Nodos e Relações.

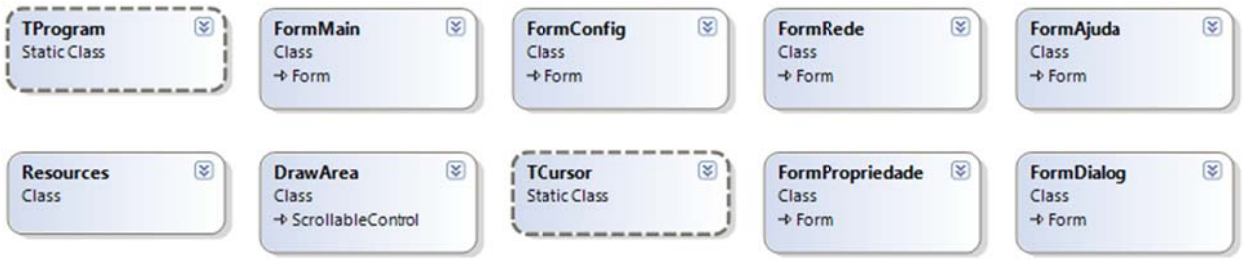

Figura 3.3 - Classes da Camada de Apresentação

3.7.2 Classes da Camada de Negócio

As classes que fazem parte da camada de negócio são as seguintes:

- TControle: Classe que implementa os métodos da Camada de Negócio;
- TArquivo: Classe que implementa os métodos de manipulação de arquivo;

- TExportarHtml: Classe que implementa os métodos de exportação de um diagrama para Html;
- TImpressão: Classe que implementa os métodos de configuração da impressões e de saída para a impressora.
- TSQLCom: Classe que implementa os métodos de comunicação com o banco de dados;
- eFerramenta: Classe que representa as ferramentas de edição de gráficos;
- eCursor: Classe que representa as posições do cursor;
- TParser: Classe que implementa os métodos de conversão dos tipos básicos;

Figura 3.4 - Classes da Camada de Negócio

3.7.3 Classes da Camada de Dados

As classes que fazem parte da camada de dados são as seguintes:

- TComposite: Classe que armazena os dados que representam um diagrama através de uma composição de objetos em estrutura de árvore para representar hierarquias parte-todo;
- TConfiguração: Classe que armazena os dados que representam uma configuração do aplicativo;
- TServerConfig: Classe que armazena os dados que representam uma configuração de Servidor de Banco de Dados;
- TProjetoInterface: Classe que armazena os dados de descrição que representam um projeto;
- TGrafico: Classe que armazena os dados que representam um gráfico genérico;
- TNodo: Classe que estende TGrafico para armazenar os dados que representam um Nodo;
- TReta: Classe que estende TGrafico para armazenar os dados que representam uma Reta;
- TGrafico: Classe que estende TNodo para armazenar os dados que representam um Gráfico;
- TQuestao: Classe que estende TNodo para armazenar os dados que representam um Questão;
- TInicio: Classe que estende TNodo para armazenar os dados que representam o Início;

- TOpcao: Classe que estende TNodo para armazenar os dados que representam Opção.
- TReta: Classe que estende TNodo para armazenar os dados que representam o Início;
- TRelacao: Classe que estende TNodo para armazena os dados que representam Opção.
- XFont: Classe que armazena os dados que representam o Início;
- XColor: Classe que armazena os dados que representam Opção.

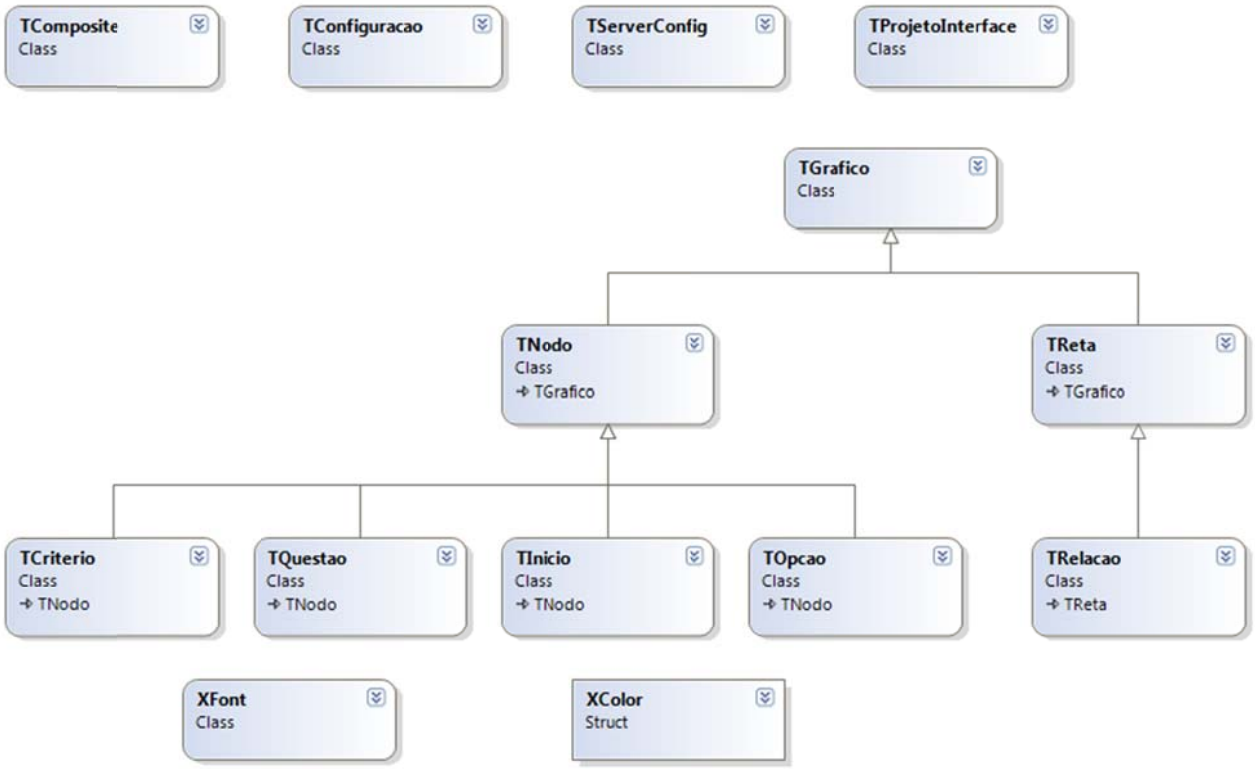

Figura 3.5 - Classes da Camada de Dados

3.8 Projeto do QOC – Editor com base em Design Patterns

Como veremos detalhadamente a seguir, alguns Designs Patterns são ótimas soluções para problemas recorrentes no desenvolvimento de aplicações gráficas, como editores de desenho ou sistemas de captura de esquema.

São soluções bem conhecidas que facilitam o reuso de artefatos e acarretam um vocabulário comum de desenho, facilitando comunicação, documentação e aprendizado dos sistemas de software.

3.8.1 Padrão Command

O Padrão Command é um padrão de projeto de software do GOF. Ele serve para enfileirar e executar solicitações em tempos, assim permitindo a parametrização de clientes com diferentes requisições, enfileirar ou requisições de log, e suportando operações de desfazer e refazer.

Os componentes do padrão: Client, Receiver, ConcreteCommand, Command, Invoker formam utilizados conforme a definição de [GAMMA, 1994]

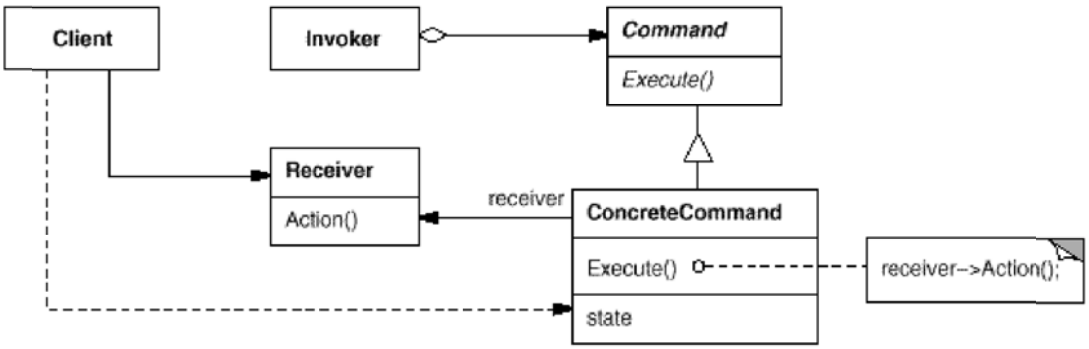

Figura 3.6 - Estrutura Abstrata do Padrão Command[GAMMA, 1994]

3.8.1.1 Utilização do Padrão Command no QOC Editor

O padrão Command foi utilizado na aplicação QOC para implementar as funções desfazer e refazer, através dos métodos: Undo, Redo, SalvarEstado.

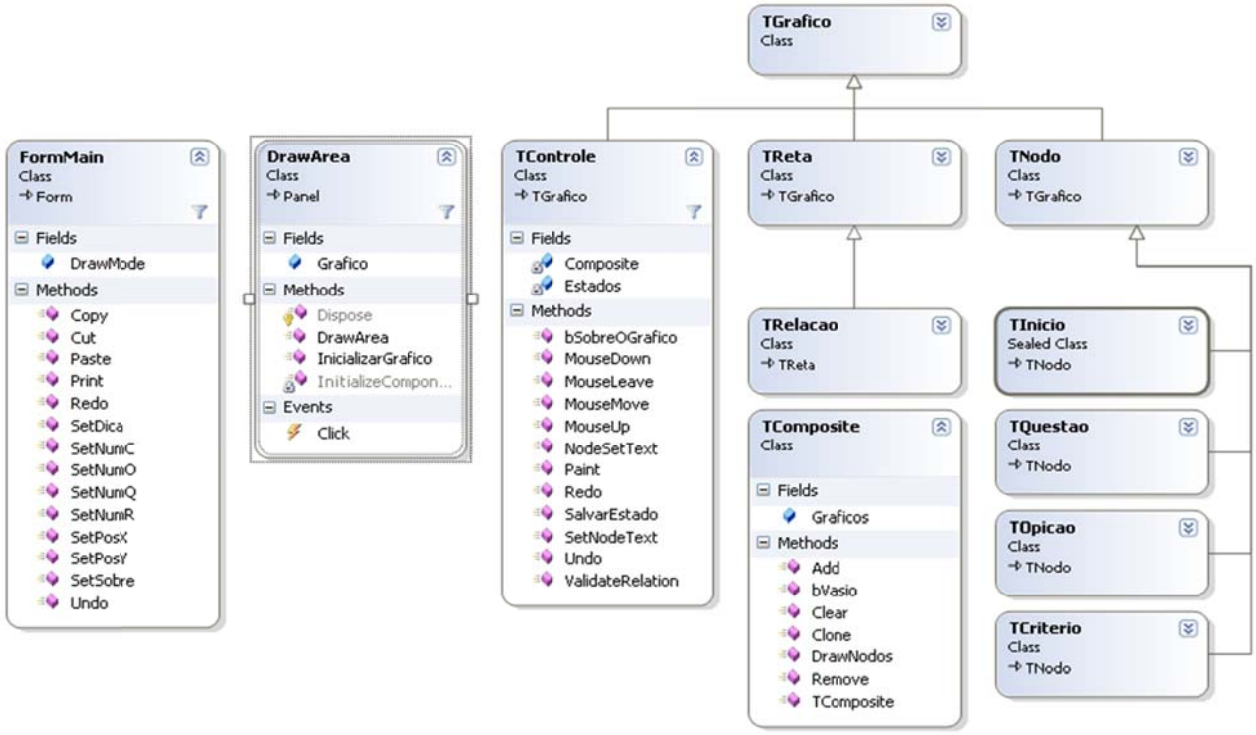

Figura 3.7 – Utilização do Padrão Command no QOC Editor

A Tabela abaixo descreve como o padrão Command foi utilizado na aplicação QOC:

Tabela 3.3 – Utilização do Padrão Command no QOC Editor

COMPONENTE	CLASSE	FUNÇÃO
Client	FormMain	• cria um objeto ConcreteCommand e define seus receptores;
Receiver	TControle	• sabe como realizar as operações associadas executando os pedidos. Qualquer classe pode servir como um Receiver;
ConcreteCommand	TComposite	• define uma ligação entre um objeto Receiver e uma ação; • implementa Execute invocando a correspondente operação no Receiver;
Command	TGrafico	• declara uma interface para executar uma operação;
Invoker	DrawArea	• solicita ao Command para executar sua requisição;

3.8.1.2 Funcionamento do Padrão Command no QOC Editor

1. O FormMain cria um objeto TComposite, especificando TControle como seu Receiver;
2. TControle armazena os objetos TComposite;
3. DrawArea invoca operações em TControle que direciona o pedido ao objeto TComposite corrente que executa o pedido.

3.8.2 Padrão Composite

O Padrão Composite é um padrão de projeto de software utilizado para representar um objeto que é constituído pela composição de objetos similares. Neste padrão, o objeto composto possui um conjunto de outros objetos que estão na mesma hierarquia de classes a que ele pertence. O padrão Composite é normalmente utilizado para representar listas recorrentes - ou recursivas - de elementos.

Além disso, esta forma de representar elementos compostos em uma hierarquia de classes permite que os elementos contidos em um objeto composto sejam tratados como se fossem um único objeto. Desta forma, todos os métodos

comuns às classes que representam objetos atômicos da hierarquia poderão ser aplicáveis também ao conjunto de objetos agrupados no objeto composto. [GAMMA, 1994]. O padrão Composite foi utilizado na aplicação QOC para implementar as classes que representam os elementos gráficos do modelo QOC, de maneira a compor objetos em estrutura de árvore para representar hierarquias parte-todo, a fim de tratar os objetos individuais e composições de objetos uniformemente.

Os componentes do padrão: Client, Component, Leaf e Composite foram utilizados conforme a definição de [GAMMA, 1994].

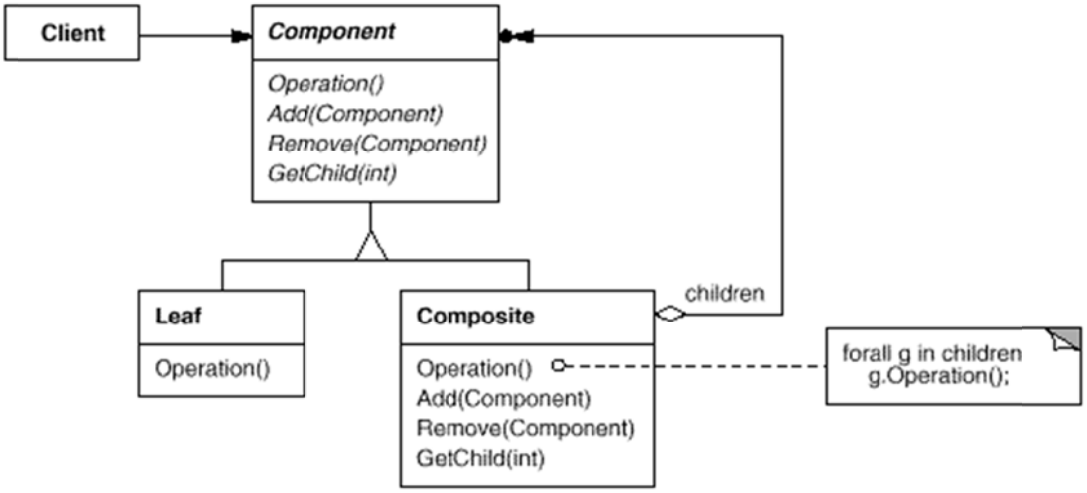

Figura 3.8 - Estrutura Abstrata do Padrão Composite [GAMMA, 1994]

3.8.2.1 Utilização do Padrão Composite no QOC Editor

Todos os métodos que criam e alteram as classes que representam os elementos gráficos do modelo QOC foram implementados através do Padrão Composite, como mostra o diagrama de classes abaixo.

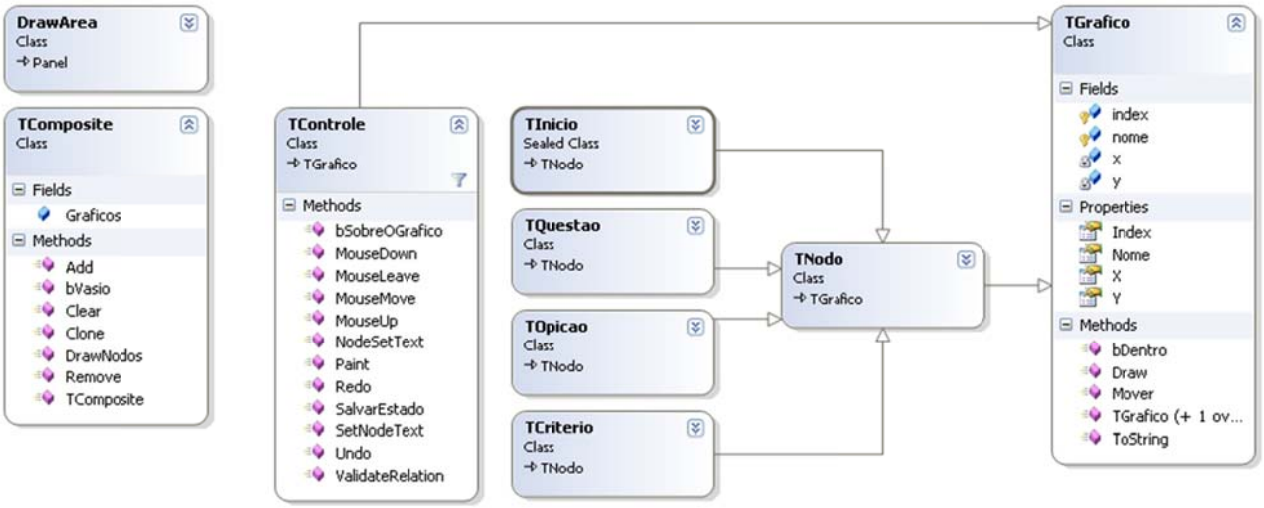

Figura 3.9 - Utilização do Padrão Composite no QOC Editor

A Tabela abaixo descreve como o padrão Composite foi utilizado na aplicação QOC:

Tabela 3.4 - Utilização do Padrão Composite no QOC Editor

COMPONENTE	CLASSE	FUNÇÃO
Client	DrawArea	• manipula objetos na composição através da interface do Component.
Component	TGrafico	• declara a interface para os objetos na composição; • implementa comportamento default para a interface comum a todas as classes, quando apropriado; • declara uma interface para acessar e gerenciar seus componentes filhos; • define uma interface para acessar o pai de um componente na estrutura recursiva e a implementa (opcional).
Leaf	TCriterio TInicio TOpicao TQuestao TRelacao	• representa o objeto folha na composição. Uma folha não tem filhos; • define comportamento para objetos primitivos na composição;
Composite	TComposite	• define comportamento para componentes que possuem filhos; • armazena componentes filhos; • implementa operações relacionadas com os filhos na interface do componente.

3.8.2.2 Funcionamento do Padrão Composite no QOC Editor

1. O DrawArea aciona um método de um objeto TGrafico através TControle;
2. TControle repasse a chama o TComposite corrente;
3. TControle invoca operações sobre ao objeto TGrafico especificado;

3.8.3 Padrão Singleton

O Padrão Singleton é um padrão de projeto de software do GOF. Ele garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto [GAMMA, 1994].

Os componentes do padrão Singleton foram utilizados conforme a definição de [GAMMA, 1994].

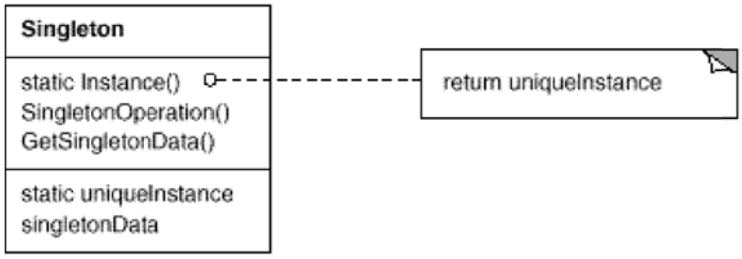

Figura 3.10 - Estrutura Abstrata do Padrão Singleton [GAMMA, 1994]

3.8.3.1 Utilização do Padrão Singleton no QOC Editor

O Padrão Singleton foi utilizado na aplicação QOC para implementar a classe que representa a Questão Inicial TInicio, de forma a assegurar que a classe tenha somente uma instância e fornecer um ponto global de acesso a ela. A figura abaixo apresenta a classe TInicio que implementa o padrão Singleton.

Figura 3.11 - Utilização do Padrão Singleton no QOC Editor

A Tabela abaixo descreve como o padrão Singleton foi utilizado na aplicação QOC:

Tabela 3.5 - Utilização do Padrão Singleton no QOC Editor

COMPONENTE	CLASSE	FUNÇÃO
Singleton	TInicio	• define uma operação Instance que permite aos clientes terem acesso a sua única instância. Instance é uma classe de operação; • pode ser responsável por criar sua própria instância única.

3.8.3.2 Funcionamento do Padrão Singleton no QOC Editor

Clientes acessam a instância Singleton exclusivamente através da operação de instanciamento da classe GetInstance. Assim, temos o controle de acesso a uma única instância, sem necessidade da utilização de variáveis globais para o controle da instância única.

3.8.4 Padrão State

O Padrão State é um padrão de projeto de software do GOF, que permite implementar objetos que mudam de comportamento dependendo do seu estado [GAMMA, 1994]. Os componentes do padrão: Context, State, ConcreteState formam utilizados conforme a definição de [GAMMA, 1994].

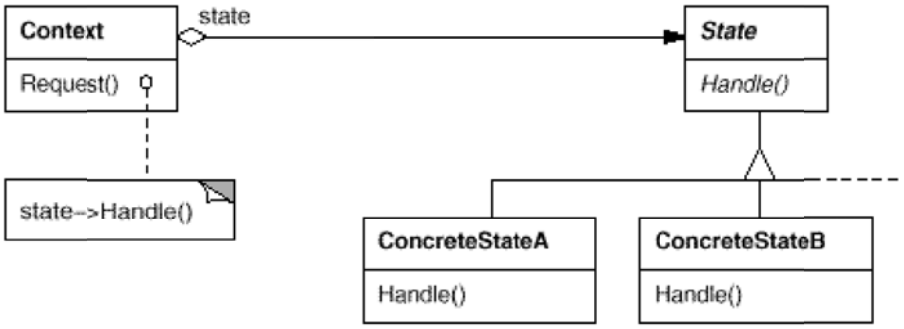

Figura 3.12 - Estrutura Abstrata do Padrão State [GAMMA, 1994]

3.8.4.1 Utilização do Padrão State no QOC Editor

O padrão State foi utilizado na aplicação QOC para implementar as funções MouseDown, MouseLeave, MouseMove e MouseUp:

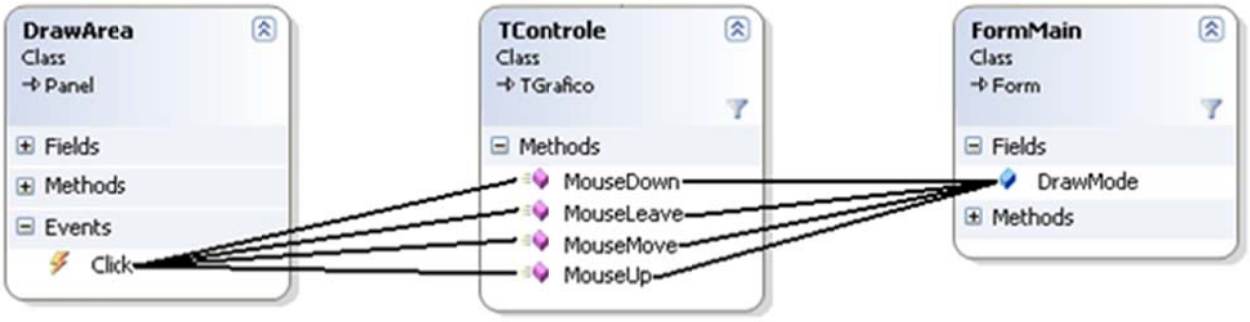

Figura 3.13 - Utilização do Padrão State no QOC Editor

A Tabela abaixo descreve como o padrão State foi utilizado na aplicação QOC:

Tabela 3.6 - Utilização do Padrão State no QOC Editor

COMPONENTE	CLASSE	FUNÇÃO
Context	TControle	• define a interface de interesse dos clientes. • Mantém uma instância de uma subclasse ConcreteState que define o estado corrente.
State	TComposite	• define uma interface para encapsular o comportamento associado com um particular estado do Context.
ConcreteState	TGrafico	• cada subclasse implementa um comportamento associado com um estado de Context.

3.8.4.2 Funcionamento do Padrão State no QOC Editor

1. DrawArea aciona um método de um objeto TGrafico através TControle;
2. TControle delega pedidos estado-específico ao corrente objeto TComposite;
3. Controle invoca operações sobre o objeto TGrafico conforme, seu estado;

3.8.5 Padrão Template Method

O Padrão Template Method é um padrão de projeto de software do GOF, que auxilia na definição de um algoritmo com partes do mesmo, definidos por Métodos abstratos. As subclasses devem se responsabilizar por estas partes abstratas deste algoritmo, que serão implementadas, possivelmente de várias formas, ou seja, cada subclasse irá implementar a sua necessidade e oferecer um comportamento concreto construindo todo o algoritmo. Os componentes do padrão: AbstractClass, ConcreteState formam utilizados conforme a definição de [GAMMA, 1994].

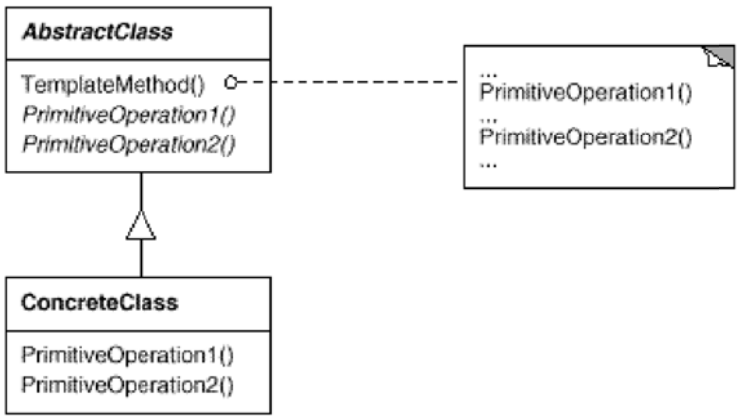

Figura 3.14 – Estrutura Abstrata do Padrão Template Method [GAMMA, 1994]

3.8.5.1 Utilização do Padrão Template Method no QOC Editor

O Padrão Template Method foi usado para implementar um algoritmo básico para as funções Mover e Draw da Classe TGrafico, permitindo que subclasses complementem os algoritmos, se necessário, sem mudar sua estrutura. Isso permite implementar uma única vez as partes invariantes do algoritmo e deixar comportamentos distintos serem modificados pelas subclasses.

Abaixo, o diagrama de classes mostra como o padrão TemplateMethod foi usado para implementar as funções Mover e Draw da classe TGrafico. A Tabela abaixo descreve como o padrão Template Method foi utilizado.

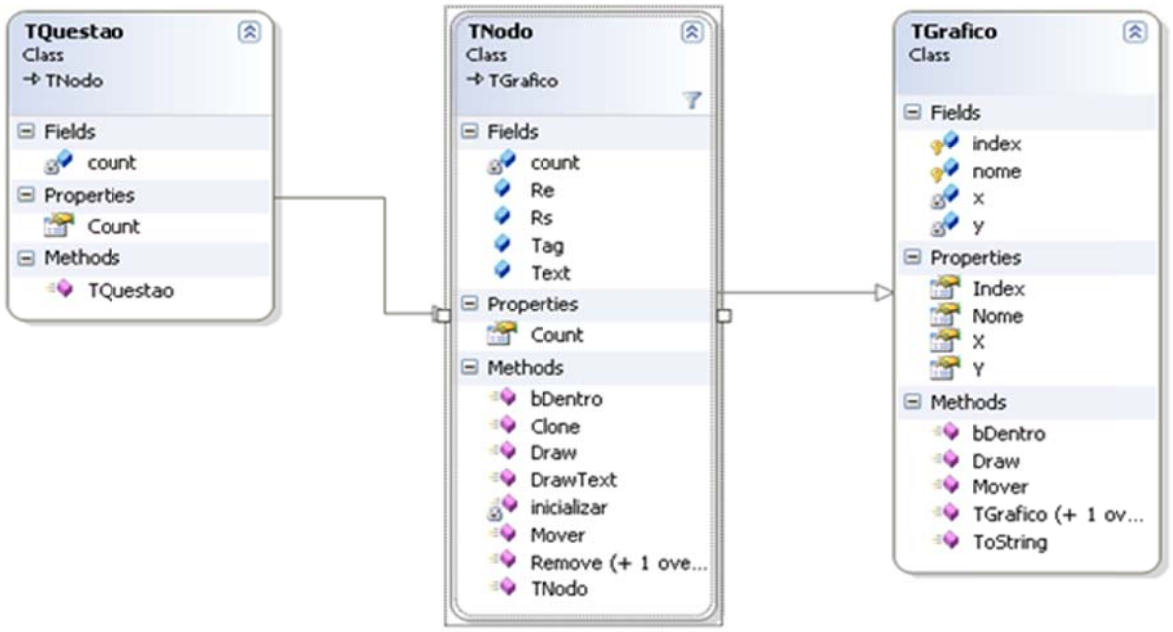

Figura 3.15 - Utilização do Padrão Template Method no QOC Editor

Tabela 3.7 - Utilização do Padrão Template Method no QOC Editor

COMPONENTE	CLASSE	FUNÇÃO
AbstractClass	TGrafico	• define operações primitivas abstratas que subclasses concretas definem para implementar passos de um algoritmo. • implementa um Template Method definindo o esqueleto de um algoritmo. O template method chama operações primitiva bem como operações definidas na AbstractClass ou aquelas de outros objetos.
ConcreteClass	TNodo TRelacao	• implementa operações primitiva que se encarregam de passos específicos de subclasses de um algoritmo.

3.8.5.2 Funcionamento do Padrão Template Method no QOC Editor

1. DrawArea aciona o método move de um objeto TGrafico através de TControle;
2. TControle repasse a chama o TComposite corrente;
3. TControle invoca as operações sobre o objeto TNodo especificado;
4. TNodo executa os passos invariantes do algoritmo e chama o padrão comum do algoritmo na Classe TGrafico.

3.9 Modelo inicial de Diagrama de Classes

Com as principais classes e seus relacionamentos definidos pelo designe pattern, podemos elaborar um bom diagrama de classes. Nesse primeiro Diagrama de Classes, modelaremos os requisitos básicos e deixaremos para a implementação o detalhamento do modelo. Métodos das classes de projeto são diretamente mapeados para implementação.

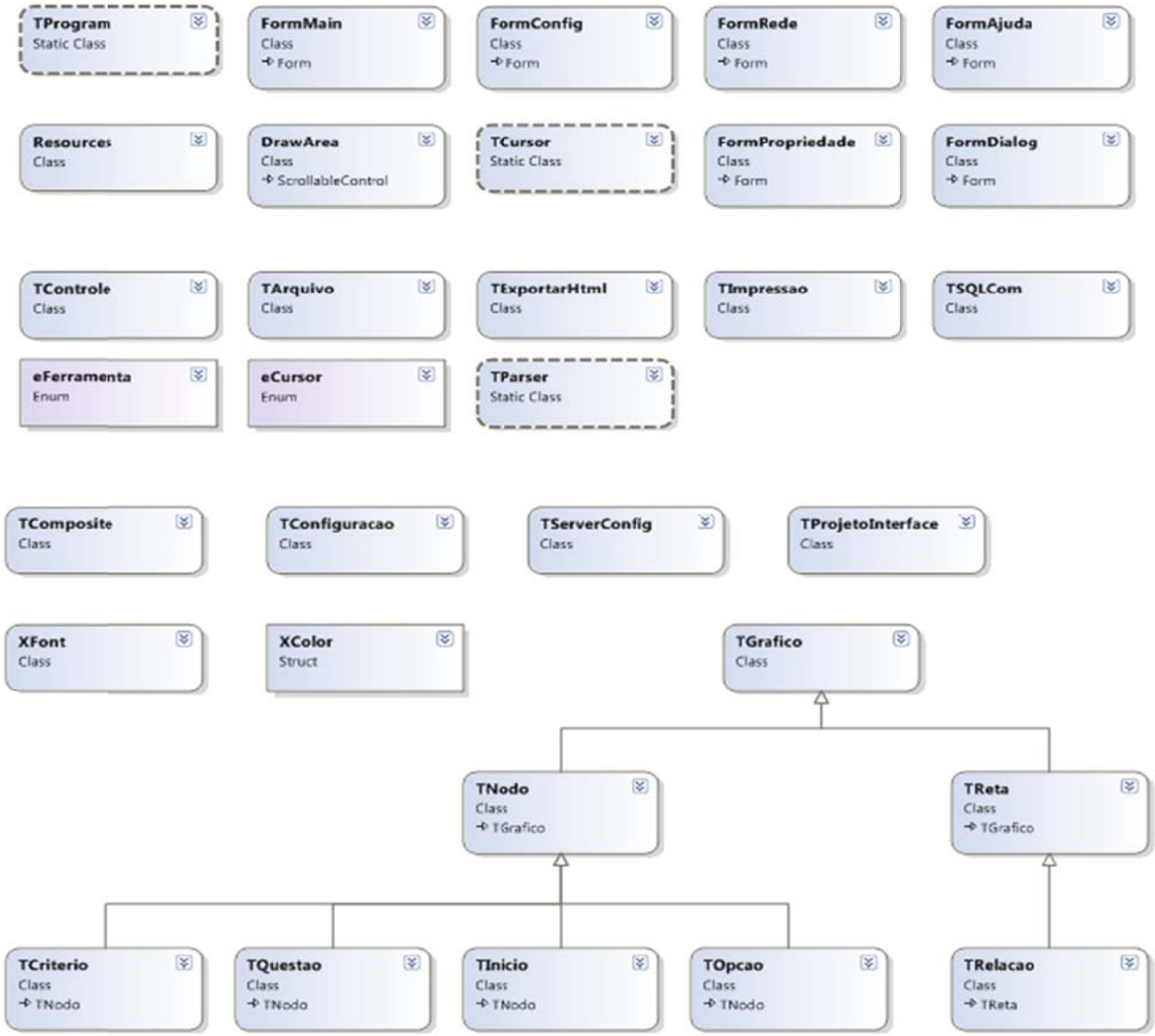

Figura 3.16 - Modelo inicial de Diagrama de Classes

Esse modelo inicial de Diagrama de Classes será usado para um primeiro protótipo funcional como veremos a seguir.

4.0 IMPLEMENTAÇÃO DO QOC – EDITOR

O Aplicativo será implementado pelo Microsoft Visual Studio 2008 usando a linguagem C# (CSharp) Framework .NET 3.5, de quem herda suas principais características:

- Orientação a Objetos, em C# é orientado a componente (classes com funções básicas), permitindo decompor o jogo de acordo com elementos abstratos do domínio do problema;
- Usa a plataforma Framework .NET como bibliotecas de classes e funções.
- Interface implementada pela IDE Gráfica do Visual Studio usando os componentes do Windows Forms;
- Alta Integração com o Windows
- Suporte Amarração Dinâmica: durante o processo de execução, tipos variantes, associação de valor e posição, invocação de métodos redefinidos.
- Classe de Coleção (List) foi muito usada.
- Alocação e Deslocação (garbage collector) automática de memória.
- Computação Paralela através do Parallel Extensions incorporado ao .NET Framework.

4.1 Implementação da Interface

A interface foi desenhada no editor de Form do Microsoft Visual Studio 2008 usando a linguagem C#. Utilizamos o mínimo de componentes para economizar memória e processamento.

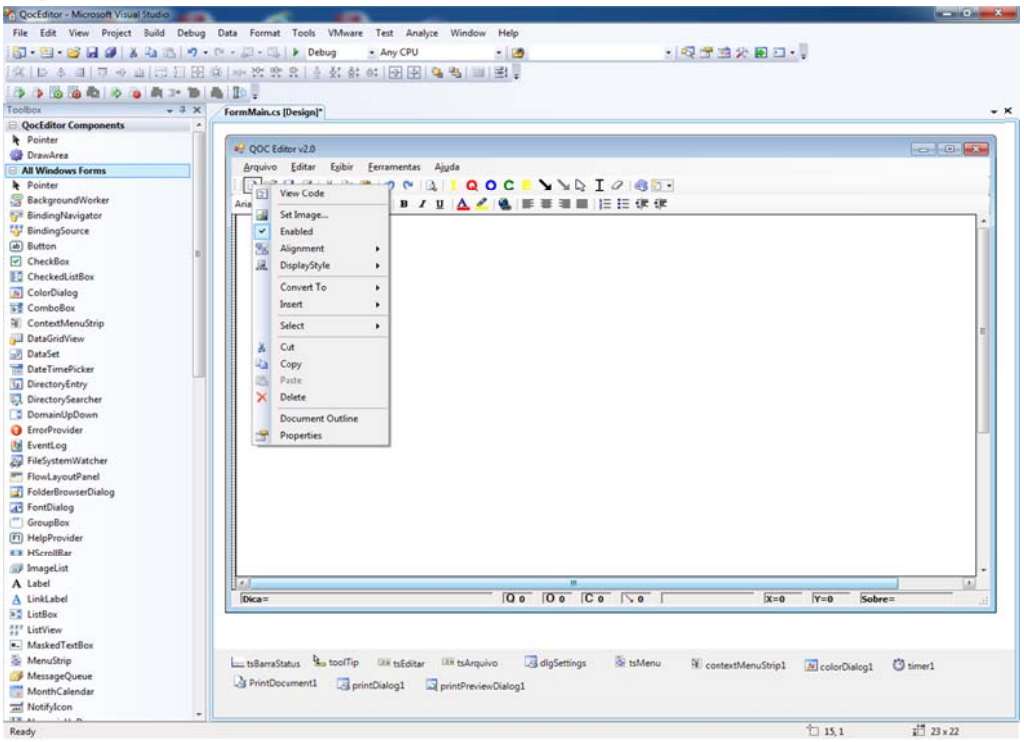

Figura 4.1 - Implementação da Interface

4.2 Desenvolvimento do QOC Editor por Prototipagem

O QOC Editor foi desenvolvido através de um processo de Prototipagem. Prototipação (ou prototipagem) de Software é um processo interativo de geração de modelos de software que faz parte da análise do ciclo de vida do desenvolvimento de sistemas. É a atividade de desenvolvimento de uma versão inicial do sistema, baseada no atendimento dos requisitos ainda pouco definidos, permitindo a descoberta de falhas difíceis de serem encontradas na comunicação verbal. Um processo que propõe a criação de um protótipo de software objetiva apoiar a fase levantamento de requisitos a fim de prevenir as possíveis falhas no sistema. [PRESSMAN, 2002]

Um protótipo simula a aparência e funcionalidade do software permitindo que os clientes, analistas, desenvolvedores e gerentes percebam os requisitos do sistema podendo interagir, avaliar, alterar e aprovar as características mais marcantes na interface e funções. Os protótipos podem ser evolutivos ou descartáveis.

Na prototipagem evolutiva, o sistema surge de evoluções refinadas dos protótipos, enquanto um protótipo descartável é usado para descobrir problemas nos requisitos e depois é abandonado. [PRESSMAN, 2002] A Figura 4.12 mostra o funcionamento genérico do modelo.

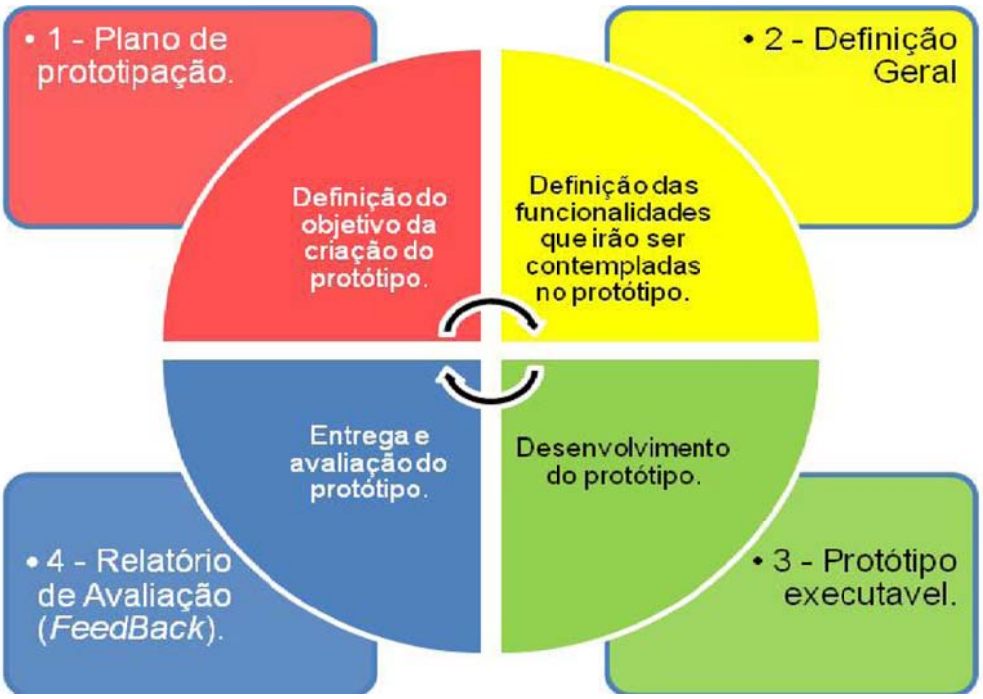

Figura 4.2 - O processo de prototipação em quatro fases [PRESSMAN, 2002]

Dentre algumas vantagens da Prototipação está a redução de custos no desenvolvimento; participação do usuário no processo de desenvolvimento; facilidade

de operação do sistema, uma vez que os usuários sabem o que esperar através do protótipo; resultados na satisfação mais elevada do usuário; diminuição de equívocos entre usuários e desenvolvedores; esclarecimento de alguns requisitos confusos. Algumas desvantagens no uso de protótipos são: a condução a uma análise insuficiente do software; os usuários esperam um desempenho do software final igual ao do protótipo; os clientes podem tornar-se unidos demais a seus protótipos [PRESSMAN, 2002].

O modelo conceitual e as funcionalidade a serem implementadas a cada ciclo estando definidas, partimos para a implementação. Através do primeiro protótipo funcional foi possível levantar os requisitos que não ficaram claros no inicio do processo.

A cada ciclo de desenvolvimento foi implementada uma funcionalidade, onde a Aplicação foi sendo moldada em torno do seu uso e o resultado final será apresentado a seguir.

5.0 USANDO O QOC EDITOR

Neste capítulo, apresentaremos cinco cenários básicos de uso, descrevendo o uso passo a passo através de ilustração de telas. Quando necessário, explicaremos o comportamento interno para que um cenário seja realizado.

5.0.1 Elaboração do Plano de Testes

Como definido pela norma internacional IEEE 829, é o documento responsável por apresentar o planejamento para execução dos testes do software em desenvolvimento, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste. No plano de Teste verifica-se a necessidade das seguintes informações:

- Identificador: um nome único que permita a identificação do plano de teste.
- Histórico de versões: lista as versões do documento, identificando o que foi alterado e o responsável por cada versão.
- Descrição: descreve o objetivo e o escopo do plano de teste; podendo ser identificados também os recursos e restrições orçamentárias.
- Itens de teste:
    - Descreve todos os elementos que serão testados.
    - Deve ser identificada a prioridade de cada elemento a ser testado.
- Funcionalidades que serão testadas:
    - Nesta seção, devem ser descritas as funcionalidades, do ponto de vista do usuário, que serão testadas.
    - O nível de risco (ou prioridade) de cada funcionalidade deve ser explicitado, de modo que seja compreensível para o usuário.
    - A diferença desta seção para a seção “Itens de teste” é o ponto de vista da explicação. Nesta, o foco é o usuário, na outra o foco é na equipe técnica.
- Funcionalidades que não serão testadas:
    - Nesta seção devem ser listadas as funcionalidades que NÃO serão testadas, do ponto de vista do usuário.
    - Também deve estar claro o porquê de tais funcionalidades não serem testadas
- Como foi o teste:
    - Descrição das estratégias que serão usadas para cada teste, identificando atividades e ferramentas usadas no teste.
    - Descrição dos critérios de sucesso ou falha de cada teste.
    - Identificar a preparação/configuração do ambiente, descrevendo quais atividades devem ser executadas antes ou após a realização do teste.
- Gerenciamento dos testes:
    - Detalhar o cronograma, os riscos e as contingências para os testes.
- Aprovação do documento: indica a aprovação do documento pelos envolvidos (assinaturas).

5.1 Cenário 1: Teste Básico

Iniciado o aplicativo, ele exibe sua interface:

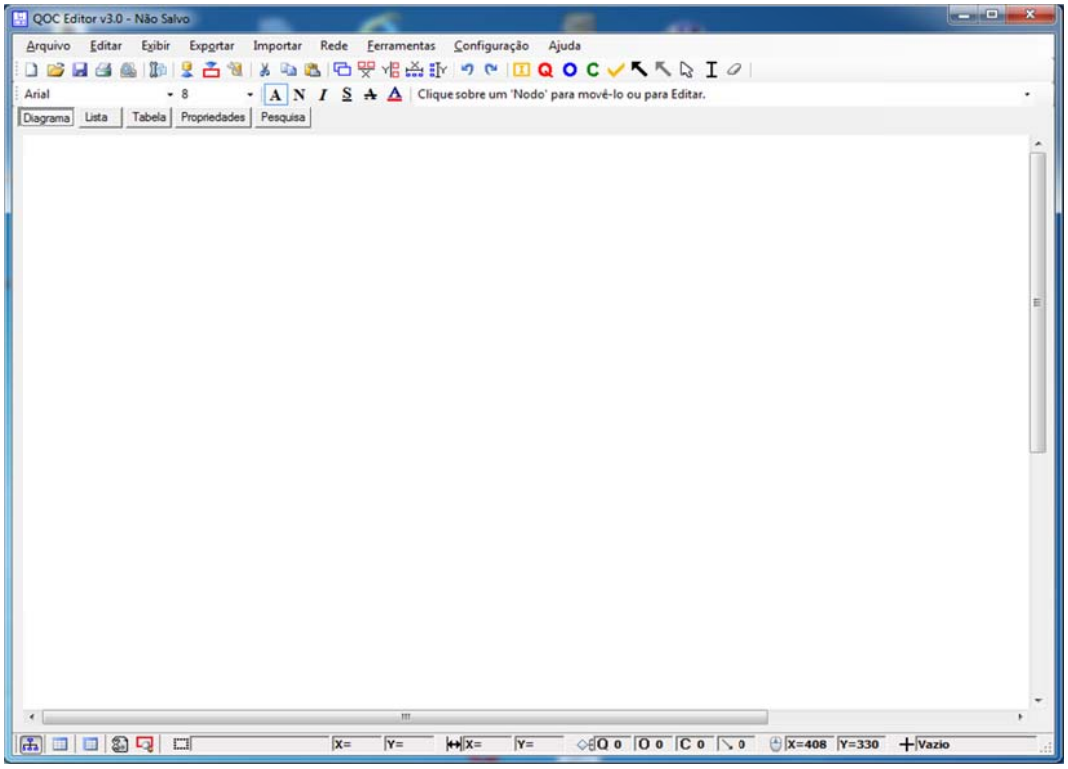

Figura 5.1

Para criar um Nodo Questão, selecionamos a Ferramenta Q:

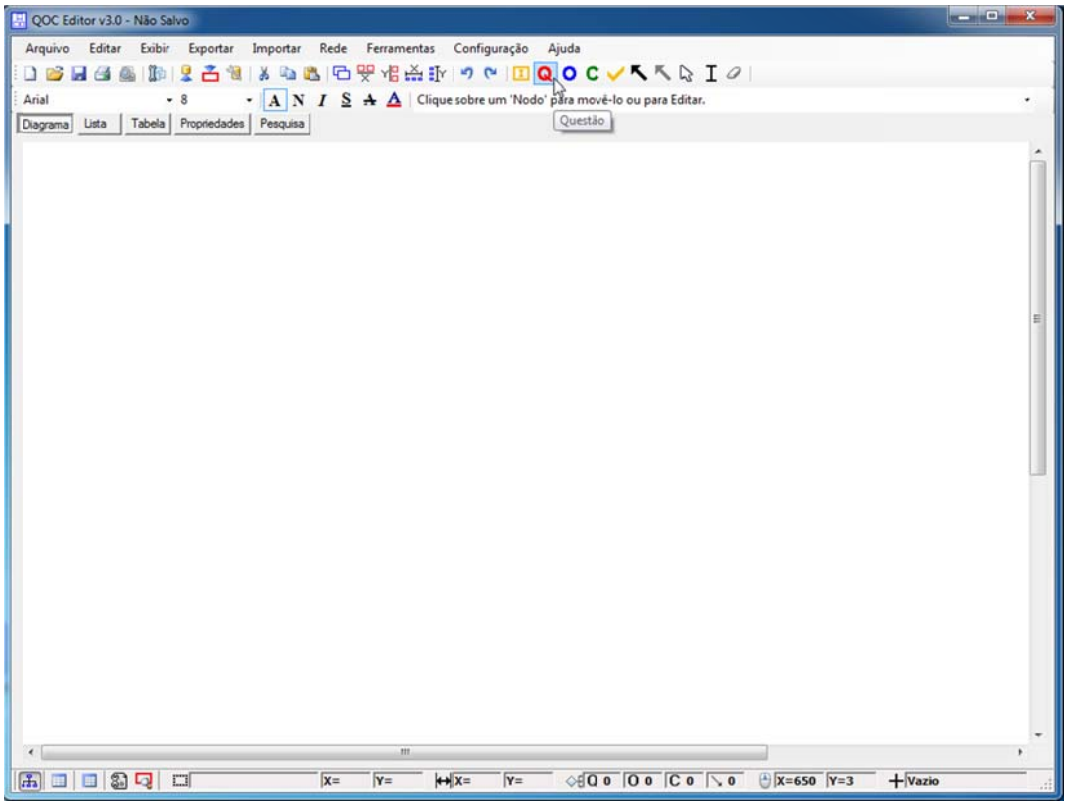

Figura 5.2

Com Ferramenta Q basta clicar na posição deseja para criar a questão:

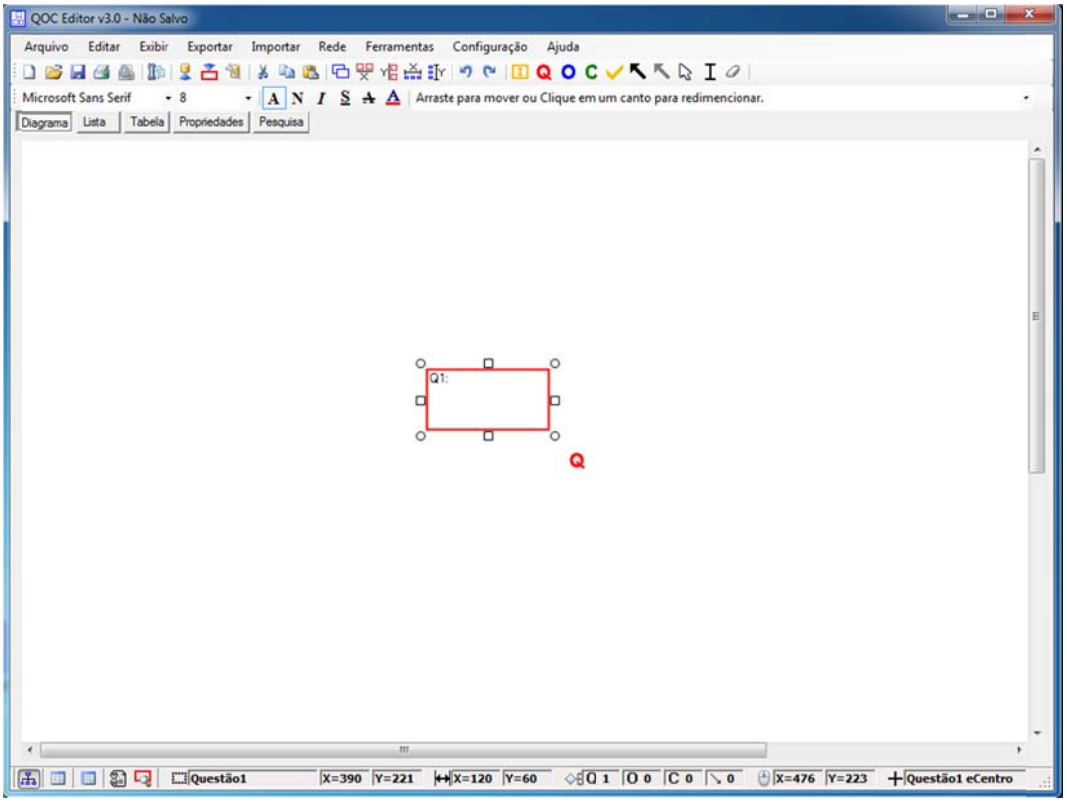

Figura 5.3

Para criar um Nodo Opção, selecionamos a Ferramenta O:

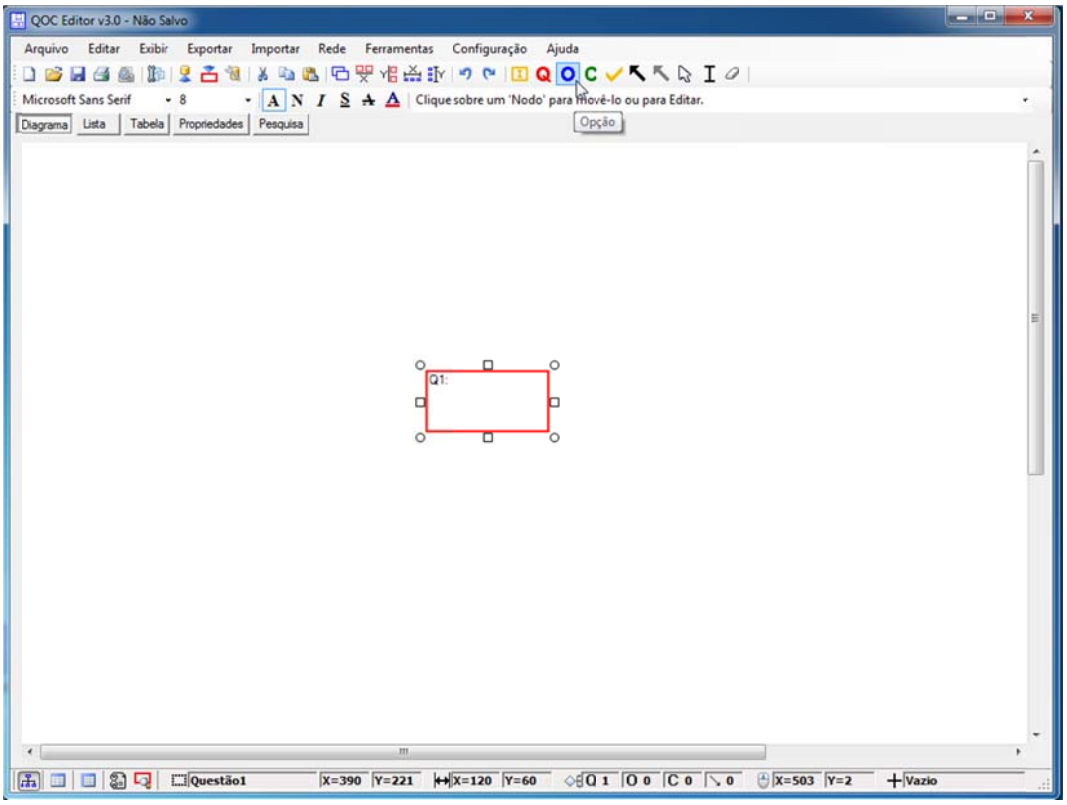

Figura 5.4

Com Ferramenta O basta clicar na posição desejada para criar o Nodo Opção:

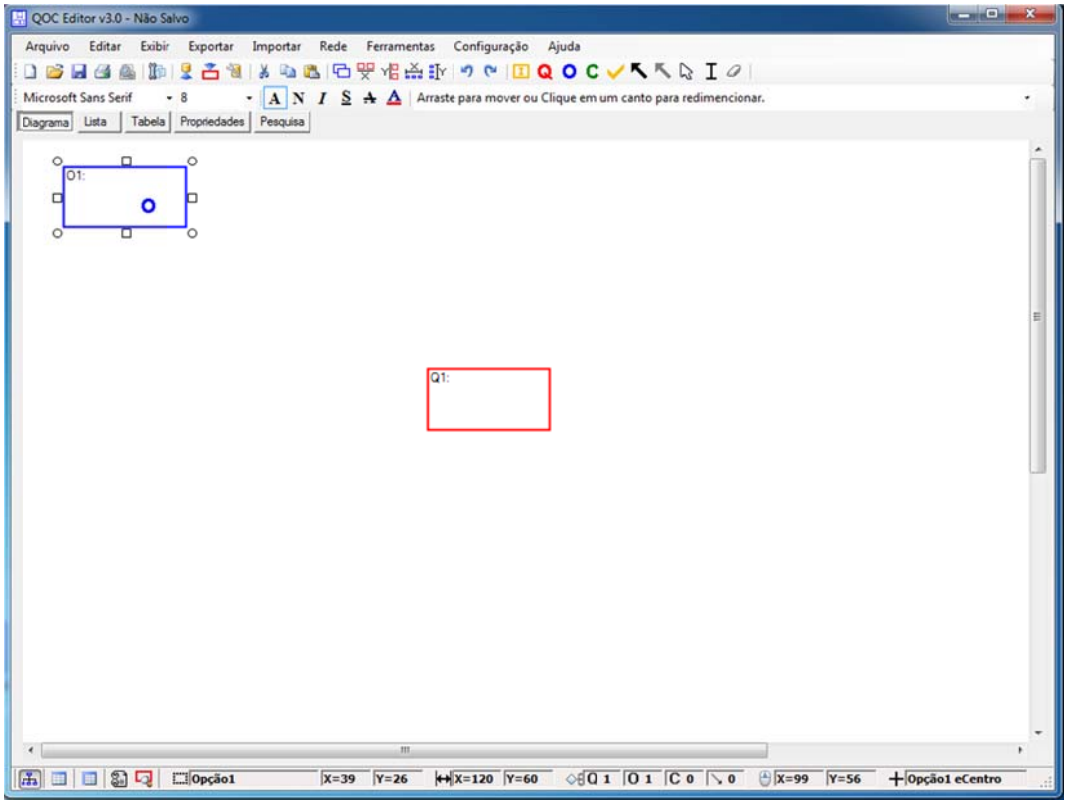

Figura 5.5

Agora iremos relacionar o Nodo Questão 1 com o Nodo Opção 1. Para isso, utilizamos a Ferramenta Relação Positiva:

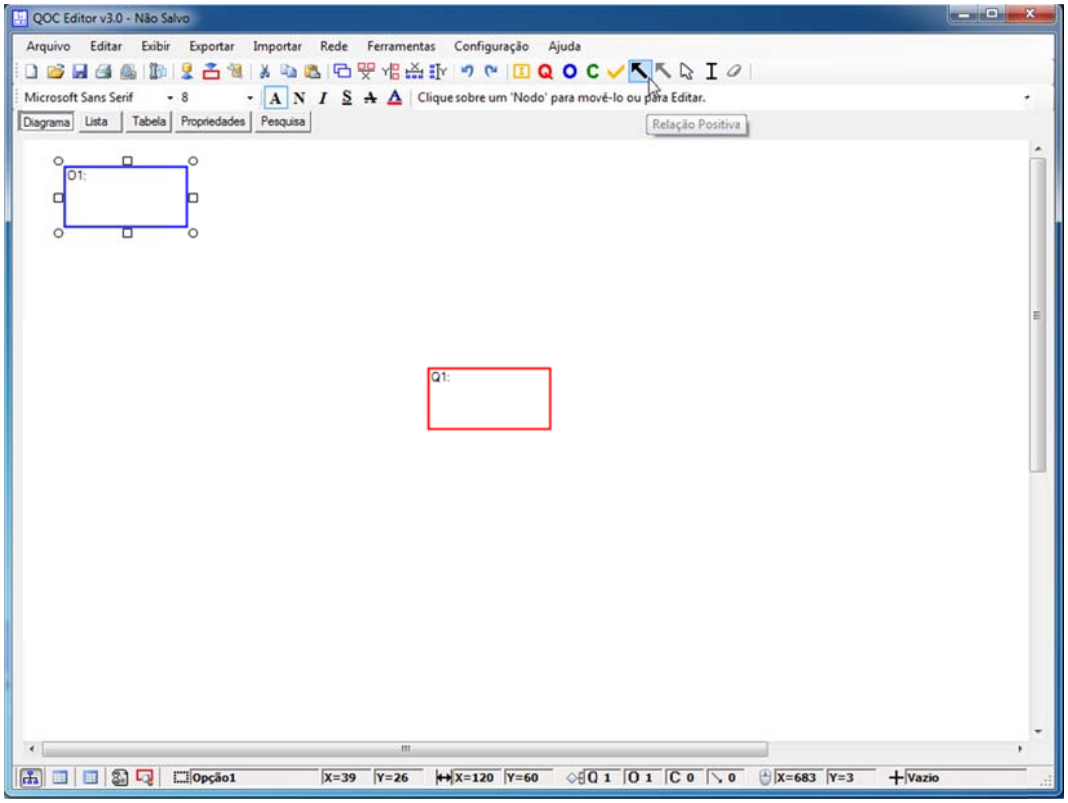

Figura 5.6

Com a Ferramenta Relação Positiva, basta clicar no Nodo Questão 1 e arrastar até o Nodo Opção 1 para criar a relação:

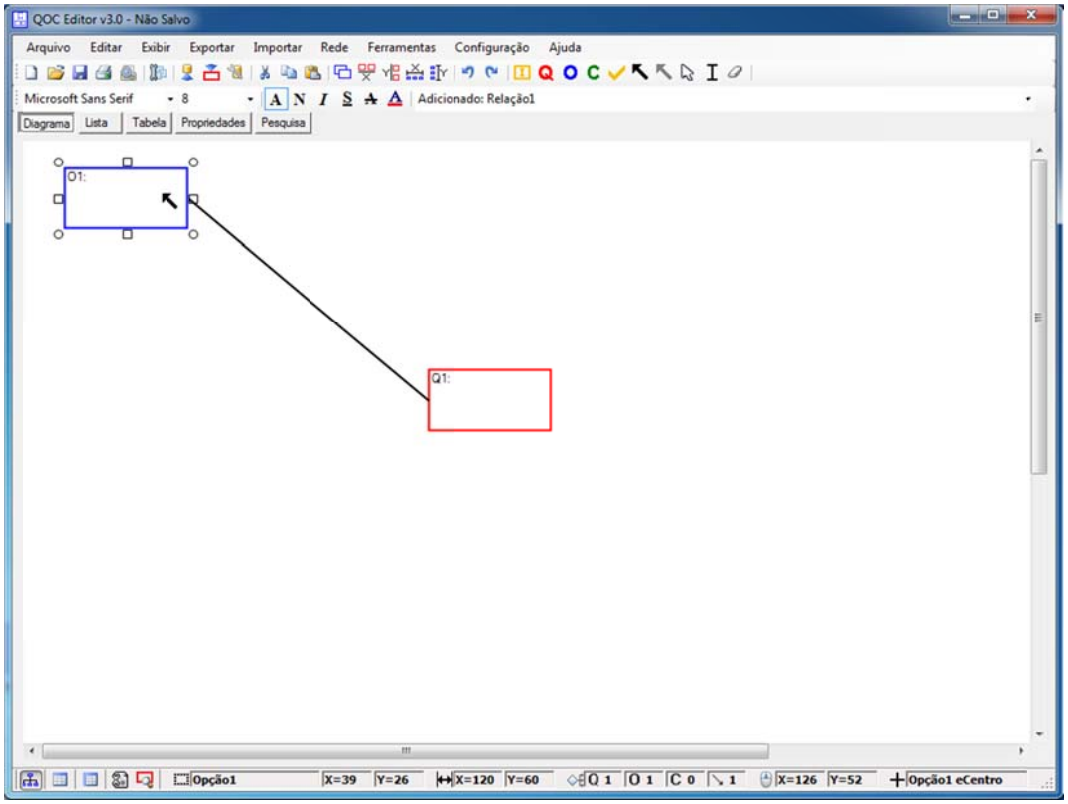

Figura 5.7

Agora iremos escrever o texto no Nodo Questão1. Para isso, clicamos com o botão direto sobre o Nodo Questão e escolhemos o item Propriedades:

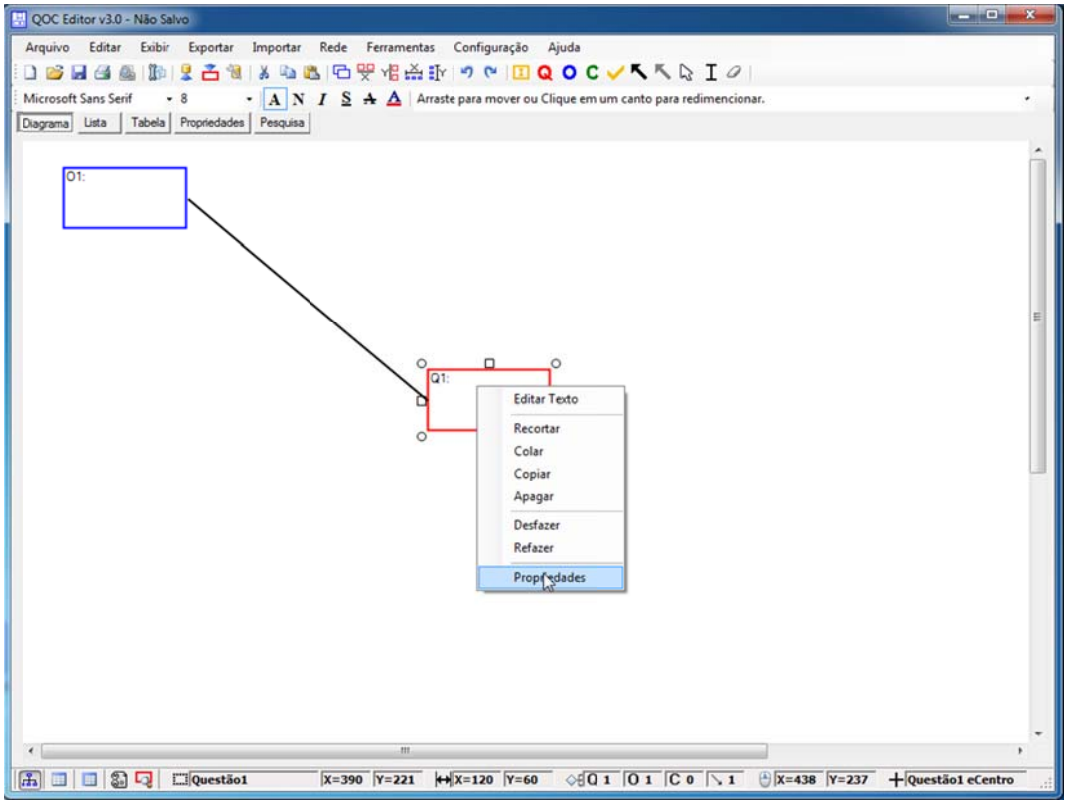

Figura 5.8

Então, o aplicativo exibe o menu de Propriedades do Nodo. No campo Texto, digitamos o Texto desejado e clicamos em salvar:

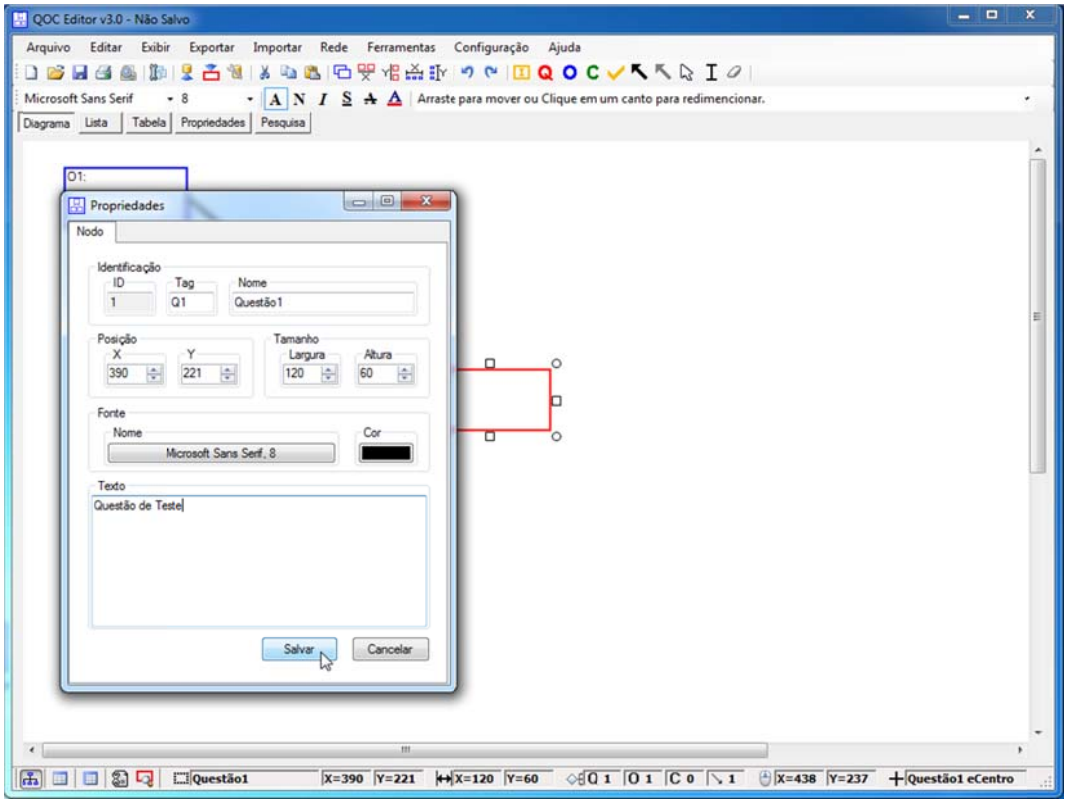

Figura 5.9

Repetimos o processo com Nodo acrescentar

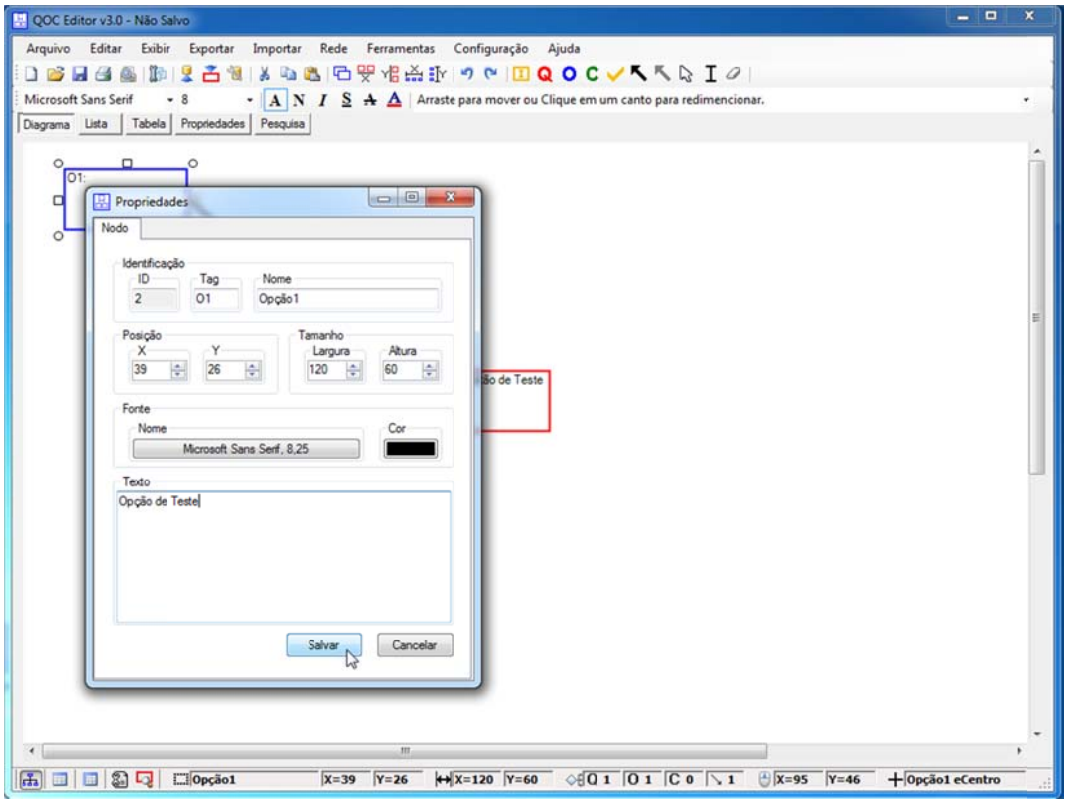

Figura 5.10

Assim, o Menu é fechado e o texto digitado é exibido dentro do Nodo:

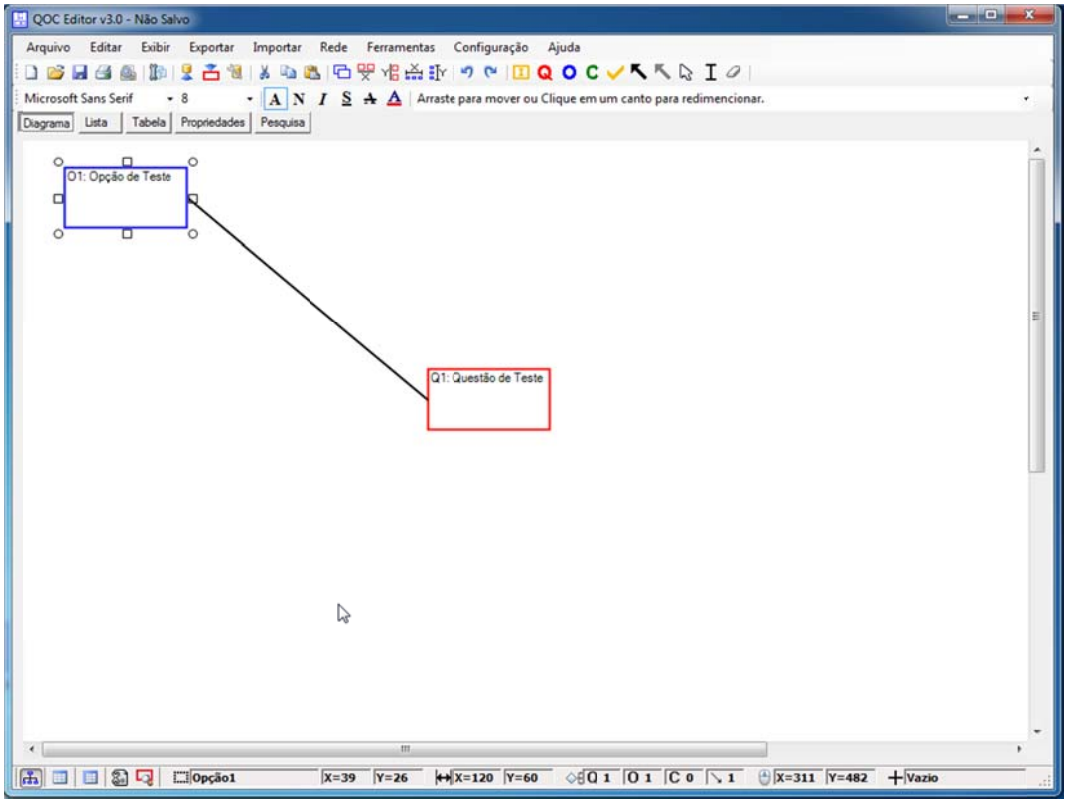

Figura 5.11

Para mover um Nodo, bastar clicar sobre o Nodo e arrastá-lo até a posição desejada:

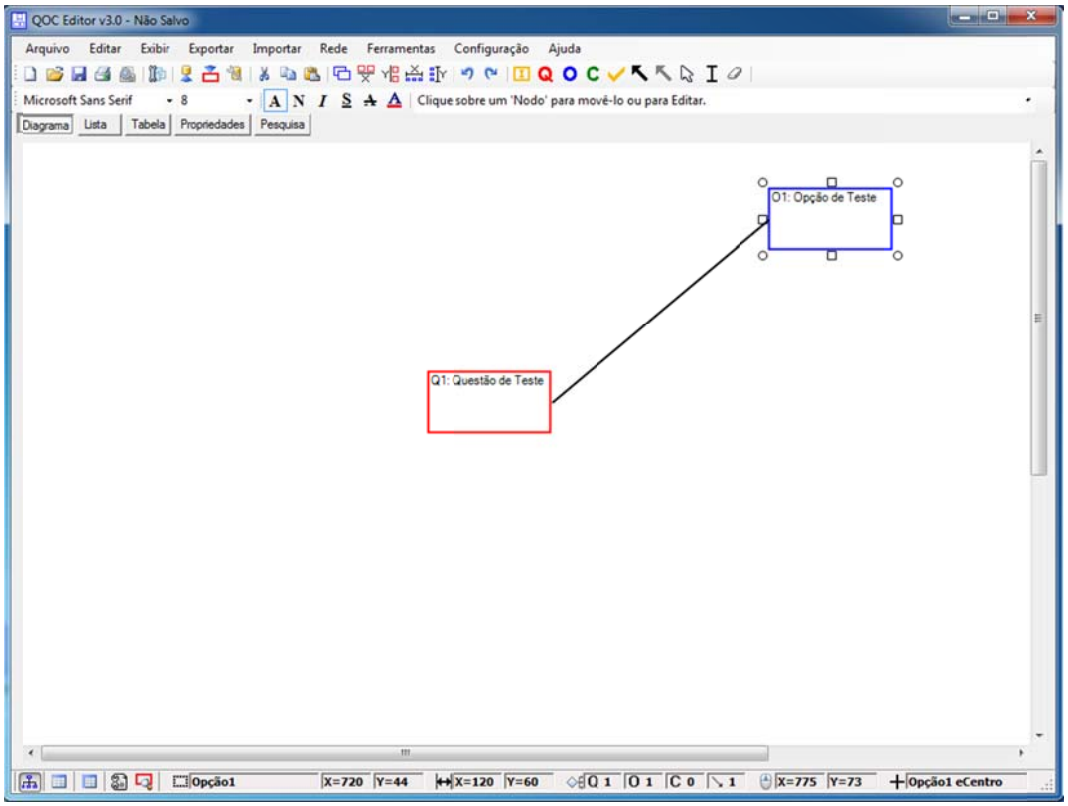

Figura 5.12

Note que Relação é automaticamente redesenhada a partir das bordas mais próximas:

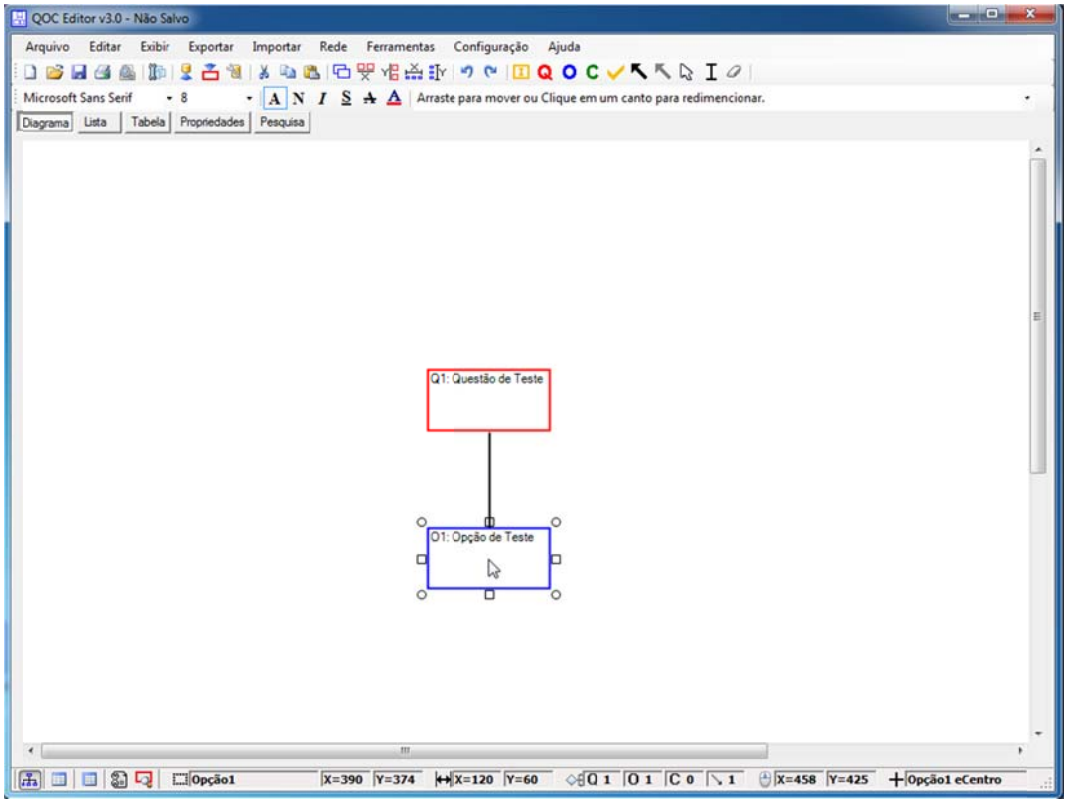

Figura 5.13

Para redimensionar o Nodo, basta clicar na borda correspondente e arrastá-la até a posição desejada:

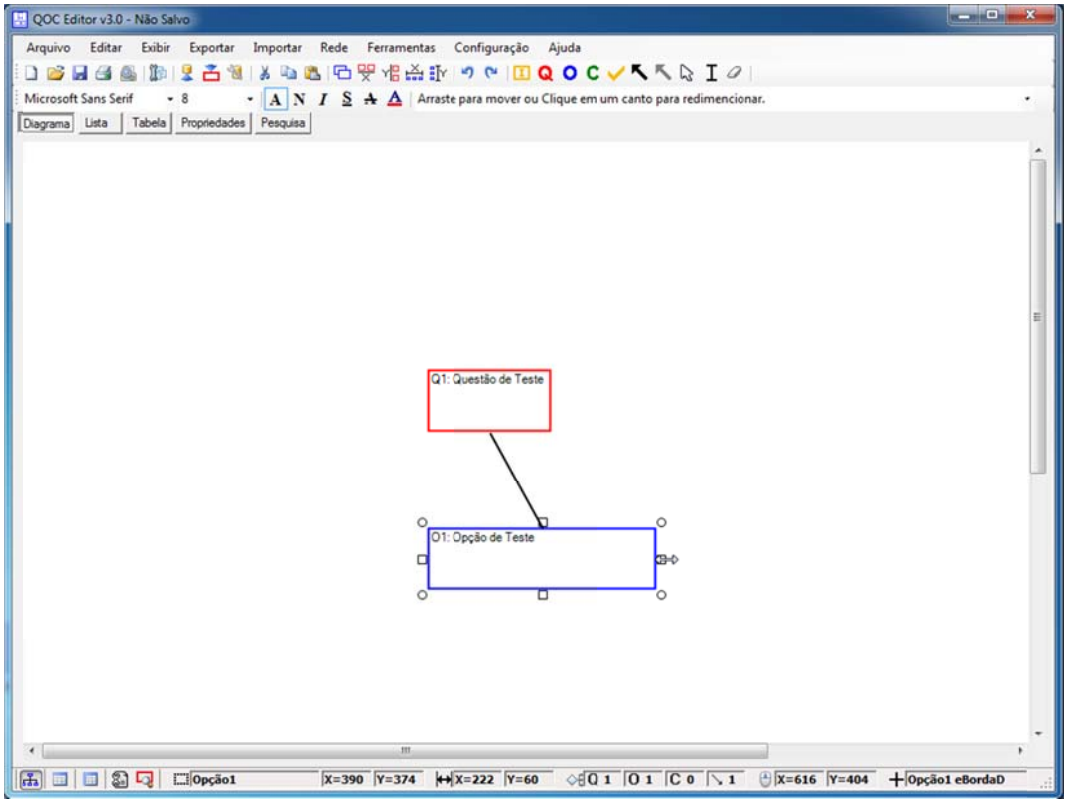

Figura 5.14

O Nodo pode ser aumentado horizontal e/ou verticalmente. Note que o texto é redimensionado ao novo tamanho do Nodo:

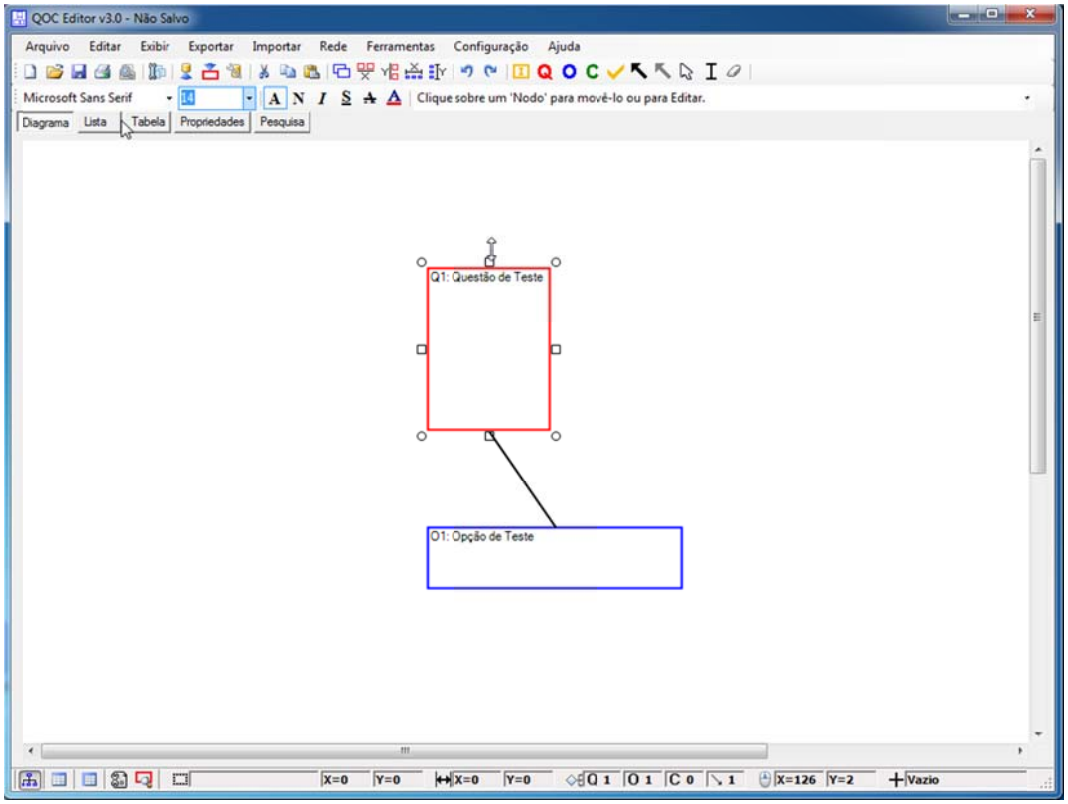

Figura 5.15

Para alterar a fonte de um Nodo, basta escolhermos um das fontes disponíveis no Sistema Operacional, através da caixa de seleção de fonte:

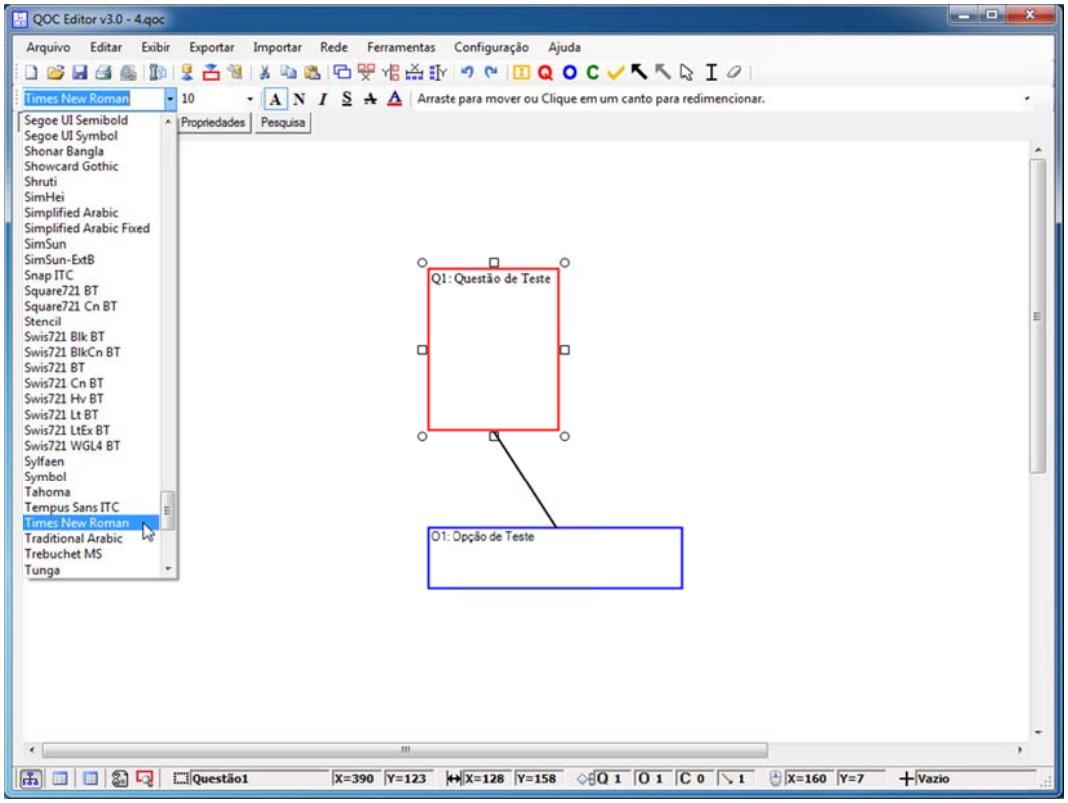

Figura 5.16

Para alterar o tamanho da fonte de um Nodo, basta escolhemos um dos tamanhos de fontes disponíveis, através da caixa de seleção de fonte:

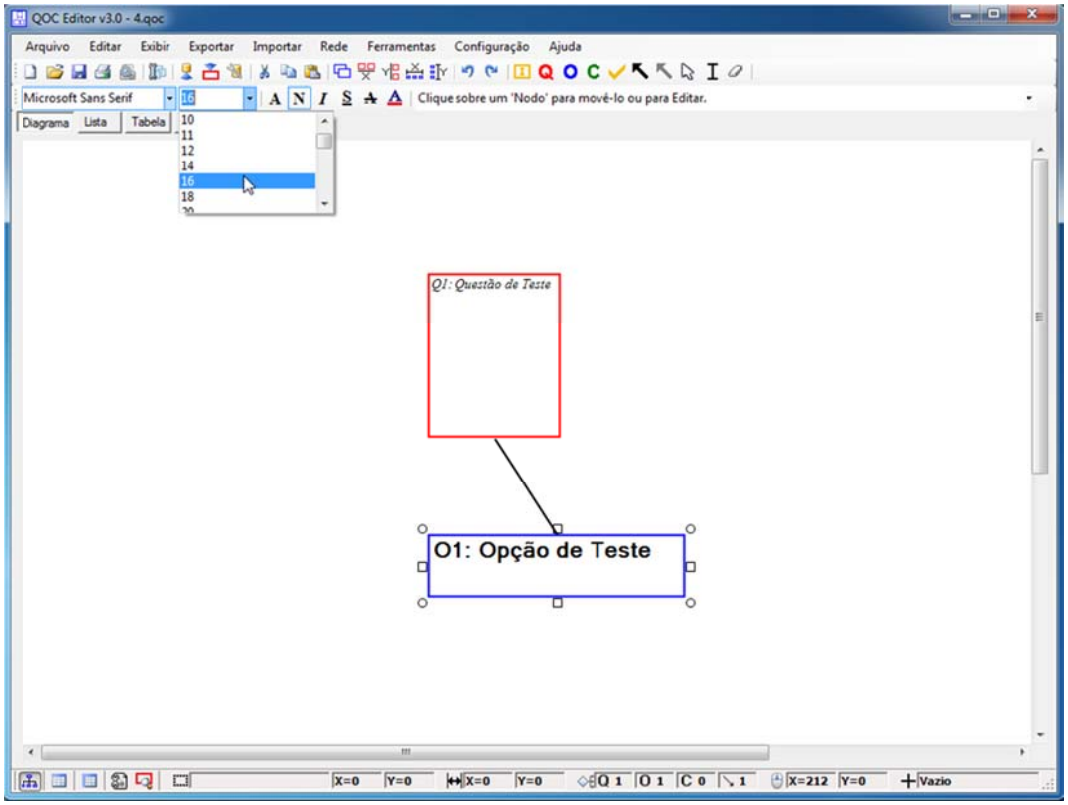

Figura 5.17

Também é possível alterar o estilo da fonte através do botão correspondente:

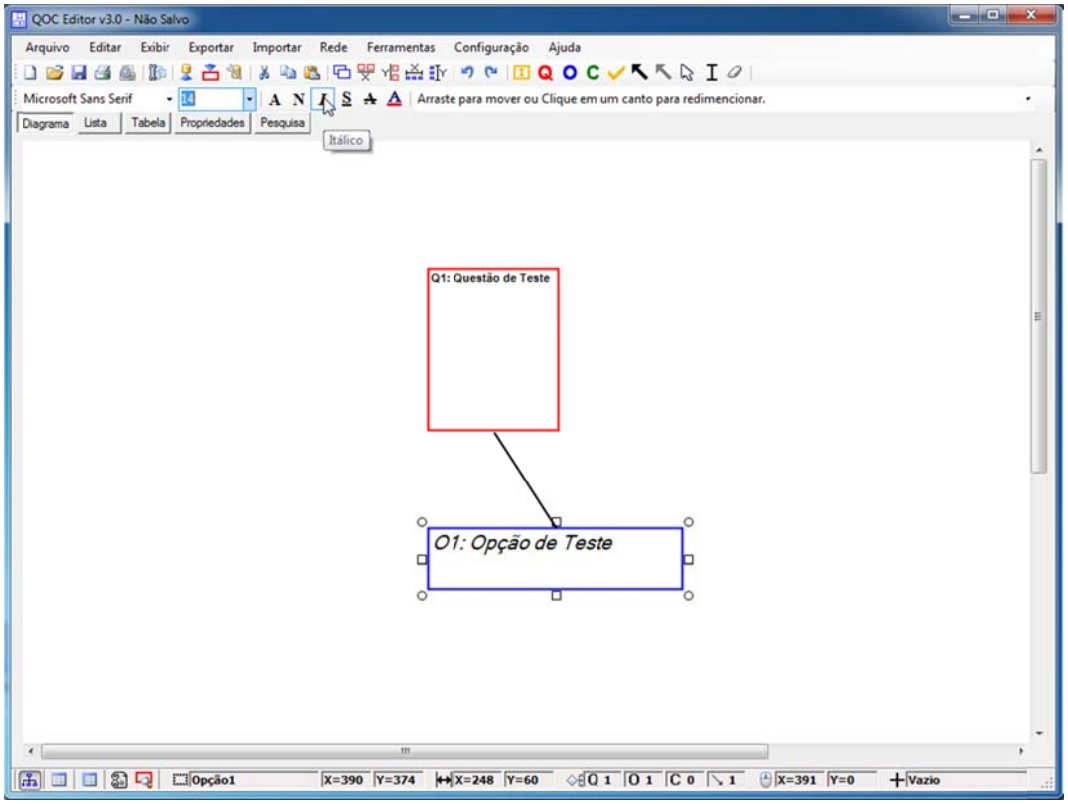

Figura 5.18

5.2 Cenário 2: Modelagem da decisão de compra de dispositivo móvel

Como exemplo, fazemos a modelagem da decisão de compra de dispositivo móvel para leitura de e-mails. Iniciado o aplicativo, ele exibe sua interface:

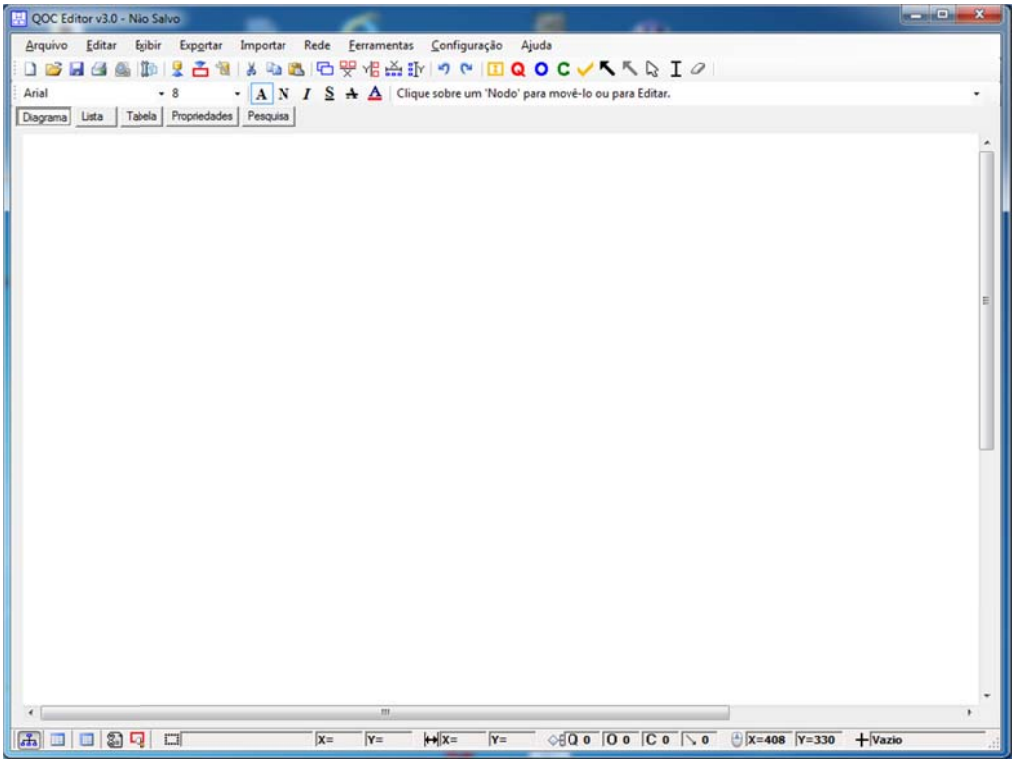

Figura 5.19

Para criar um Diagrama, acessamos o Menu Arquivo e o item Novo:

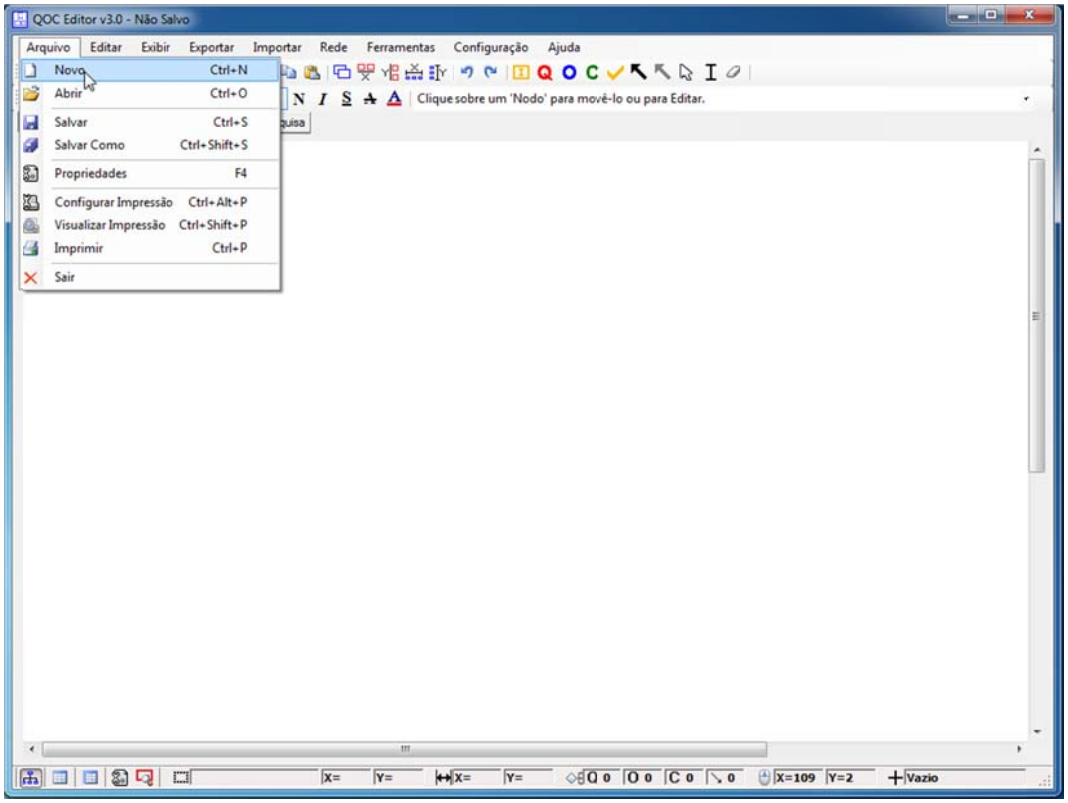

Figura 5.20

O Diagrama deve iniciar pelo Nodo de Início, que é instanciado pela Ferramenta I disponível no Menu Ferramentas ou no Botão I na Barra Ferramentas:

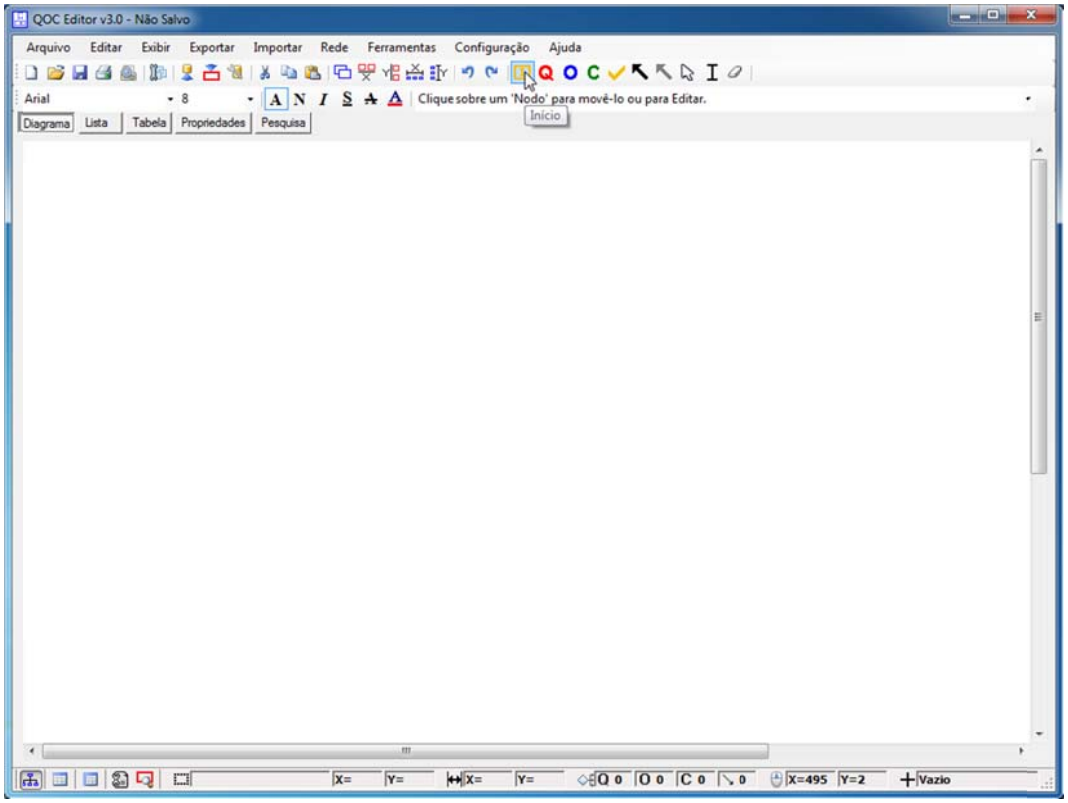

Figura 5.21

Com a Ferramenta I, basta clicar na posição deseja para criar o Nodo Inicio:

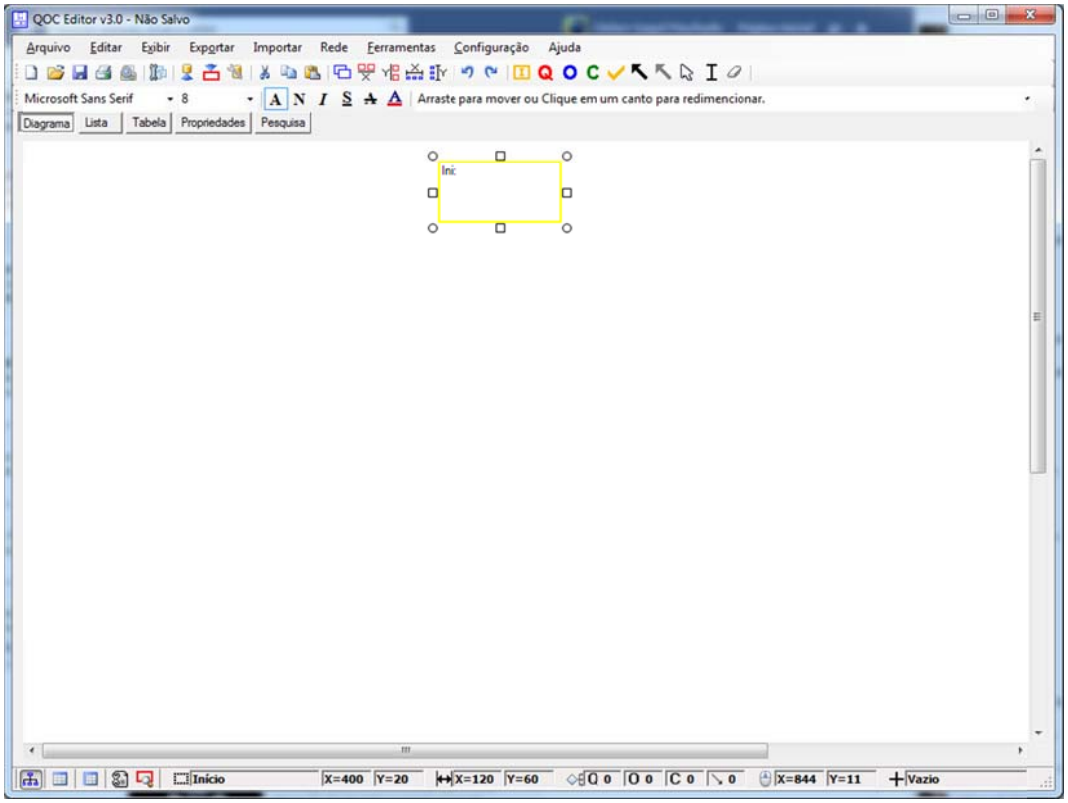

Figura 5.22

Agora vamos escrever o texto no Nodo Inicio para isso usamos a Ferramenta Texto:

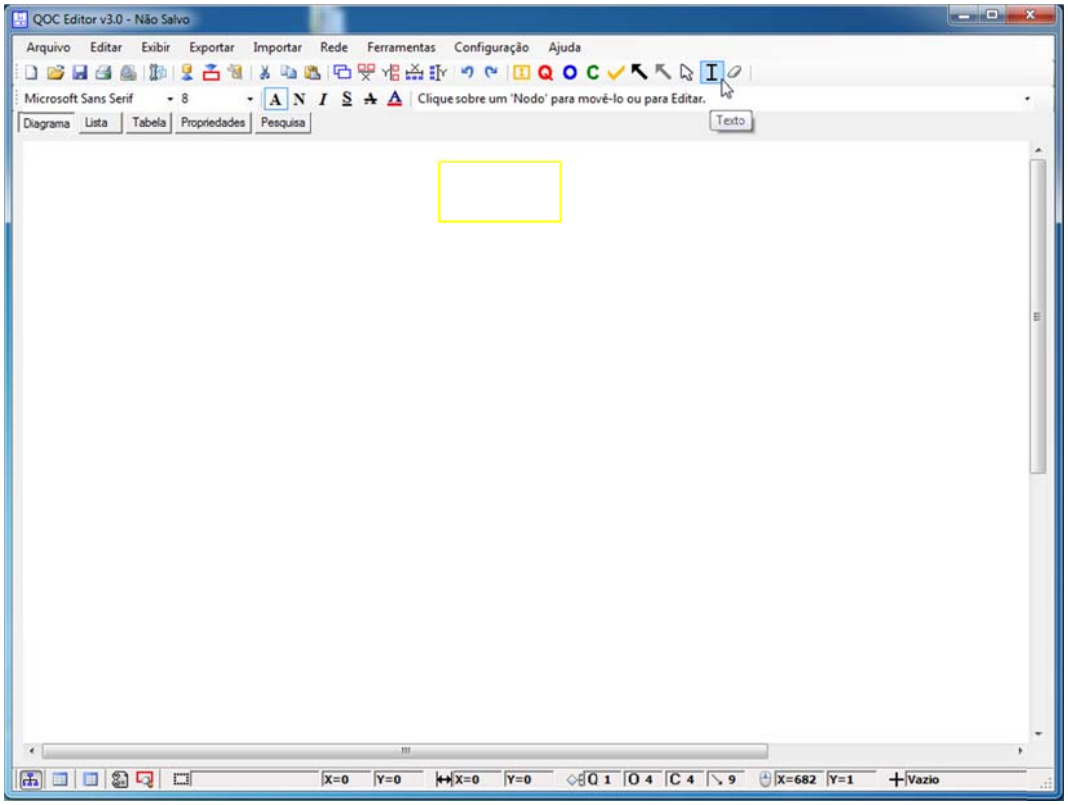

Figura 5.23

Digitamos o Texto e, então, pressionamos Enter para encerrar a edição de texto:

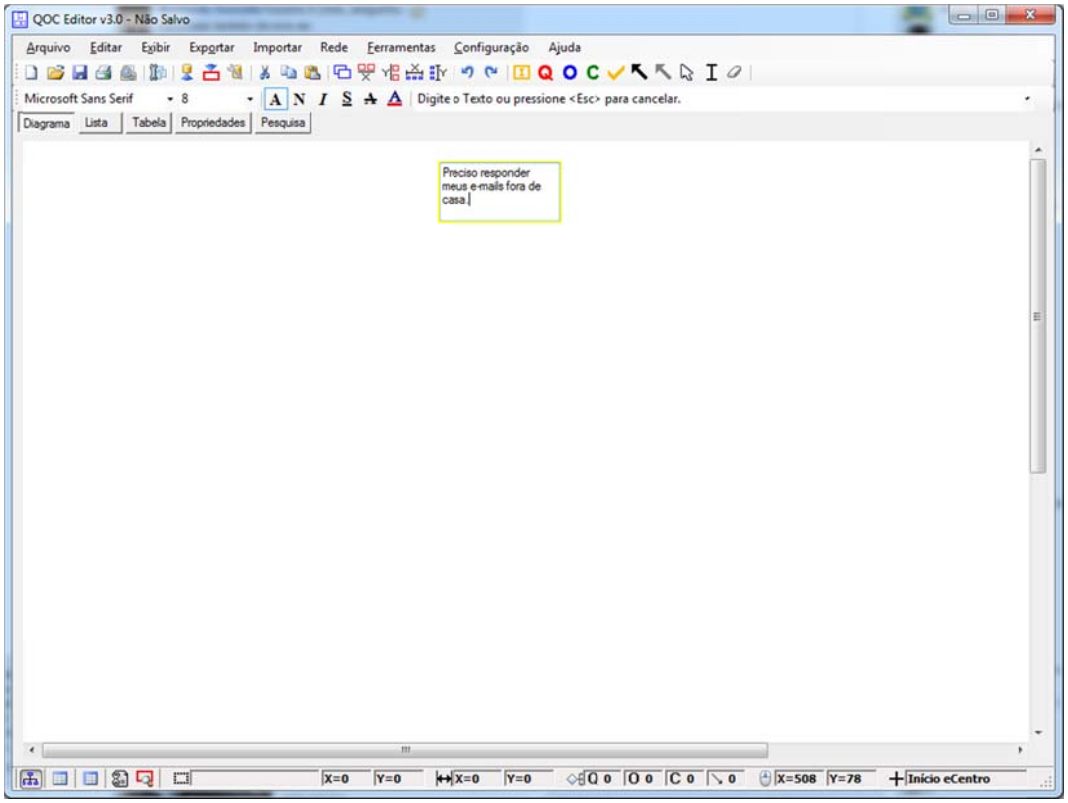

Figura 5.24

Agora vamos criar um Nodo Questão, para isso usamos a Ferramenta Q:

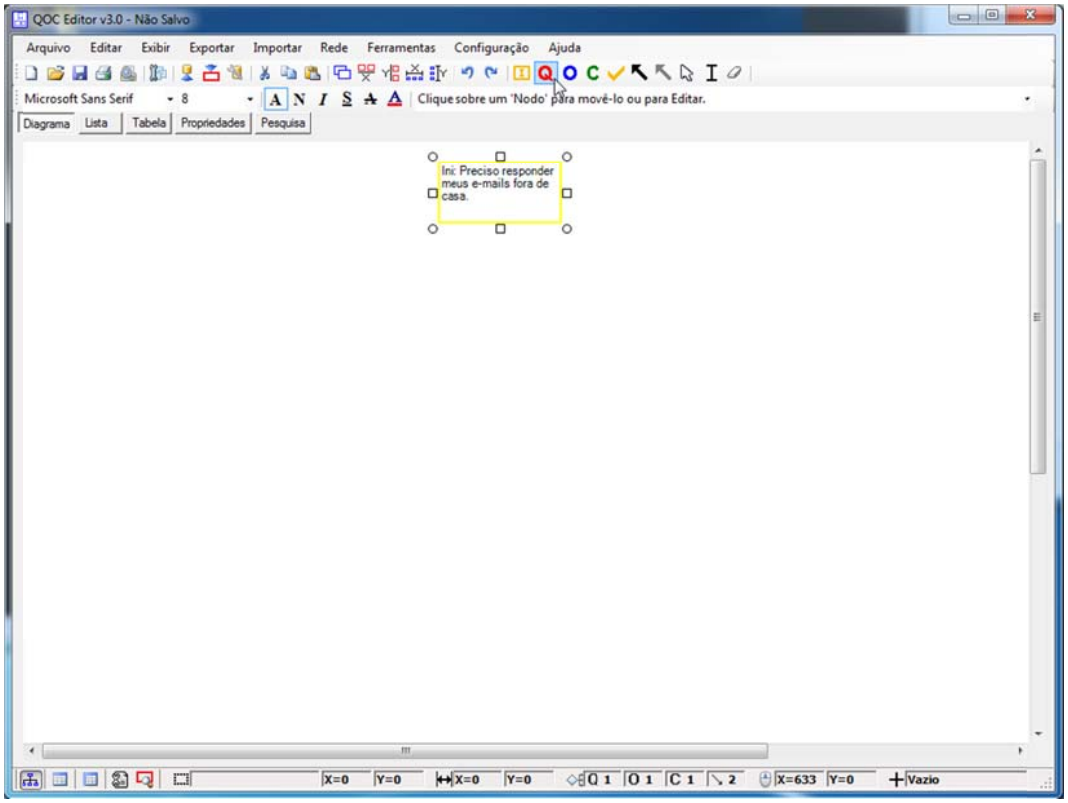

Figura 5.25

Com a Ferramenta Q, basta clicar na posição deseja para criar o Nodo Questão e, após, usamos Ferramenta Texto para escrever o Texto no Nodo Questão 1:

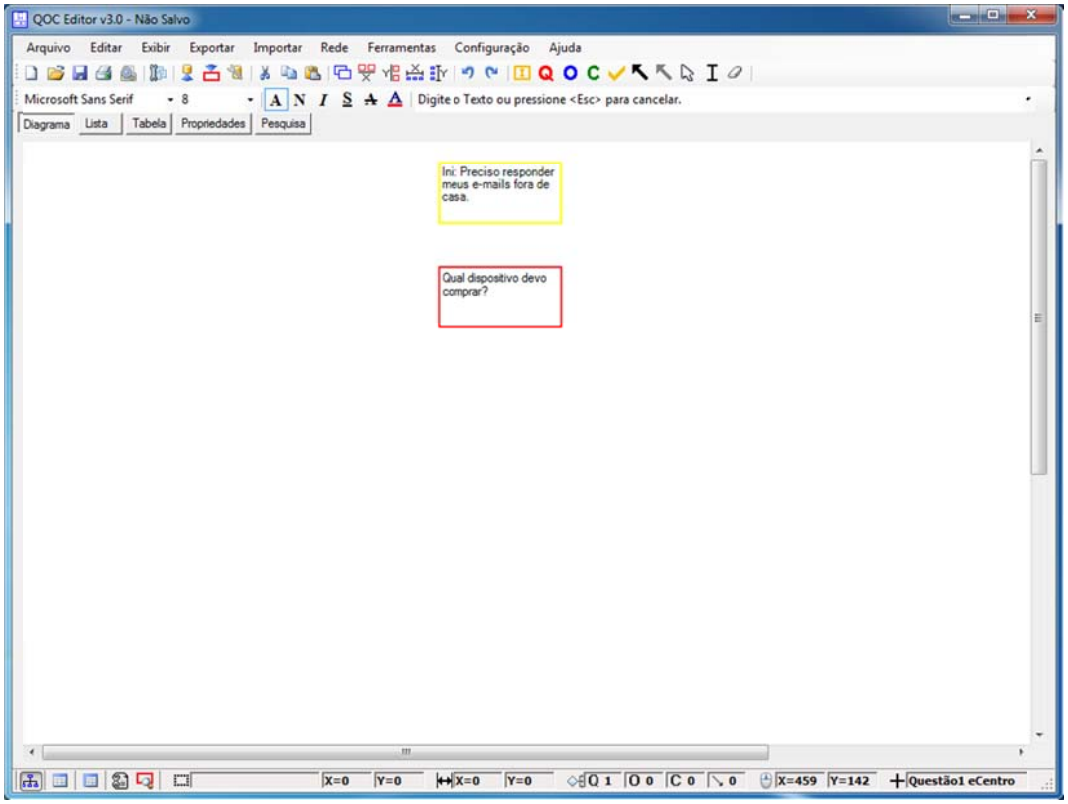

Figura 5.26

Agora iremos relacionar o Nodo de Inicio com o Nodo Questão 1, utilizando a Ferramenta Relação Positiva:

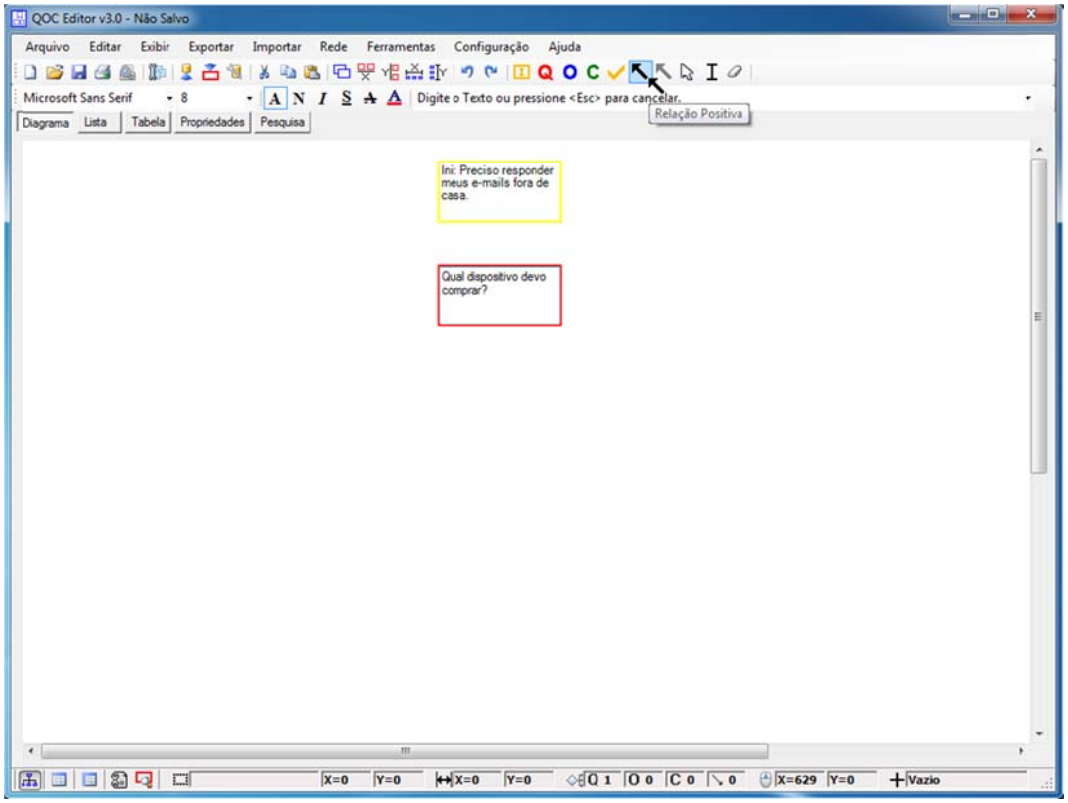

Figura 5.27

Para criar a Relação, basta clicar no Nodo de Inicio e arrastar até o Nodo Questão 1:

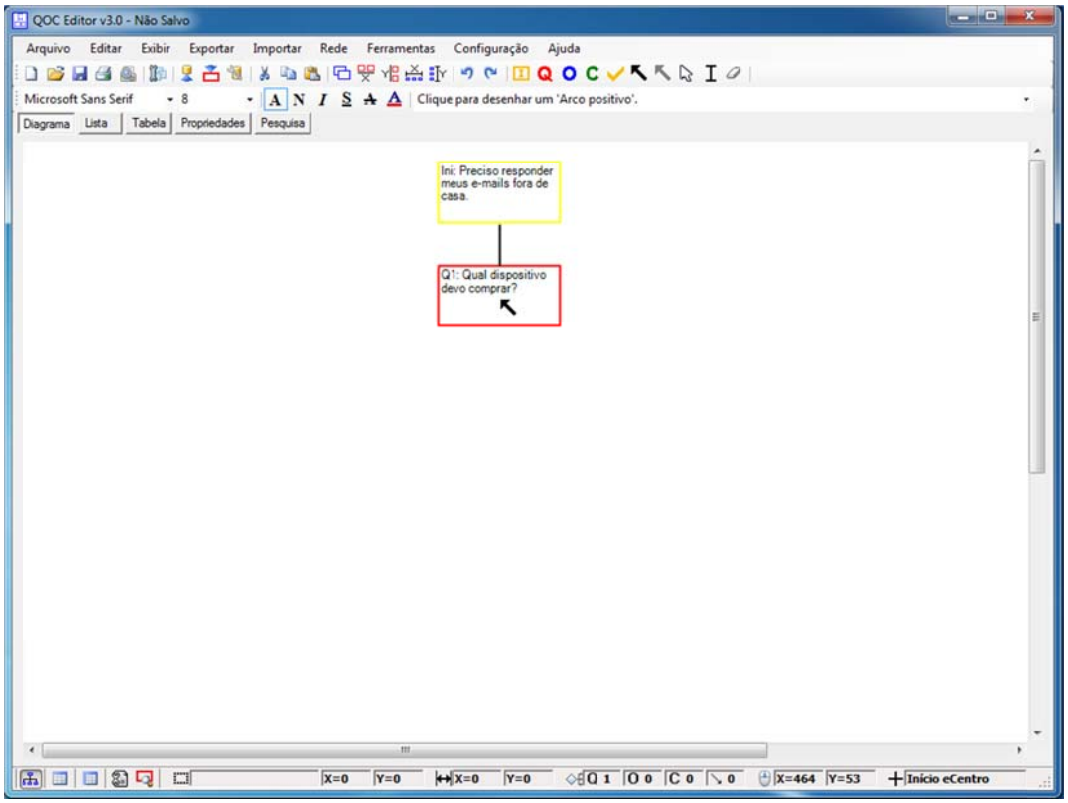

Figura 5.28

Agora vamos criar a primeira Opção usando a Ferramenta O:

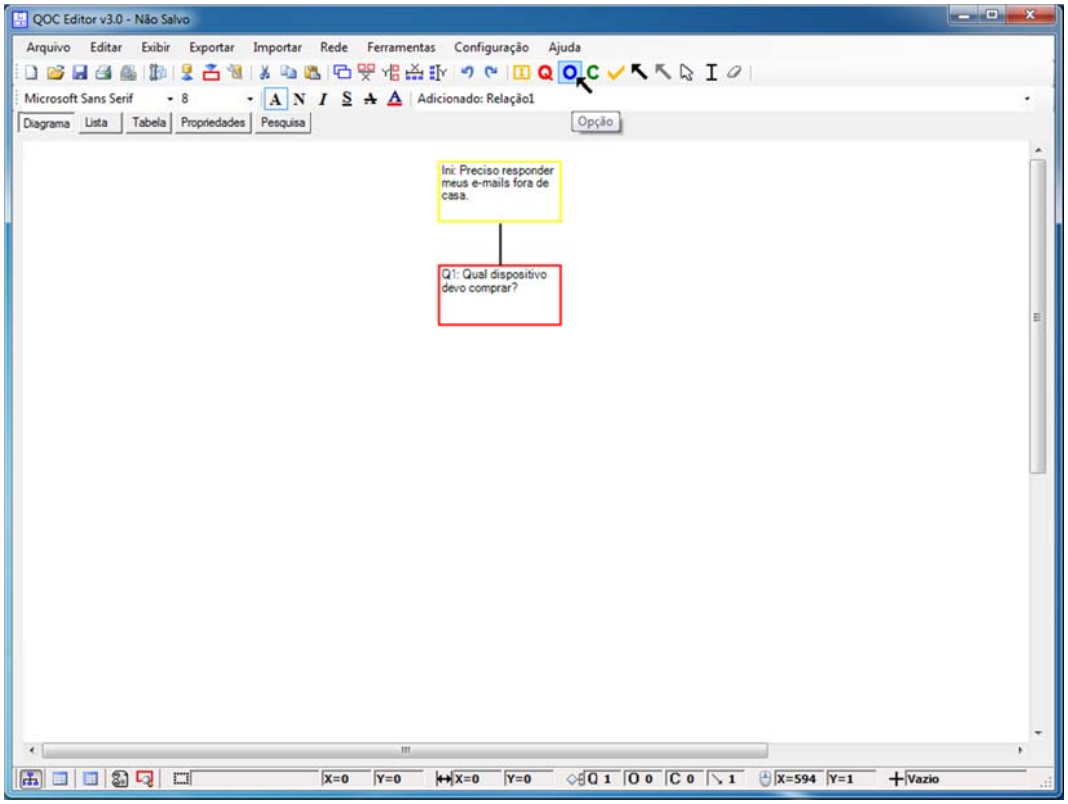

Figura 5.29

Com a Ferramenta O, basta clicar na posição deseja para criar a Opção 1 e, após, usar a Ferramenta Texto para escrever o Texto no Nodo Opção 1:

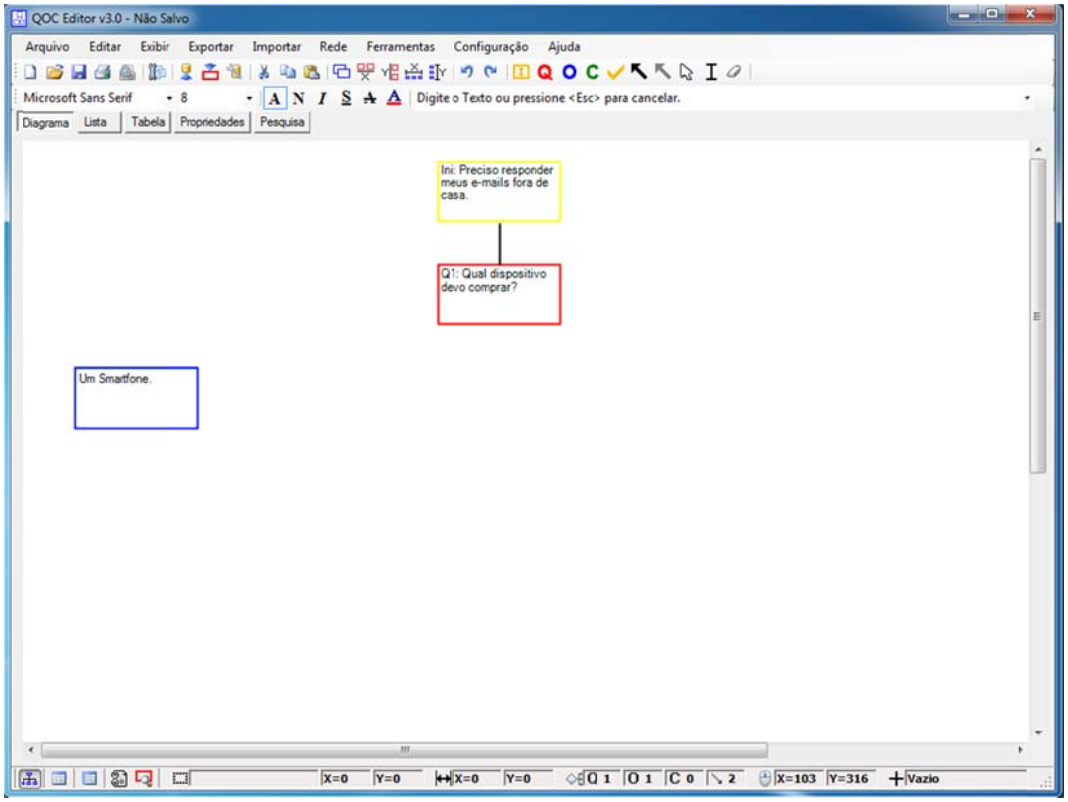

Figura 5.30

Agora iremos relacionar o Nodo Questão 1 com o Nodo Opção 1, para usamos a Ferramenta Relação Positiva, clicamos no Nodo Questão 1 e arrastamos até o Nodo Opção 1:

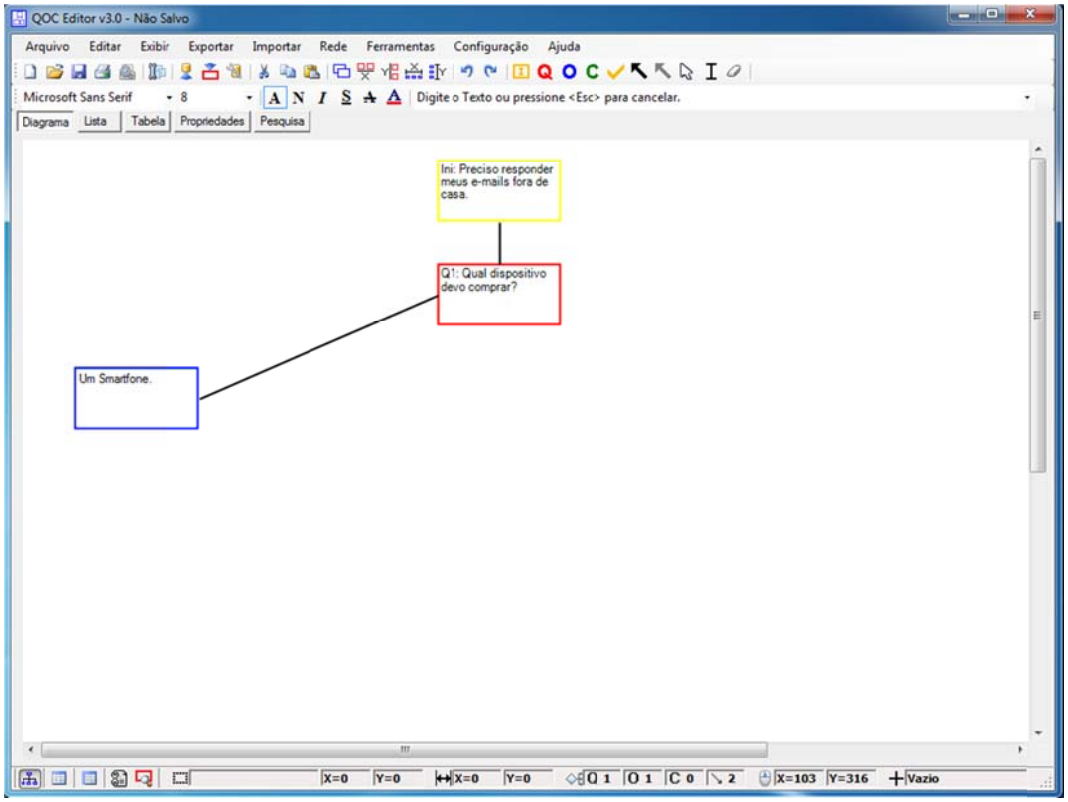

Figura 5.31

Agora vamos criar o primeiro Nodo Critério para isso usamos a Ferramenta C:

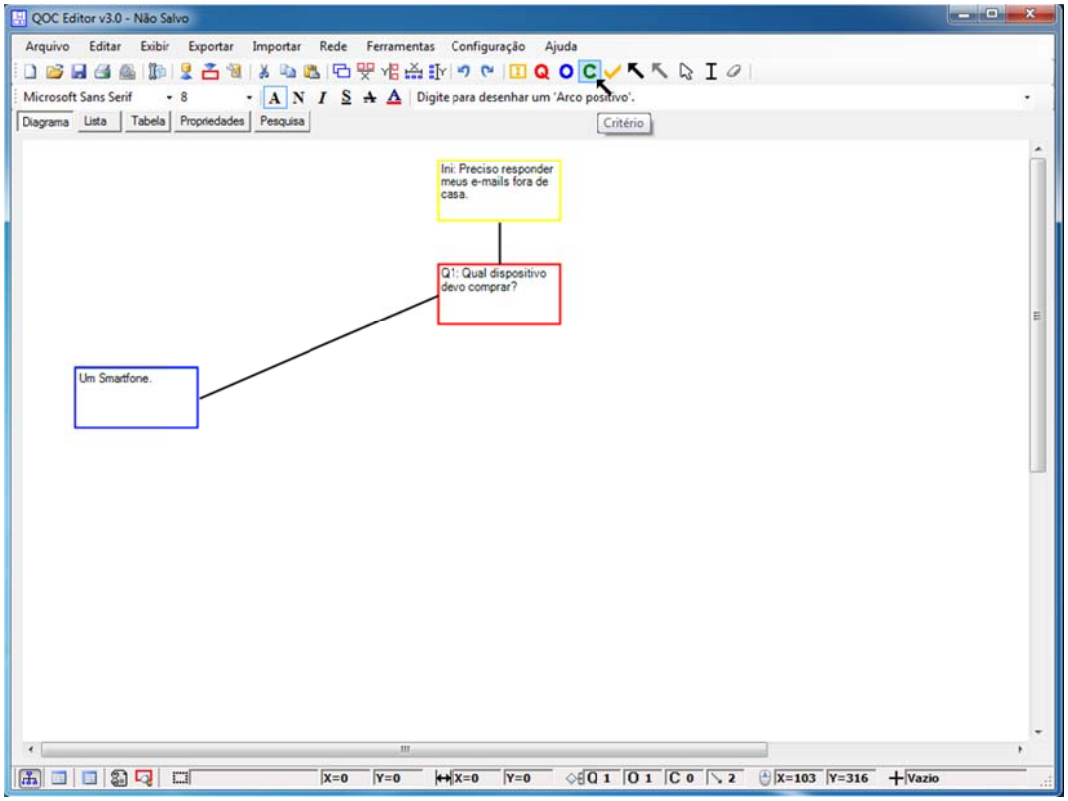

Figura 5.32

Com a Ferramenta C basta clicar na posição desejada para criar o Nodo Critério 1 e, após, usamos Ferramenta Texto para escrever o Texto no Nodo:

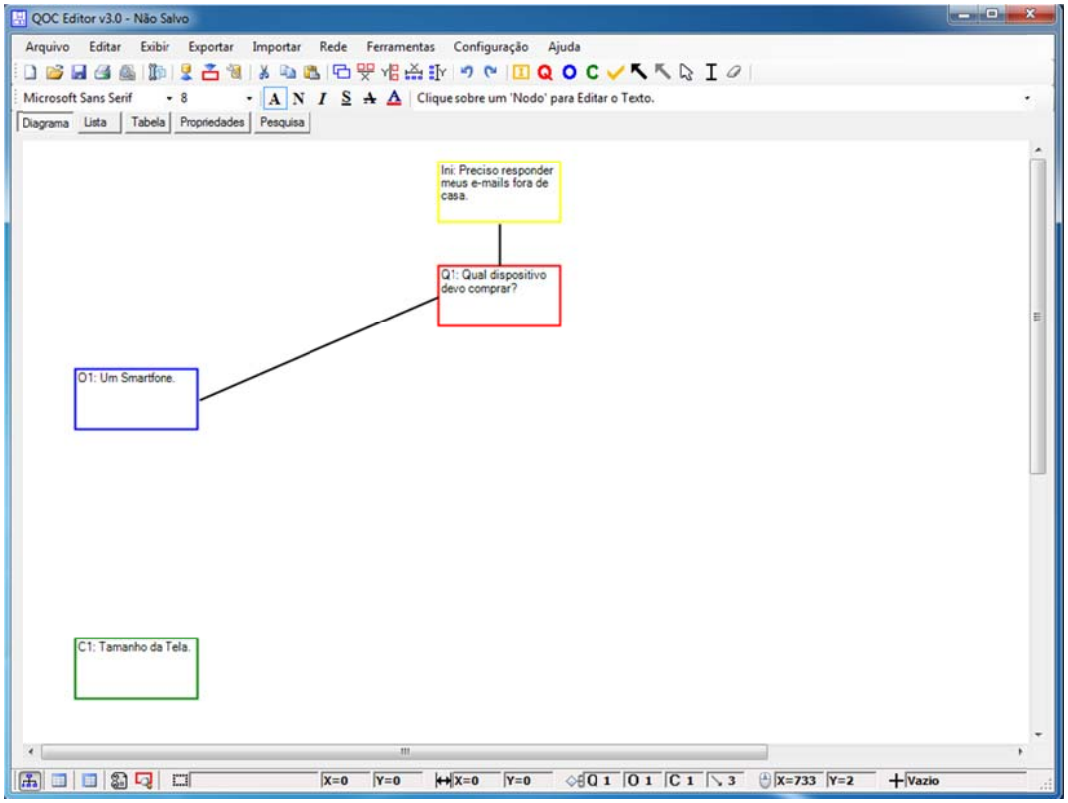

Figura 5.33

Agora iremos relacionar o Nodo Opção 1 com Nodo Critério 1. Para isso, selecionamos a Ferramenta Relação Negativa:

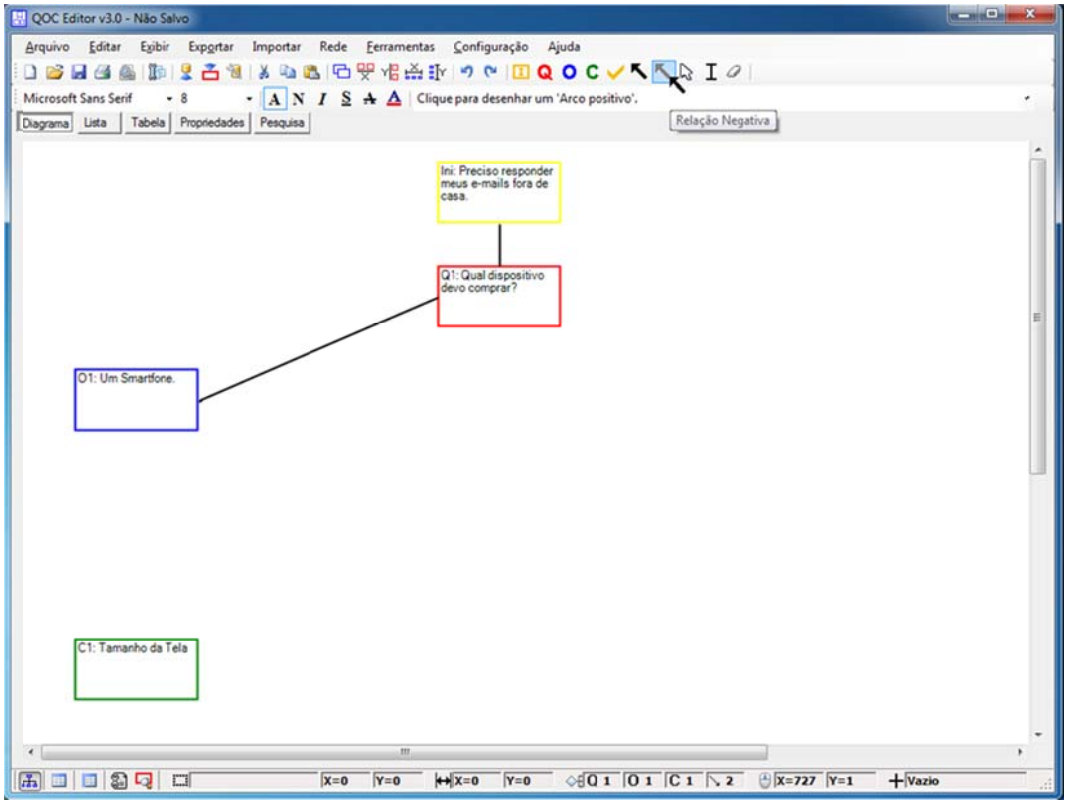

Figura 5.34

Com a Ferramenta Relação Negativa, clicamos no Nodo Opção 1 e arrastamos até o Nodo Critério 1 para relacionar os Nodos:

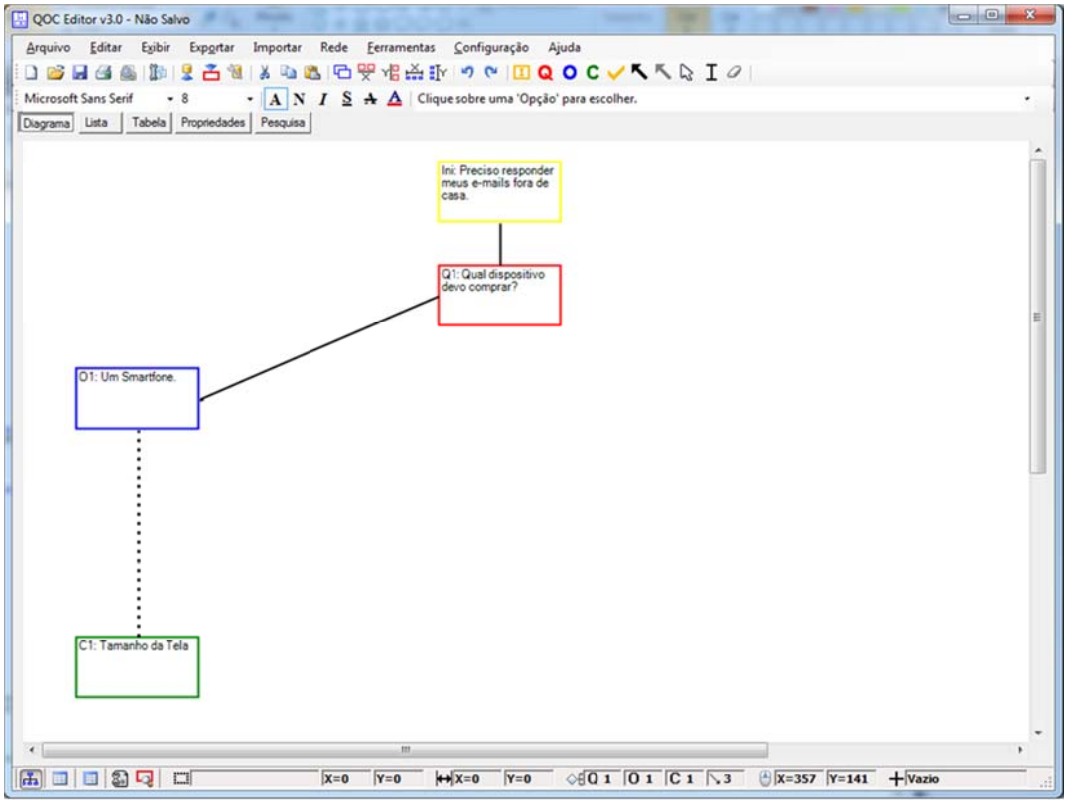

Figura 5.35

Repetimos o processo para criar Nodos Opção 2, Opção 3, Opção 4, Critério 2, Critério 3 Critério 4

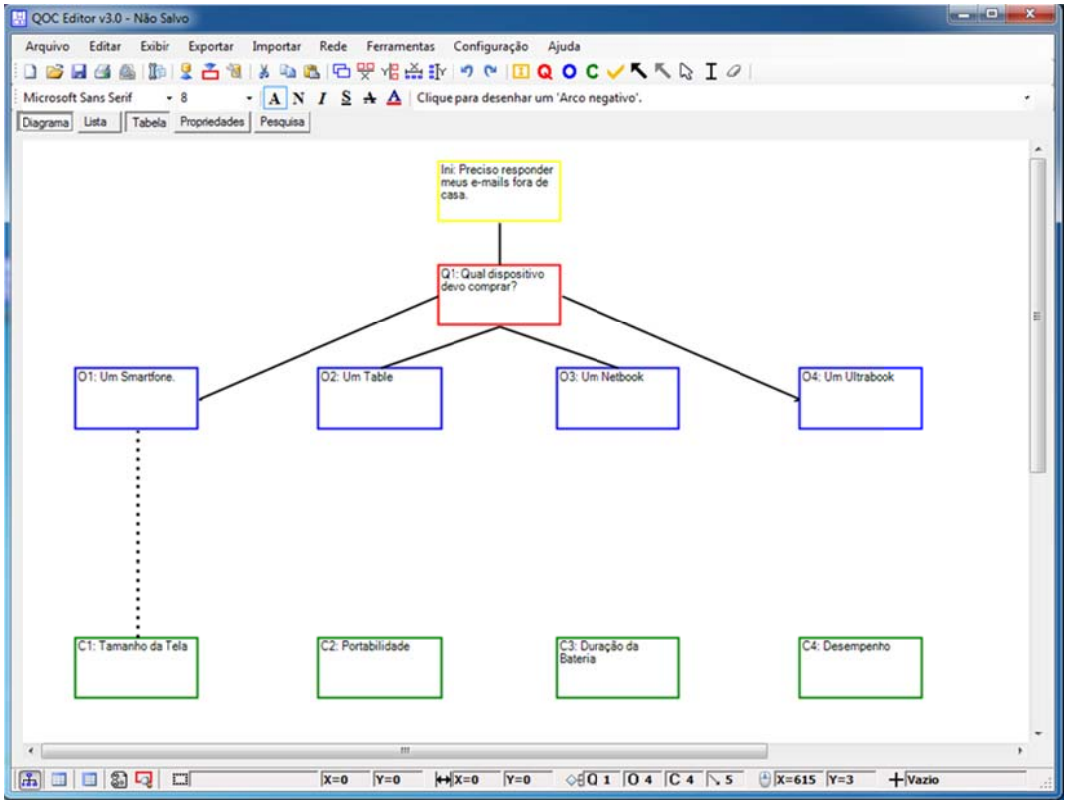

Figura 5.36

Com a Ferramenta Relação Positiva e a Ferramenta Relação Negativa, respectivamente, relacionamos os critérios as opções:

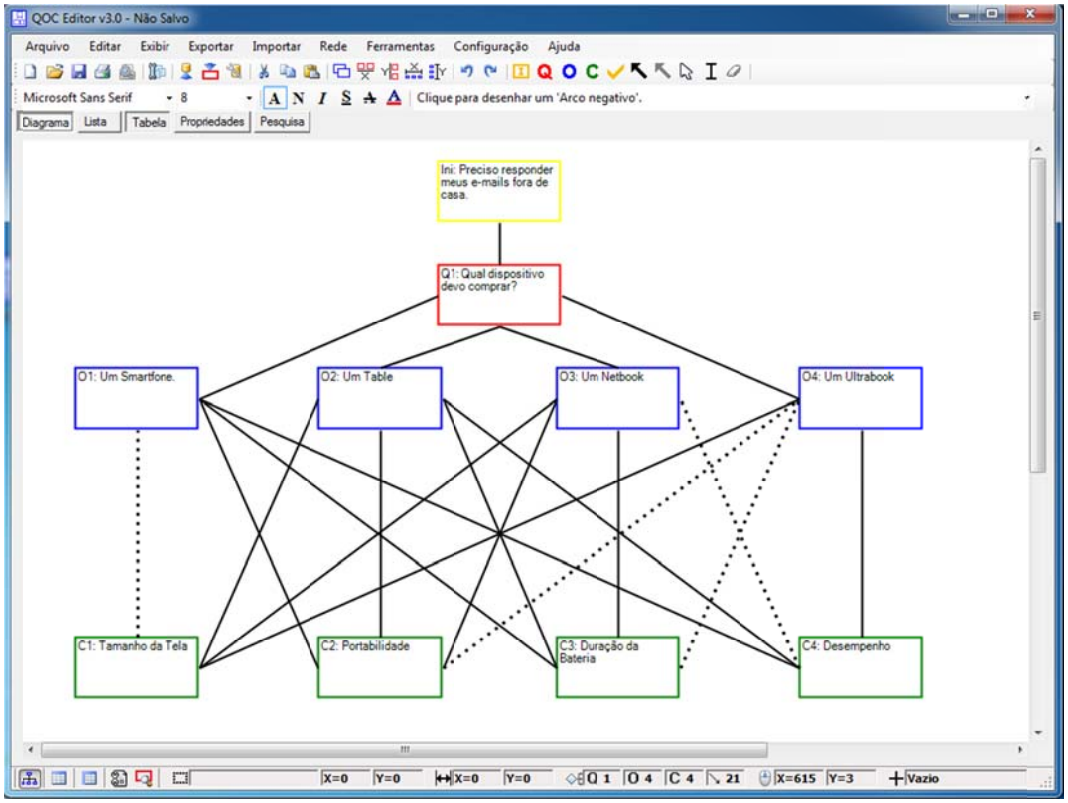

Figura 5.37

Para marcar a opção escolhida, selecionamos a Ferramenta Escolher uma Opção:

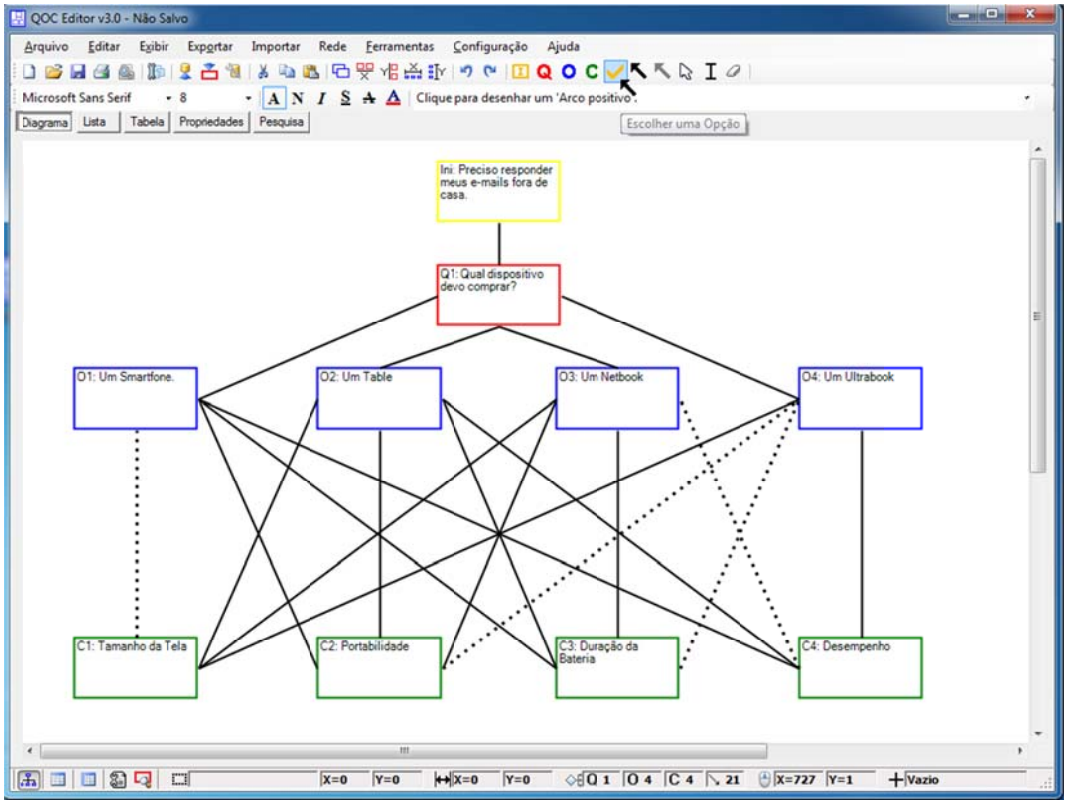

Figura 5.38

Com a Ferramenta Escolher uma Opção, clicamos sobre o Nodo da opção escolhida para marcá-lo:

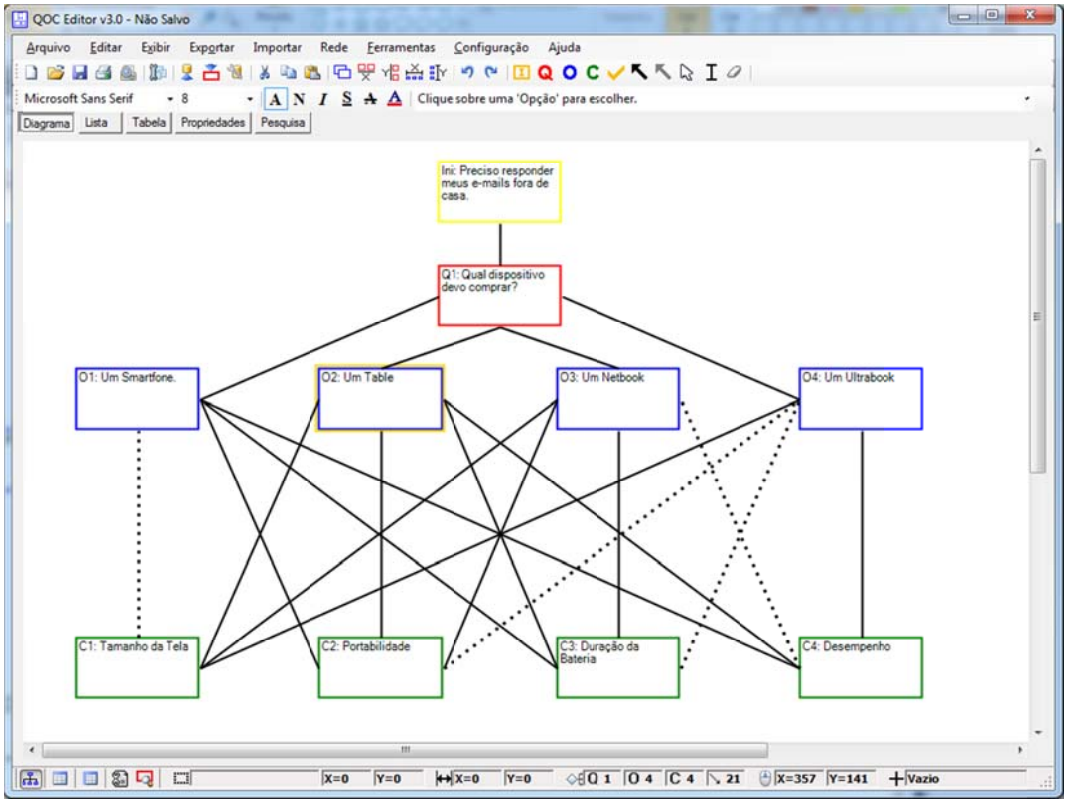

Figura 5.39

Por fim, para salvar o Diagrama, acessamos o Menu Arquivo item Salvar Como:

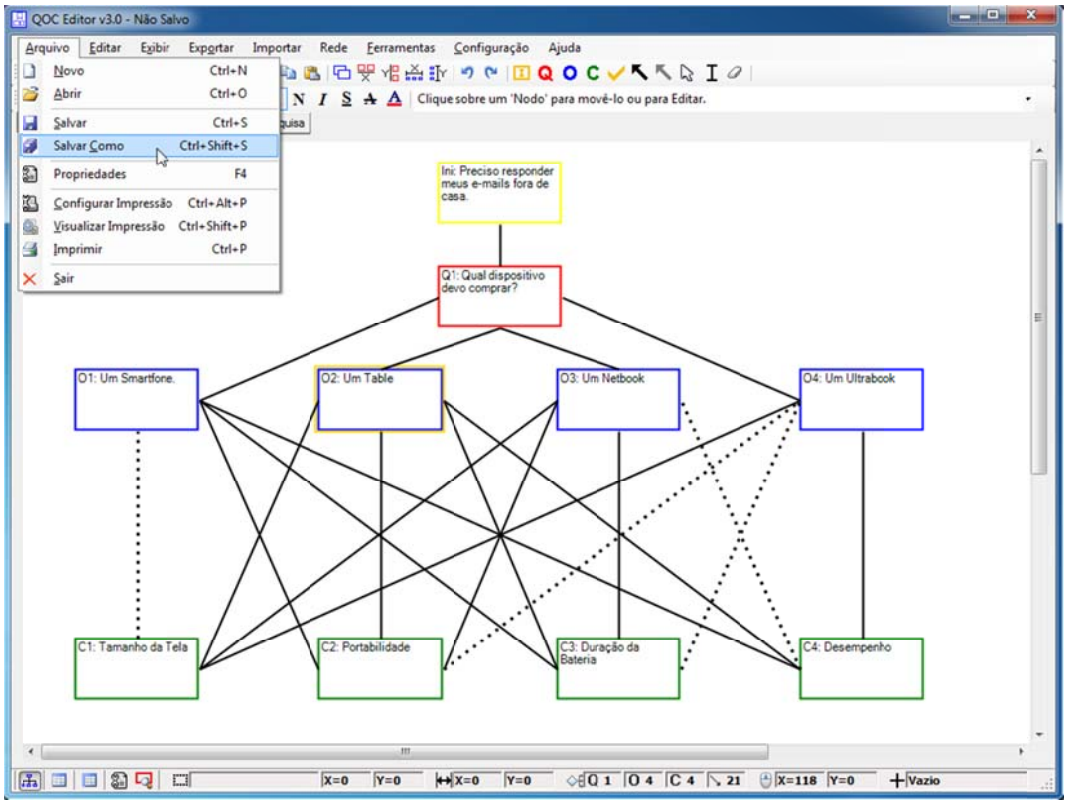

Figura 5.40

Em seguida, digitamos um nome para o diagrama e clicamos em salvar:

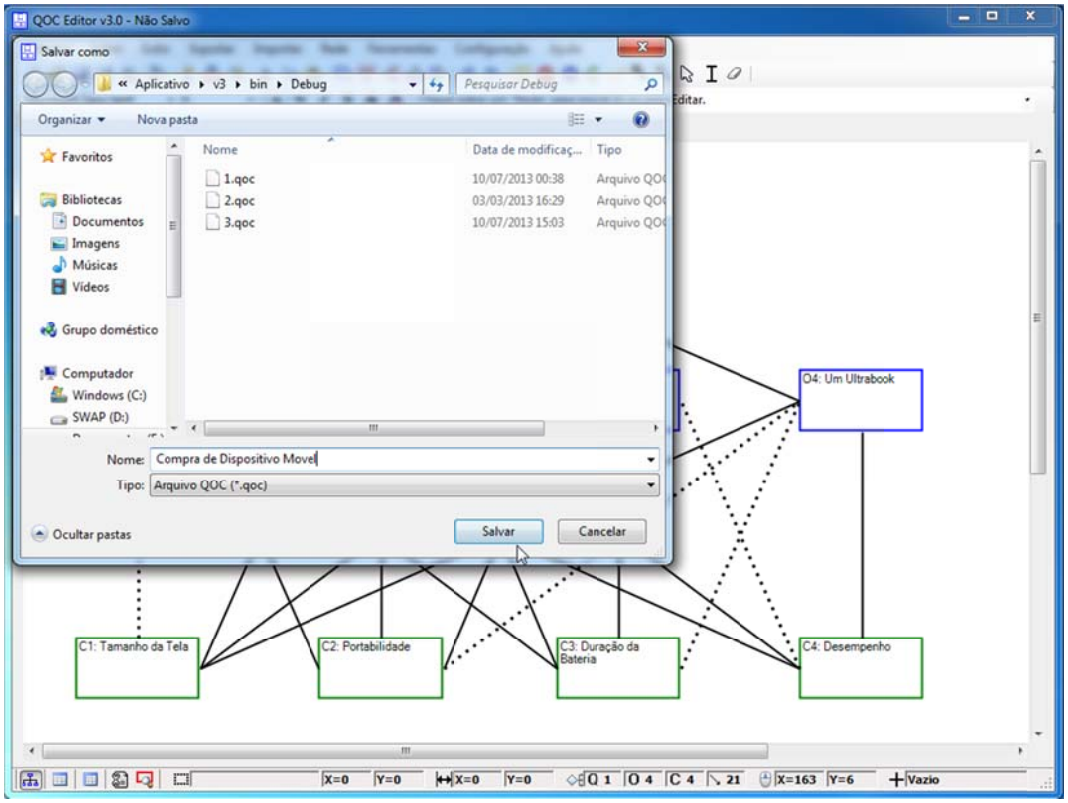

Figura 5.41

5.3 Cenário 3: Abrir um diagrama QOC de arquivo e exportar JPG

Como segundo exemplo, abriremos um diagrama QOC e exportaremos para imagem JPG. Iniciado o aplicativo, ele exibe sua interface:

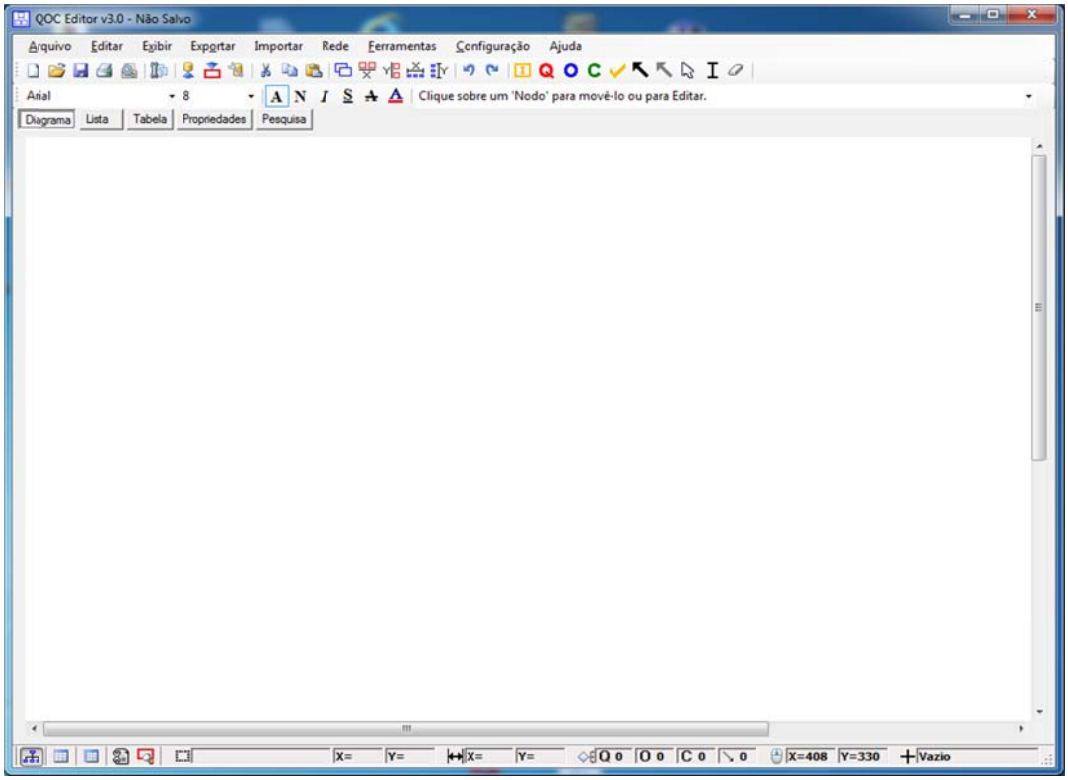

Figura 5.42

Para abrir um Diagrama, acessamos o Menu Arquivo e a opção Abrir:

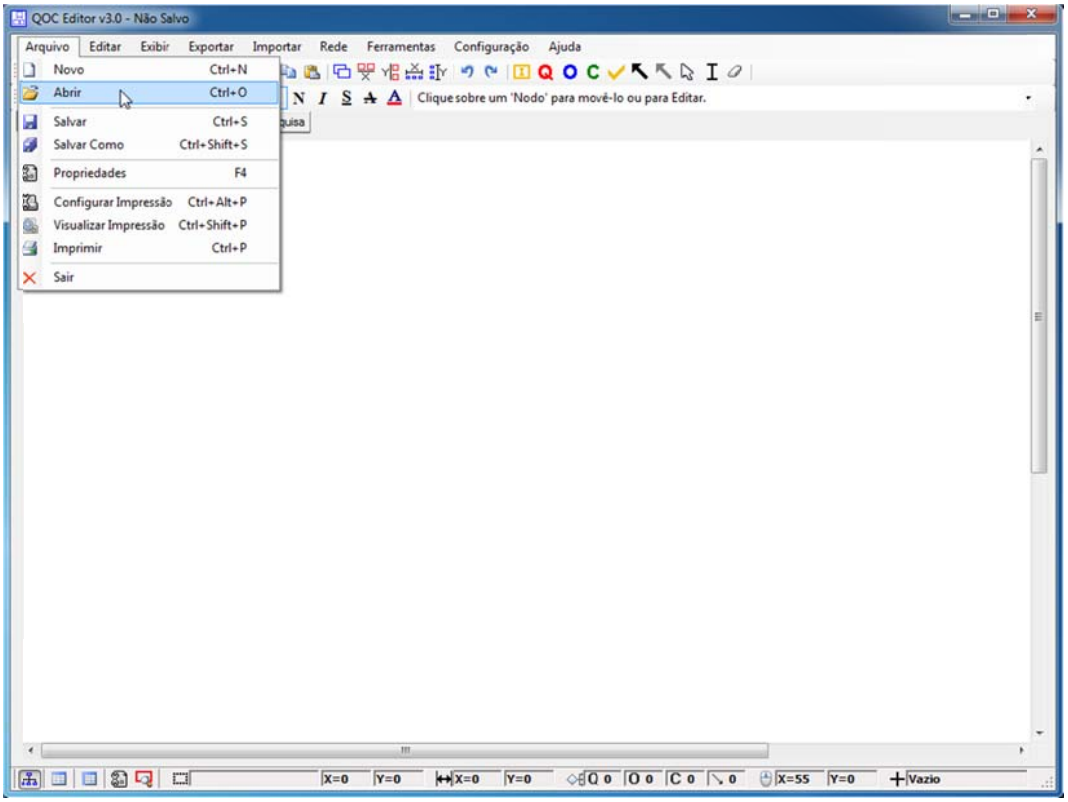

Figura 5.43

Então, selecionamos o arquivo QOC que contém o Diagrama a ser aberto:

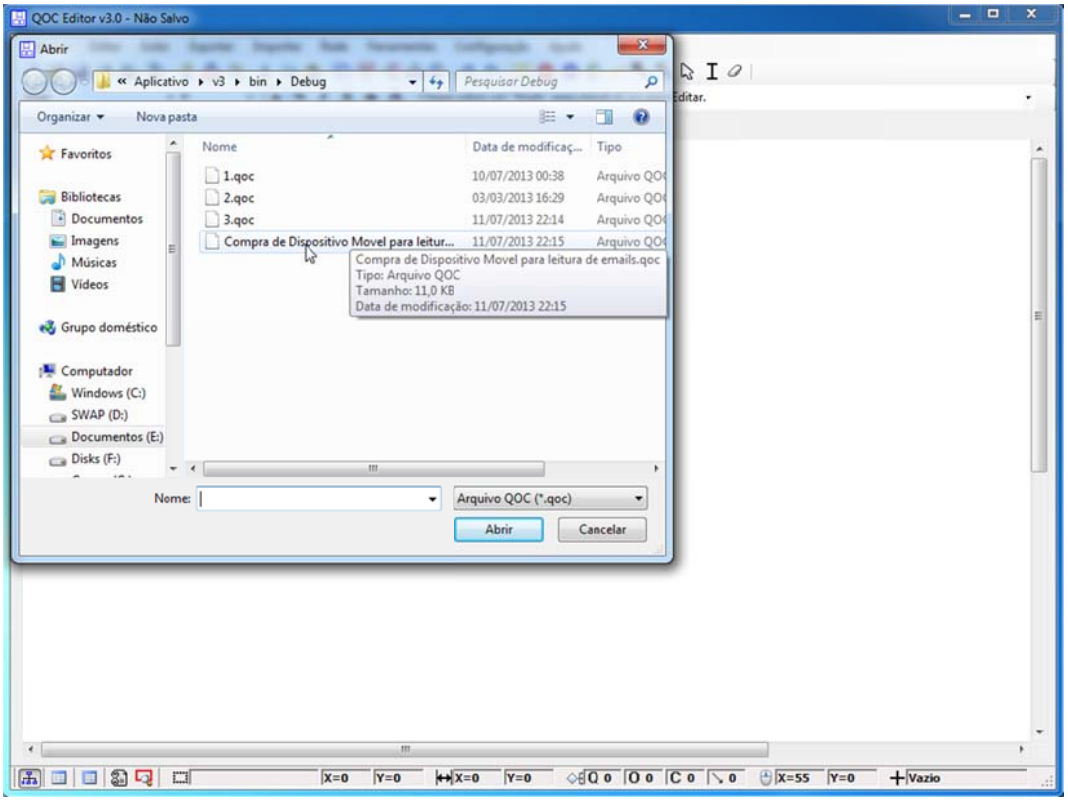

Figura 5.44

Assim, o Diagrama é carregado e exibido:

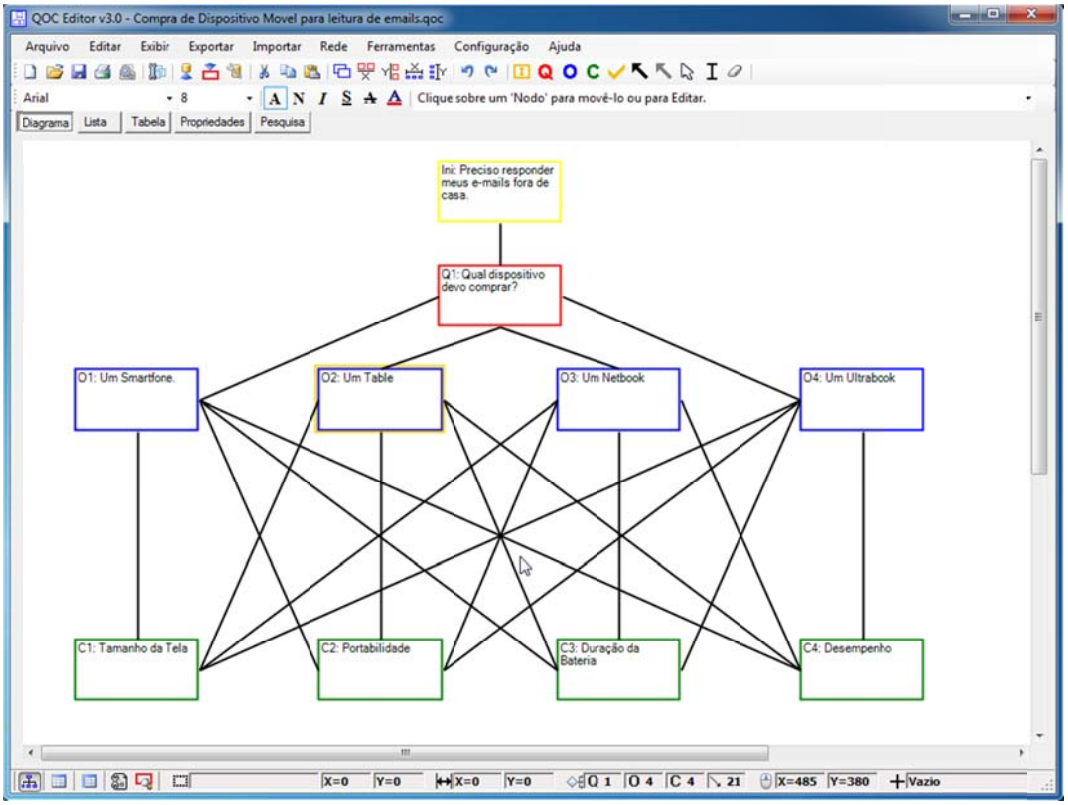

Figura 5.45

Para exportar o Diagrama, o Menu Exportar e o item Exportar Imagem:

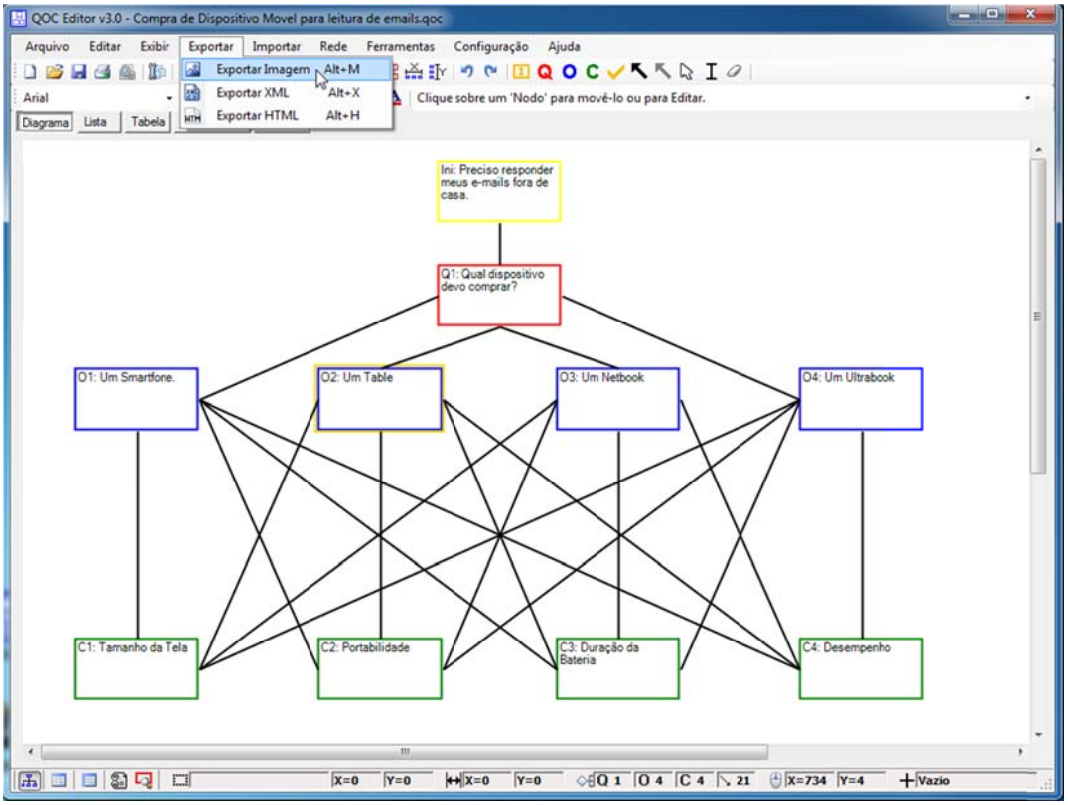

Figura 5.46

Então, digitamos o nome do arquivo a ser criado, escolhemos o formato da imagem entre os Formatos BMP, JPG ou PNG, e clicamos no botão salvar:

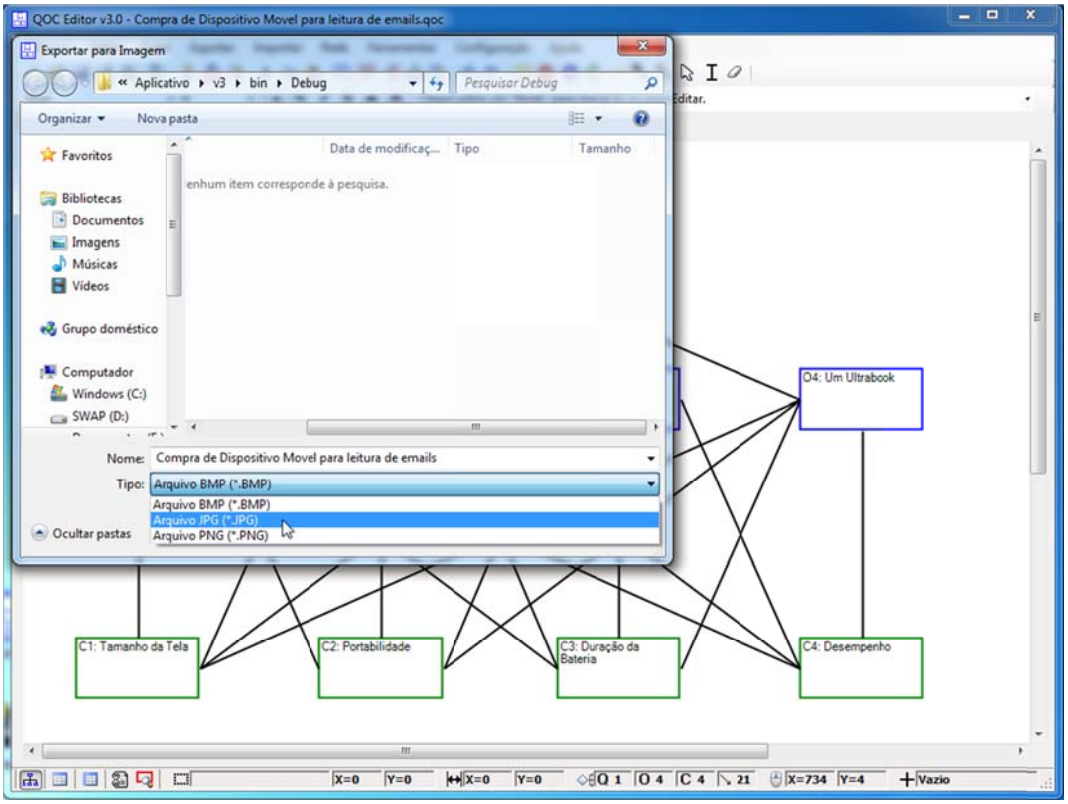

Figura 5.47

Após, o programa gera a imagem, que pode ser aberta por qualquer visualizador de imagem e usada em outros editores:

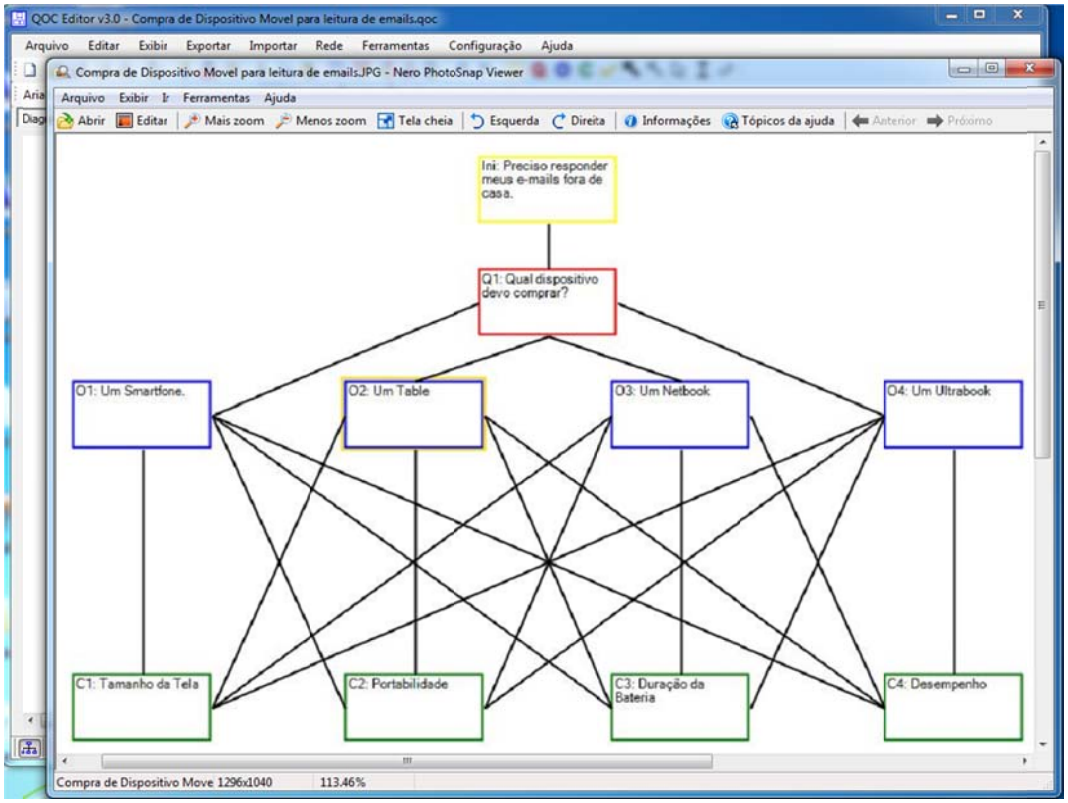

Figura 5.48

5.4 Cenário 4: Importar um diagrama QOC de um XML e imprimir

Como quarto exemplo, importaremos um diagrama QOC de um XML e imprimiremos através de uma impressora instalada. Iniciado o aplicativo, ele exibe sua interface:

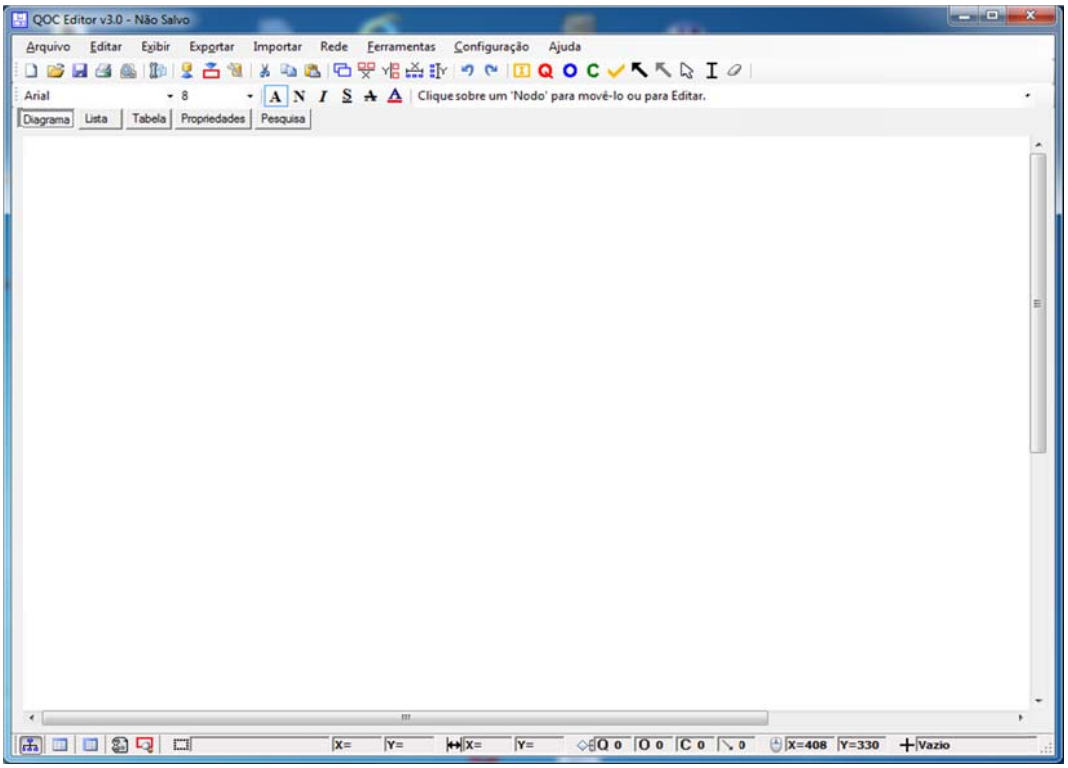

Figura 5.49

Para importar um Diagrama de um arquivo XML, acessamos o Menu Importar e o Item Importar XML:

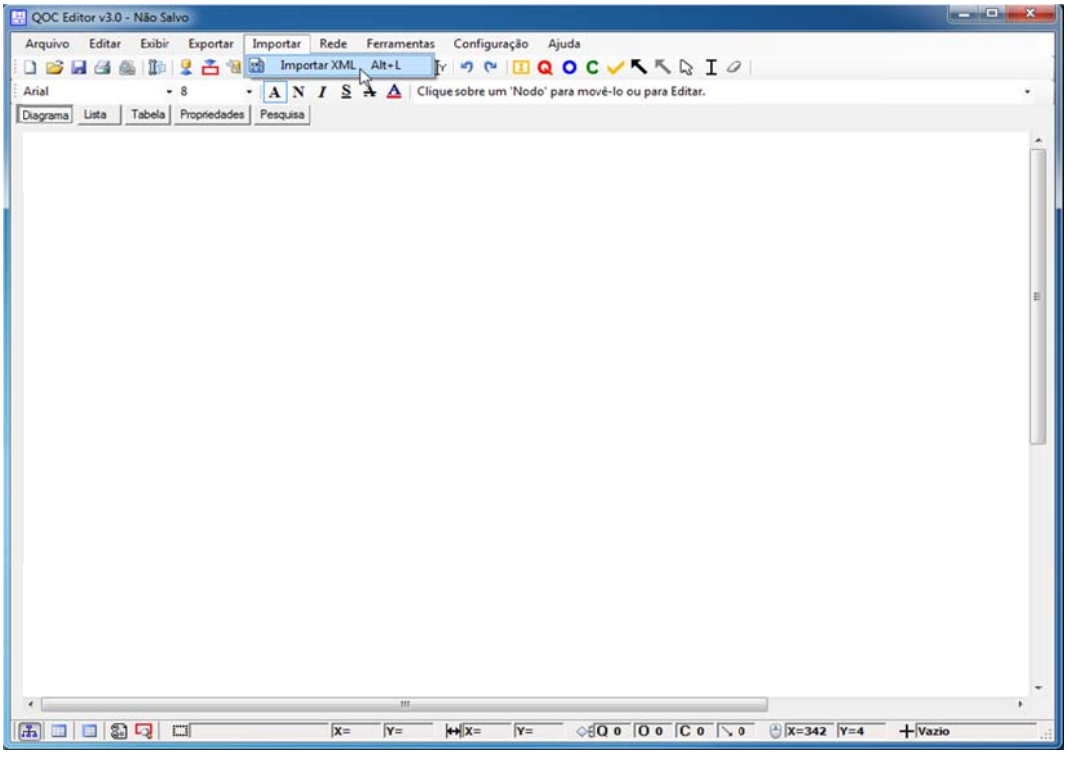

Figura 5.50

Então, selecionamos o arquivo XML que contém o Diagrama a ser aberto:

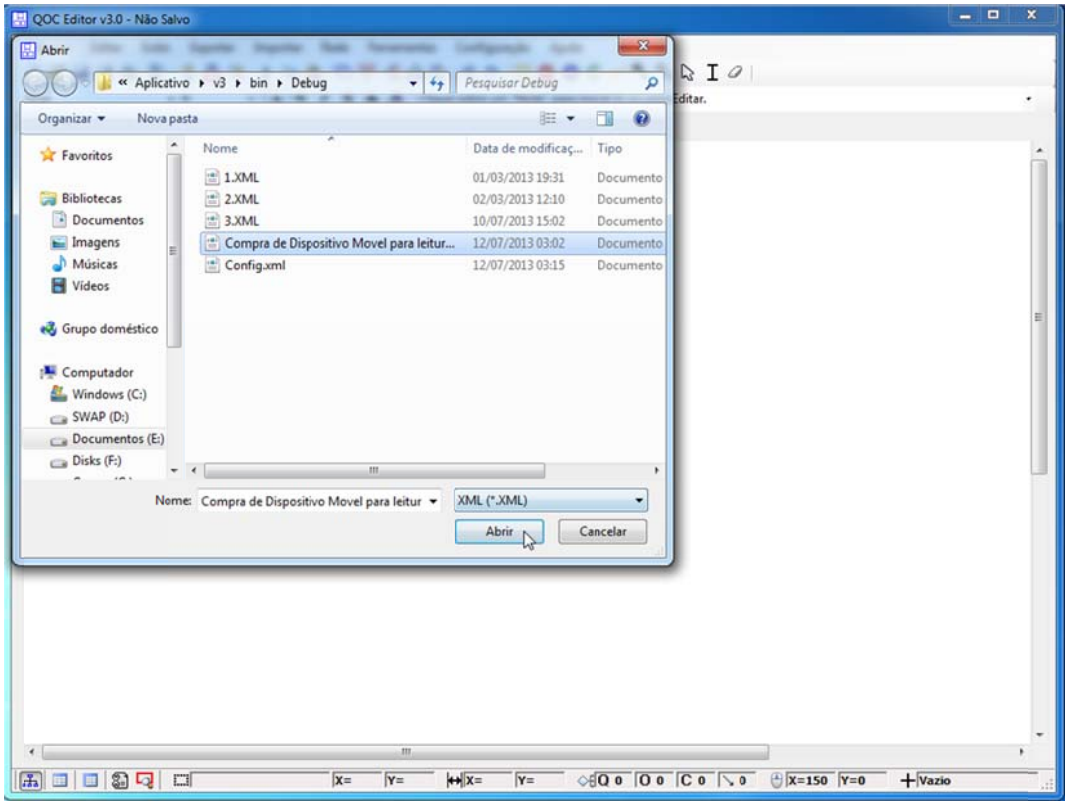

Figura 5.51

Assim, o Diagrama é carregado e exibido:

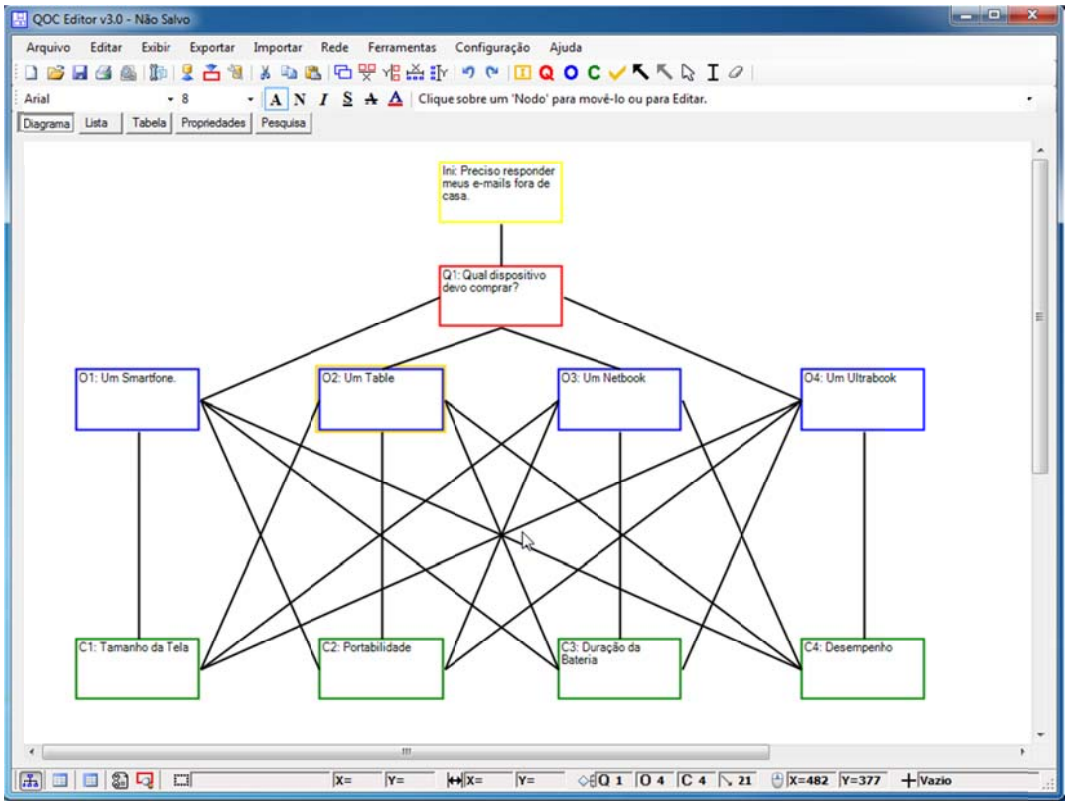

Figura 5.52

Para imprimir, primeiro temos que configurar a impressão. Para isso, acessamos o Menu Arquivo e o item Configurar Impressão:

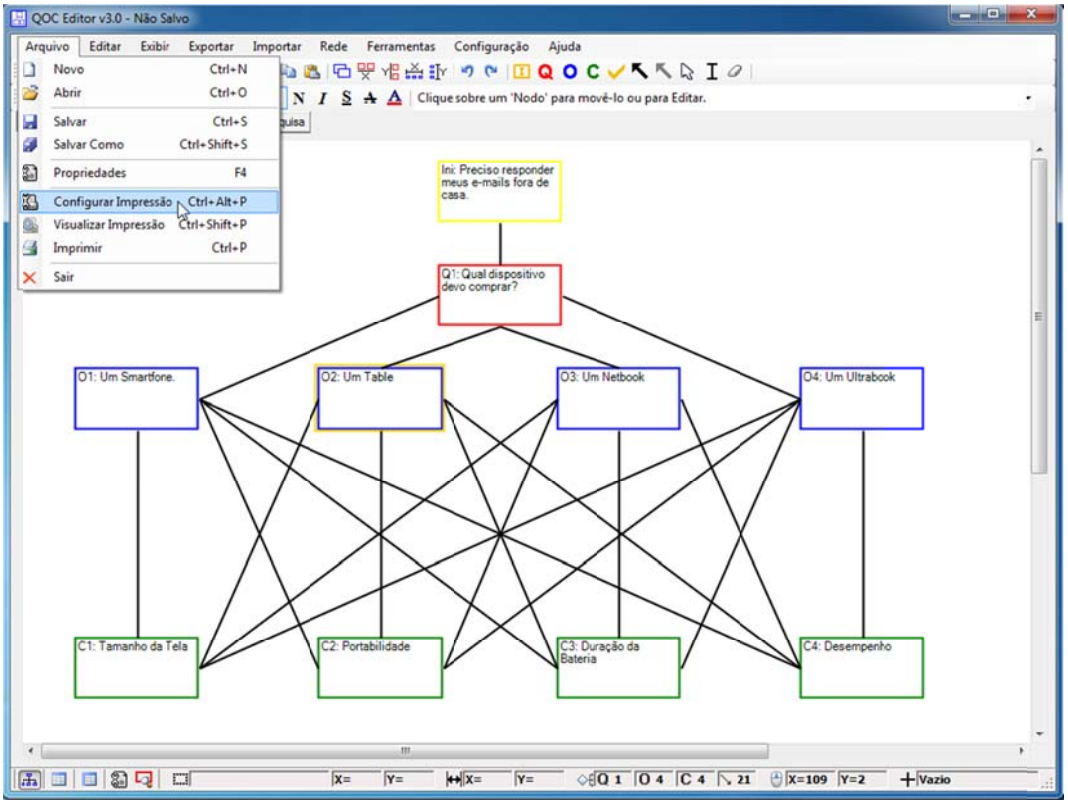

Figura 5.53

Então, o aplicativo exibe a interface de configuração de impressão, onde é possível escolher a Impressora, o Tamanho do Papel, a Orientação, as margens, e ajustar o Diagrama ao Tamanho do Papel.

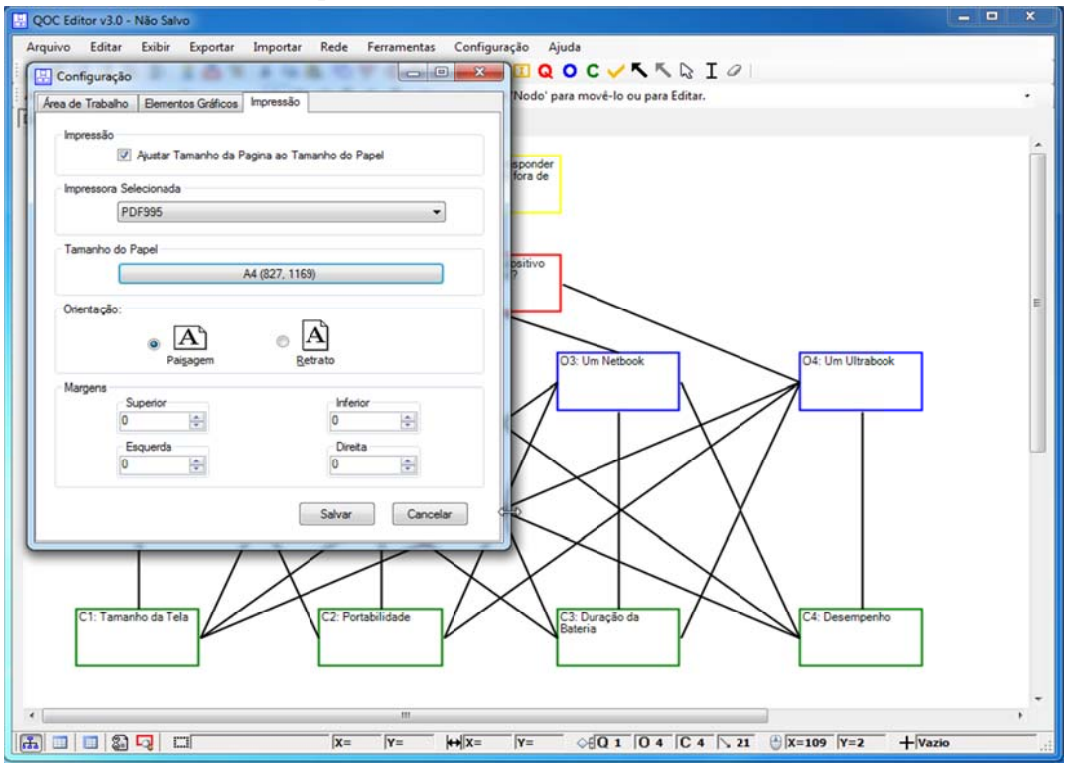

Figura 5.54

Após, podemos visualizar a impressão acessando o menu Arquivo item Visualizar Impressão:

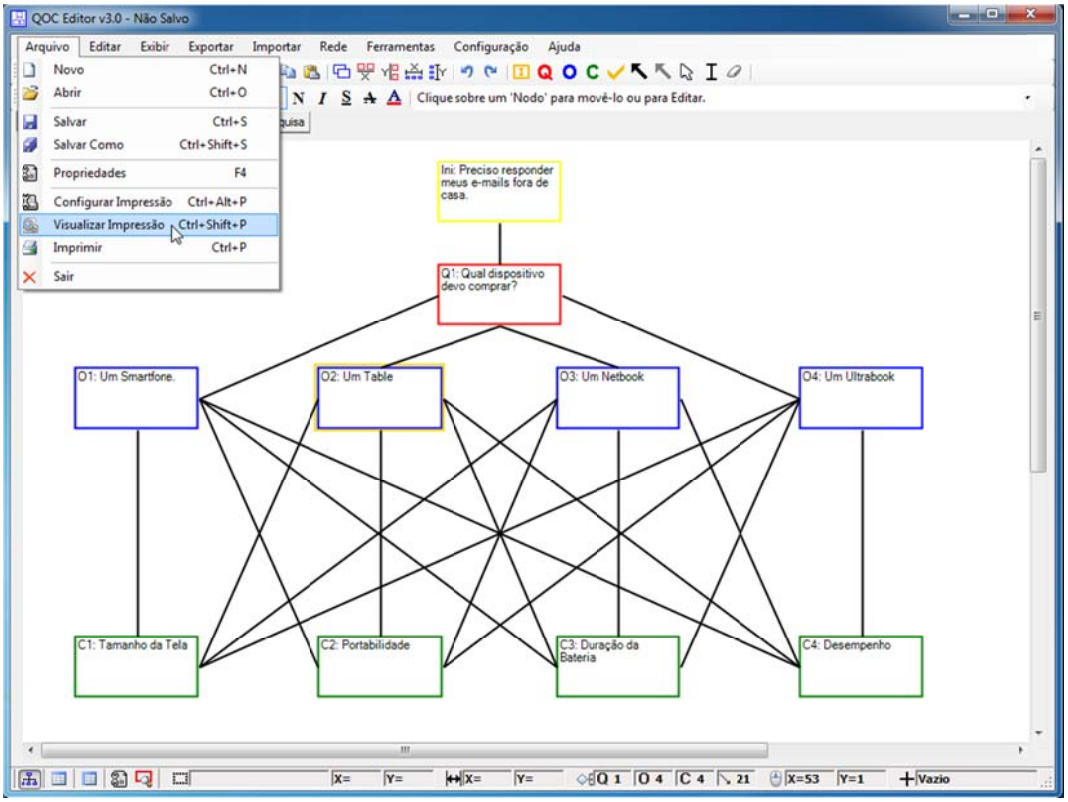

Figura 5.55

Então, o aplicativo exibe a interface de Visualização de Impressão. Se a impressão estiver como desejado, basta clicar no botão imprimir para imprimir:

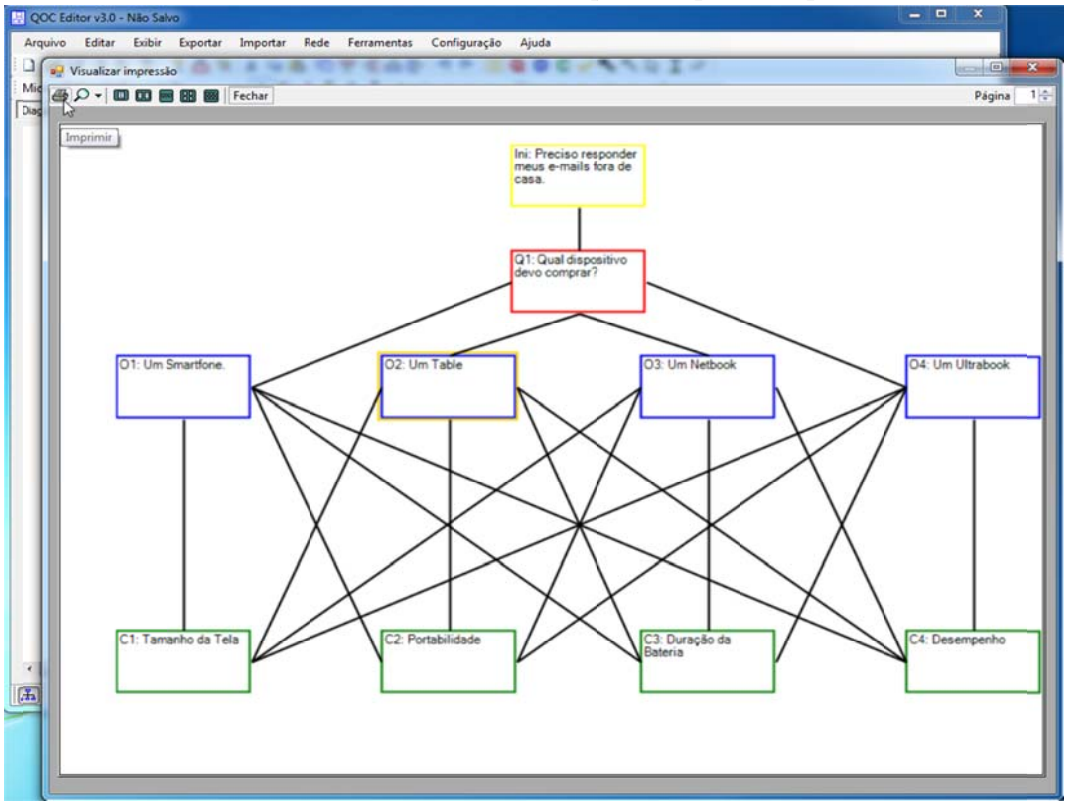

Figura 5.56

Também é possível imprimir acessando o menu Arquivo item Imprimir:

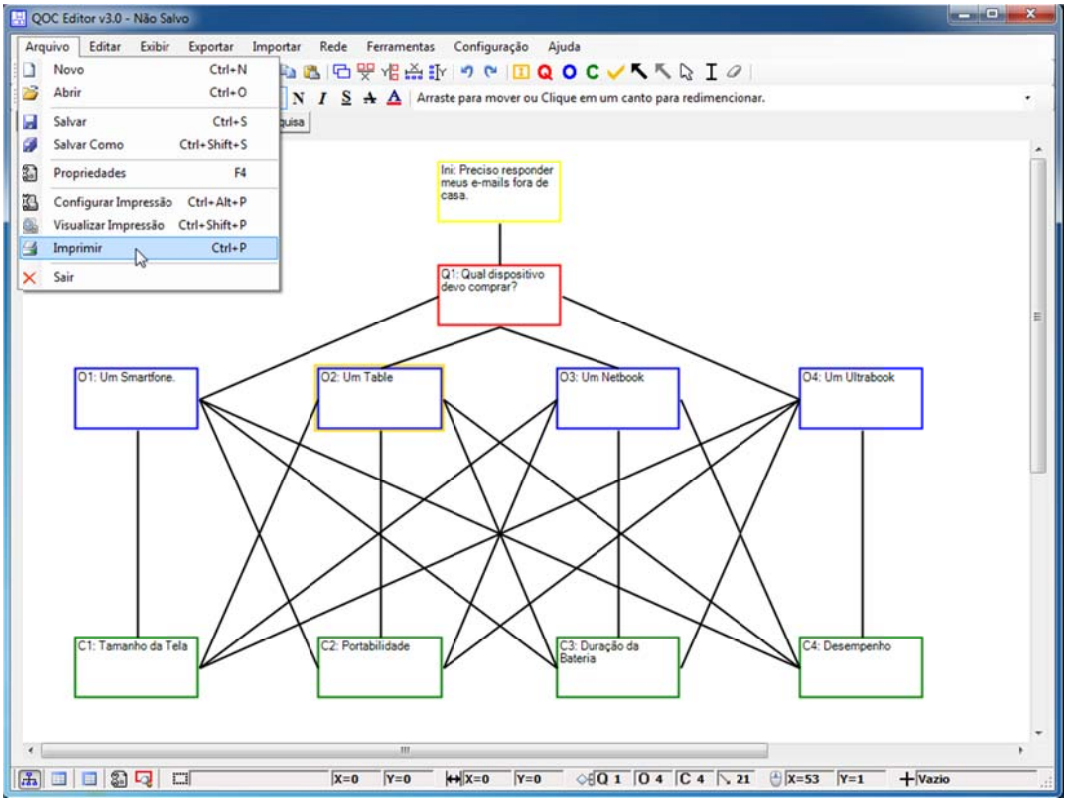

Figura 5.57

A tela abaixo exibe um arquivo PDF impresso pelo procedimento acima:

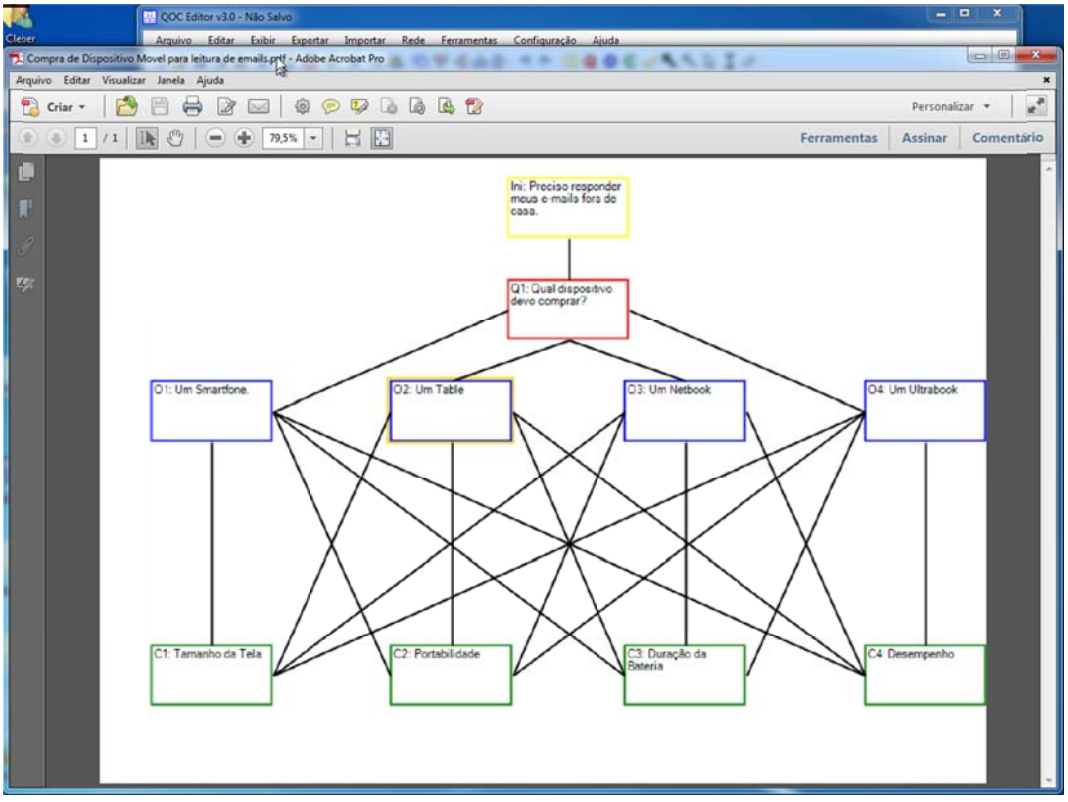

Figura 5.58

5.5 Cenário 5 : Abrir um diagrama QOC da rede e exporte pata HTML

Iniciado o aplicativo, ele exibe sua interface:

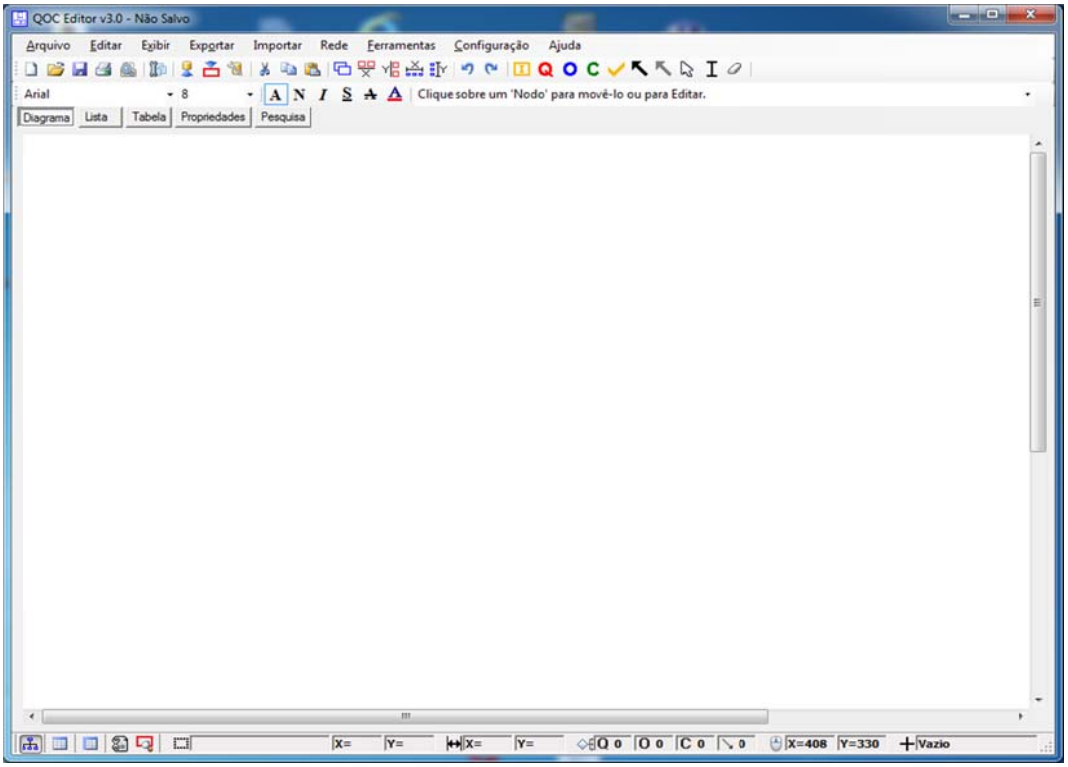

Figura 5.59

Para abrir um Diagrama de um banco de dados da rede, acessamos o Menu Rede e o Item Abrir da Rede:

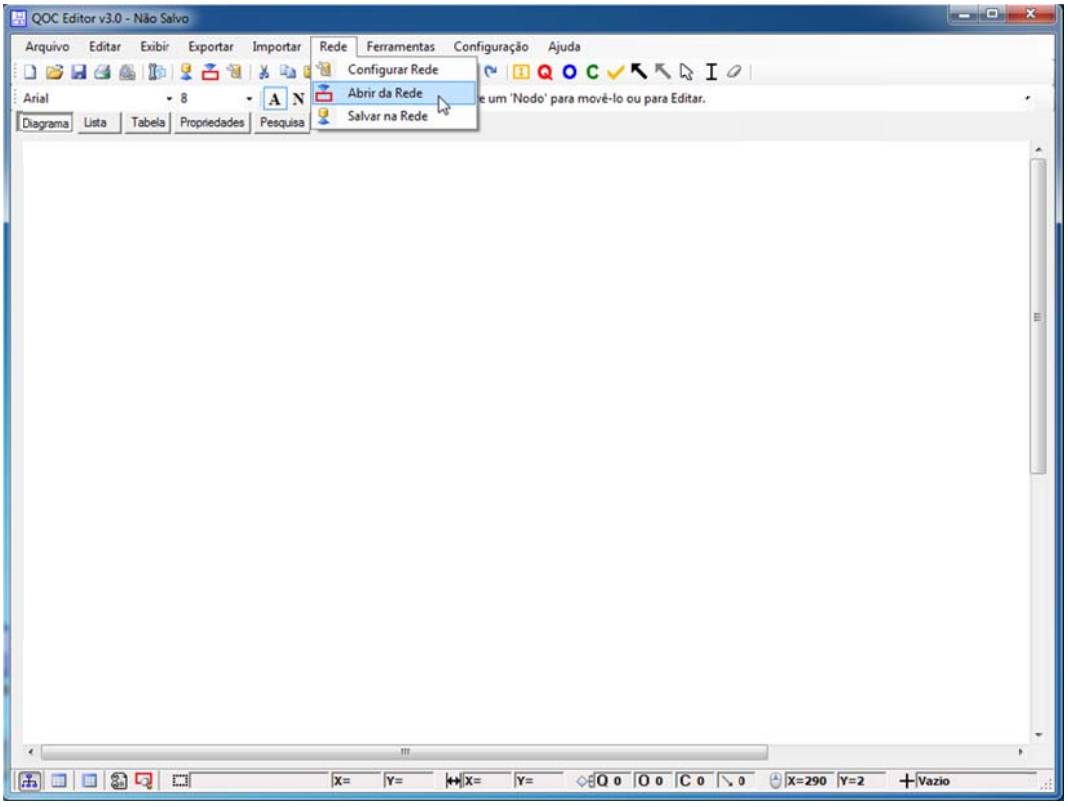

Figura 5.60

Após, digitamos host, porta, nome do banco de dados, usuário e senha, clicamos em conectar para conectar ao servidor:

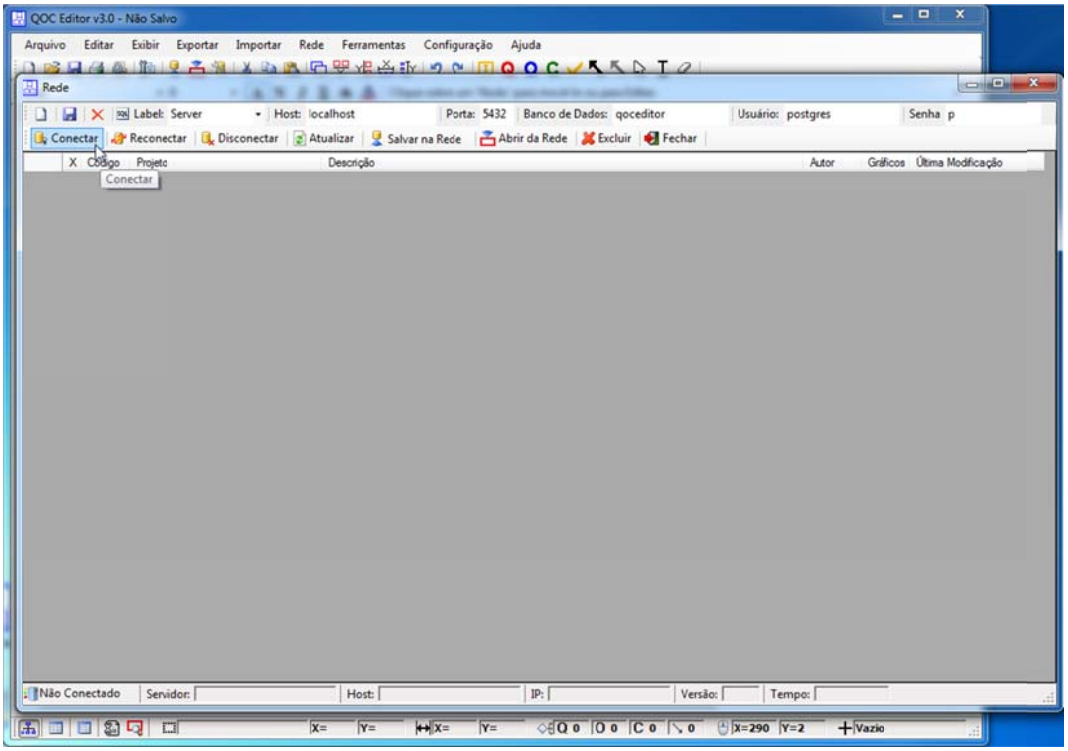

Figura 5.61

Após, o aplicativo conecta ao servidor e exibe a lista de diagramas salvos no banco de dados:

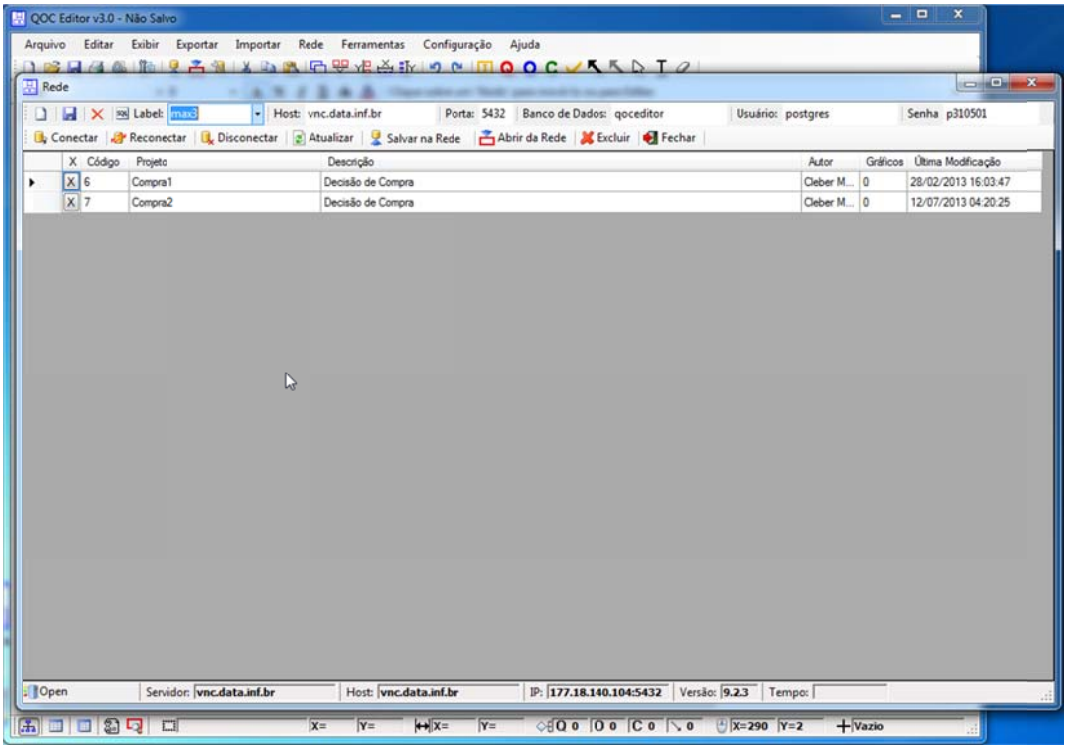

Figura 5.62

Então, selecionamos o Diagrama a ser aberto e clicamos no botão em Abrir da Rede para carregá-los:

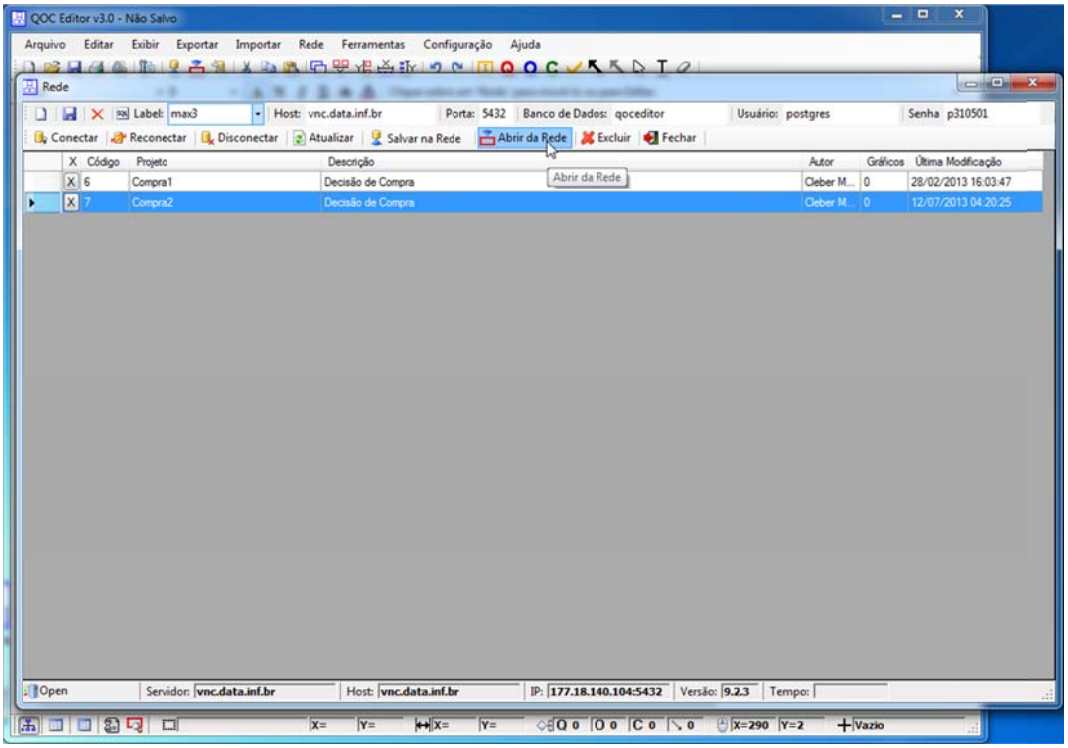

Figura 5.63

Assim, o Diagrama é carregado e exibido:

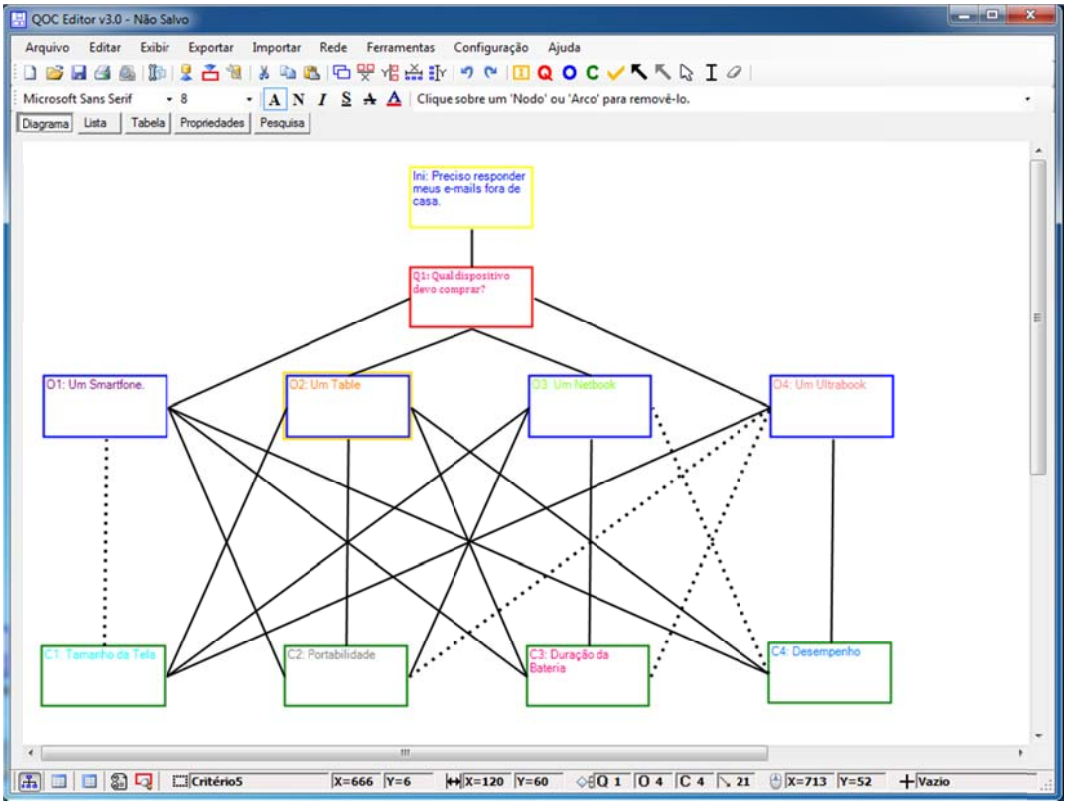

Figura 5.64

Para exportar o Diagrama para HTML, acessamos o Menu Exportar item Exportar HTML:

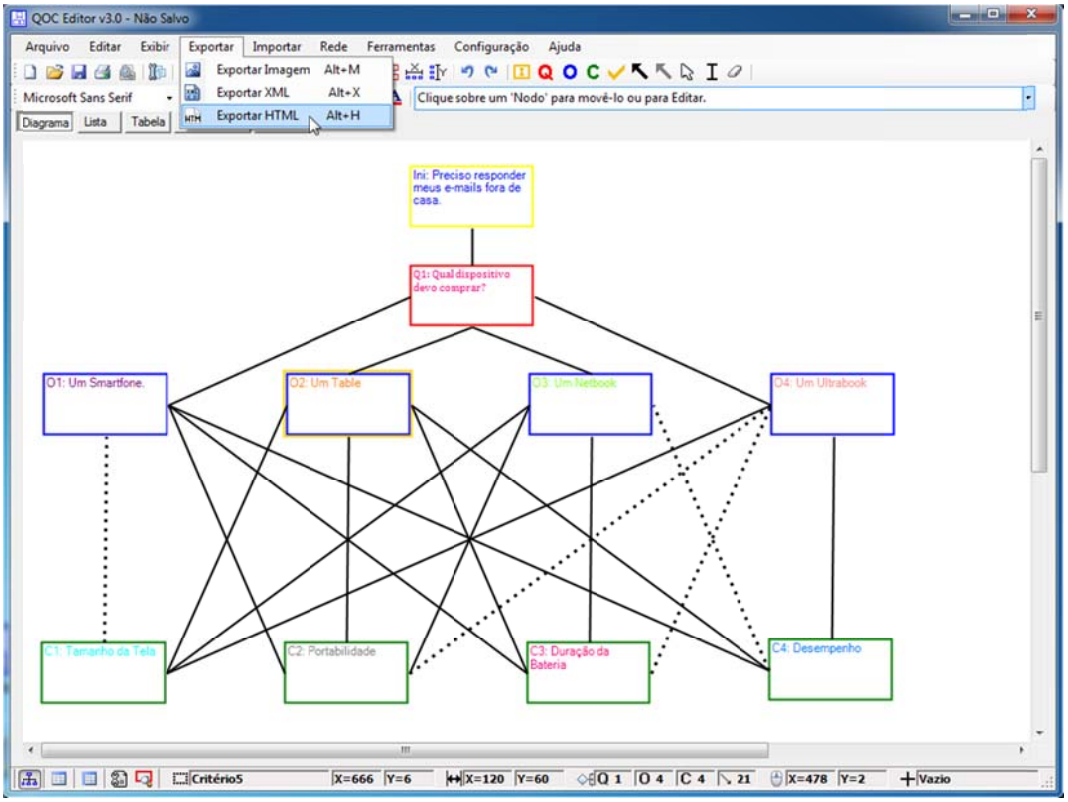

Figura 5.65

Então, digitamos o nome do arquivo a ser criado, e clicamos no botão salvar:

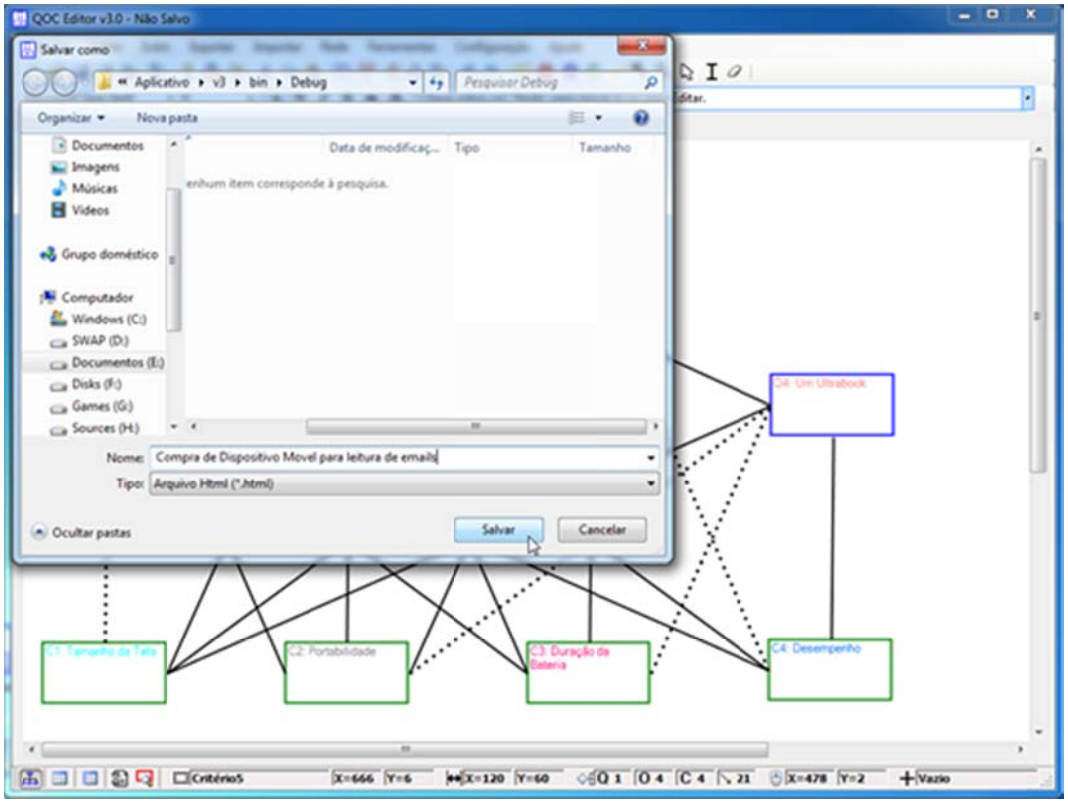

Figura 5.66

Após, o programa gera o arquivo HTML e salva o arquivo no disco:

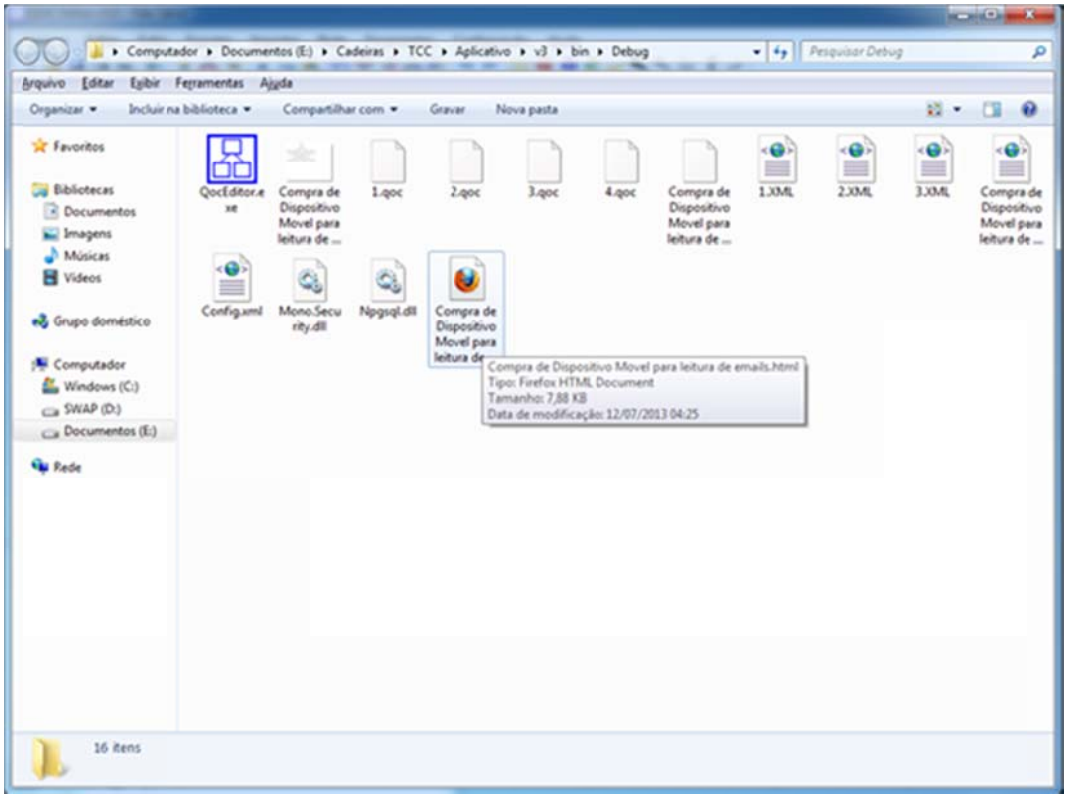

Figura 5.67

O arquivo HTML pode ser aberto por qualquer web browser ou visualizador de HTML:

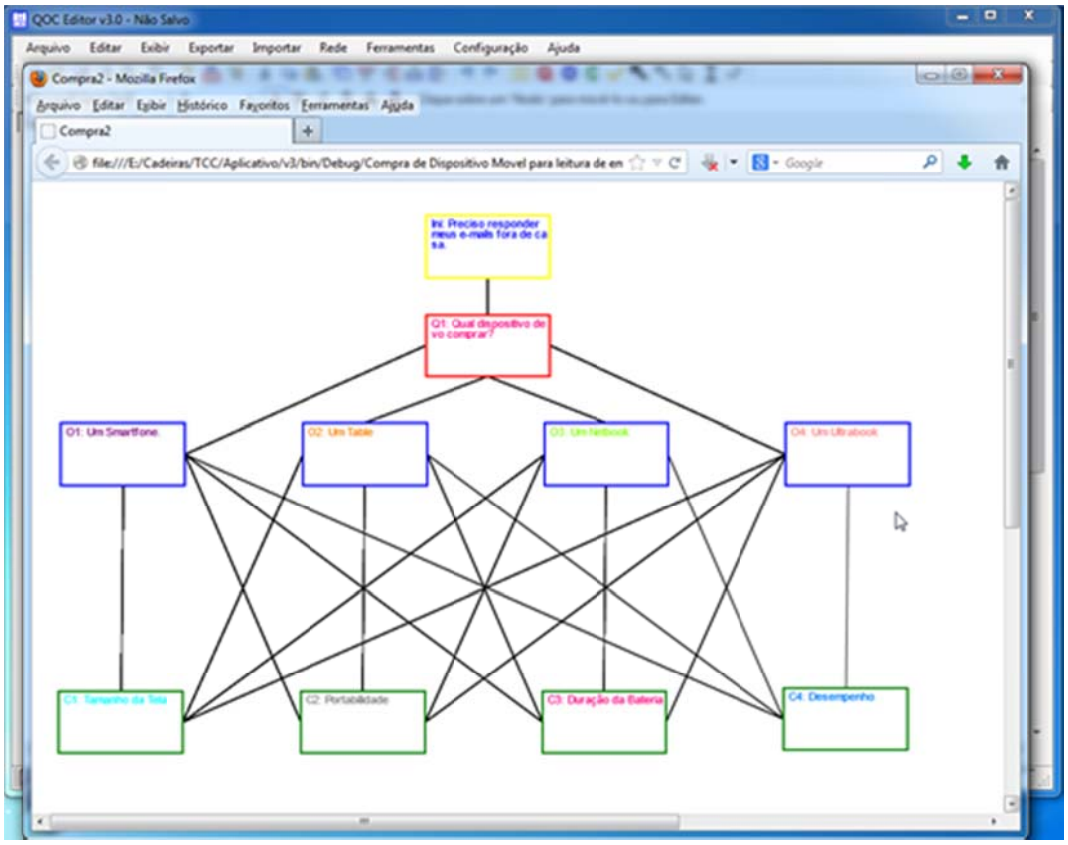

Figura 5.68

6.0 CONCLUSÕES

O QOC Editor é um gerador de diagramas QOC, que permite aos usuários gerenciar o conhecimento relacionado ao processo de tomada de decisão em projetos de software. Através dessa ferramenta, podemos representar, de forma simples e intuitiva, os problemas e alternativas ponderadas antes de cada decisão, reduzir a perda de capital intelectual quando pessoas saem do projeto, propiciar a outros projetos o reuso de experiências positivas e prevenir a repetição de problemas já vivenciados anteriormente, entre outras vantagens.

Neste trabalho foram resumidos os principais conceitos de DR, assim como resumidos os principais modelos de DR, dentre os quais o QOC. Em seguida, foram apresentadas as características do QOC Editor e descritos aspectos de sua modelagem e implementação.

O Design Rationale tem grande potencial para ser uma tecnologia que agregue valor ao processo de desenvolvimento de software. No entanto, Design Rationale ainda não é utilizado adequadamente por empresas em casos reais e, raramente, há casos de captura de informações, fornecendo pouca oportunidade para investigar o problema da captura do Design Rationale na prática.

Espera-se que com esta ferramenta e com o suporte que ela oferece, mais desenvolvedores adotem a ideia de modelar DR.

O QOC Editor, devido a seus pré-requisitos de usabilidade, é uma aplicação complexa. Nesse trabalho, tivemos que lidar com a tarefa de projetar e programar regras de negócio e estruturas de dados complexas na tecnologia escolhida. Nossa abordagem para tratar tal complexidade foi antecipar necessidades futuras e utilizar técnicas comprovadamente funcionais conhecidas como design Patterns. Os design patterns utilizados são padrões que transcendem a aplicação e que facilitaram a incorporação da experiência de desenvolvimentos anteriores em novos projetos.

O Modelo em três camadas escolhido separa as classes do sistema em camadas lógicas: camada de apresentação, camada de negócio e camada de dados. Isso tornou o sistema mais flexível permitindo que as partes possam ser alteradas de forma independente. As funcionalidades da camada de negócio podem ser divididas em classes e essas classes podem ser agrupadas em pacotes ou componentes, reduzindo as dependências entre as classes e pacotes; podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes.

Essas características de modularidade e a escalabilidade facilitarão futuras manutenções e alterações que se façam necessárias e, também, permitirão expansões

do sistema com relativo pouco esforço. Poderíamos implementar uma versão das aplicações que suporte outros modelos de DR como IBIS, PB e DRL, modicando código de poucas classes. Por se tratar de uma área da computação que é objeto de muitas pesquisas, certamente alterações serão necessárias.

REFERÊNCIAS BIBLIOGRÁFICAS

[AMBLER, 1988] AMBLER, Scott W., Análise e Projeto Orientado a Objeto. Volume 2, IBPI Press, Livraria e Editora Infobook AS, 1988.

[BURGE, 1998] BURGE, J. E.; BROWN, D. C. Design Rationale Types and Tools. 1998. Disponível em: http://web.cs.wpi.edu/Research/aidg/DR-Rpt98.html Acesso em: 17/05/2013.

[BURGE, 2000] BURGE, J. E.; BROWN, D. C. Reasoning with Design Rationale. In Proceedings of the Artificial Intelligence Design Conference, 2000.

[BURGE, 2005] BURGE, J. E. Software Engineering Using design RATionale, PhD Dissertation, CS Dept., WPI, May 2005. Disponível em: http://www.wpi.edu/Pubs/ETD/Available/etd-050205-085625/unrestricted/BurgeDissertation.pdf. Acesso em: 17/05/2013.

[CONKLIN, 2006] CONKLIN, J. The IBIS Manual: A Short Course in IBIS Methodology. Disponível em: http://www.touchstone.com/tr/wp/IBIS.html. Acesso em: mar. 2006.

[CONKLIN, 1988] CONKLIN, J.; BEGEMAN, M. L. gIBIS: A hypertext tool for exploratory policy discussion. ACM Transactions on Office Information Systems, 1988;

[CONKLIN, 1989] CONKLIN, J. Interissue dependencies in gIBIS. Technical Report STP-091-89, Microelectronics and Computer Technology Corporation, 1989

[CONKLIN, 1989] CONKLIN, J.; BEGEMAN, M. L. gIBIS: A tool for all reasons, Journal of the American Society for Information Science, 1989.

[CONKLIN, 1989] CONKLIN, J. Design rationale and maintainability. Proceedings 22nd Hawaii International Conference on System Science, 555-561. IEEE Computer Society Press, 1989.

[CONKLIN, 1991] CONKLIN, J.; YAKEMOVIC, K. C. B. A process-oriented approach to Design Rationale. Human-Computer Interaction, 6 (3&4), 357-391, 1991.

[CONKLIN, 1996] CONKLIN, J. Designing organizational memory: Preserving intellectual assets in a knowledge economy. Corporate Memory Systems, 1996.

[CONKLIN, 2009] CONKLIN, J. Hypertext: An introduction and survey. IEEE Computer, 2009.

[FRAMEWORK, 2013] Microsoft, .NET Framework; http://msdn.microsoft.com/pt-br/library/8bs2ecf4%28v=vs.90%29.aspx Acesso em: 24 de junho de 2013.

[GAMMA, 1994] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patters: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.

[GARCIA, 1993] GARCIA, A.; HOWARD, H.; STEFIK, M. Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design, Tech. Report 82, Stanford Univ. Center for Integrated Facility Engineering, Stanford, Calif., 1993.

[GARCIA, 1994] GARCIA, A.; HOWARD, H.; STEFIK, M. Design Rationale for Collaboration: The Active Document Approach, Submitted to Research in Engineering Design Journal, Mar. 1994.

[GRUBER, 1991] GRUBER, T. R.; RUSSEL, D. M. Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use, Technical Report KSL 90-45, Knowledge Systems Laboratory, Standford, California, 40p, 1991.

[HU, 2000] HU, X., PANG, J., PANG, Y., ATWOOD, M., AND SUN, W. A survey on design rationale: representation, capture and retrieval. Engineering with Computers: An Int'l Journal for Simulation-Based Engineering, 2000.

[KUNZ, 1970] Kunz, W.; Rittel, H., Issues as elements of information systems. Working Paper 131, Center for Urban and Regional Development, University of California Berkley, 1970.

[LAKIN, 1989] LAKIN, F. et al. The electronic design notebook: performing medium and processing medium, 1989.

[LACAZE, 2006] LACAZE X., PALANQUE P., BARBONI E., BASTIDE R., NAVARRE D.. From DREAM to Realitiy: Specificities of Interactive Systems Development with respect to Rationale Management, LIIHS-IRIT, University Paul Sabatier, 118 route de Narbonne, 31062 Toulouse Cedex, 2006.

[LACAZE, 2007] LACAZE X., PALANQUE P., DREAM & TEAM: A Tool and a Notation Supporting Exploration of Options and Traceability of Choices for Safety Critical Interactive Systems, INTERACT'07 Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction - Volume Part II Pages 525-540, 2007

[LACAZE, 2010] LACAZE X., BARBONI E., MARTINIE C., Design Rationale Environment for Argumentation and Modelling - DREAM, 2010, Disponível em: http://www.irit.fr/recherches/ICS/softwares/dream/index.html. Acesso em 24 de junho de 2013.

[LEE, 1990a] LEE, J. SIBYL: A qualitative design management system. In P.H. Winston and S. Shellard, eds., Artificial Intelligence at MIT: Expanding Frontiers, Cambridge MA: MIT Press, pp. 104-133, 1990.

[LEE, 1990b] LEE, J. SIBYL: A Tool for Managing Group Decision Rationale. In Proceedings of the CSCW'90 Conference, ACM Press, New York, 79-92 , 1990.

[LEE, 1997] LEE, J. Design rationale Systems: Understanding the Issues. IEEE Expert, Vol. 12, no. 3, pp78-85, 1997.

[LEE, 1991] LEE, J.; LAI, K. A Comparative Analysis of Design Rationale Representations, Human-Computer Interaction Special Issue on Design Rationale, 1991, 6 (3-4) pp. 251-280.

[LEE, 1991a] LEE J. Extending the Potts and Bruns model for recording design rationale. In: Proceedings of 13th International Conf. on Software Engineering, Austin, Texas, May 1991. pp 114-125.

[LEE, 1996] LEE, J.; LAI, K. What's in Design Rationale? in Design Rationale - Concepts, Techniques, and Use, T. Moran and J. Carroll, Eds. New Jersey: Lawrence Erlbaum, 1996, pp. 21-51.

[LOURIDAS,2000] LOURIDAS, P.; LOUCOPOULOS, P. A Generic Model for Reflective Design, ACM Transactions on Software Engineering and Methodology (TOSEM), v.9 n.2, p. 199-237, Apr. 2000.

[JARCZYK, 1992] JARCZYK, A., LOFFLER, P., SHIPMAN F., 1992, "Design rationale for software engineering: A survey". In: Proceedings of 25th Annual Hawaii International Conference on System Sciences, January.

[MACLEAN, 1996] MACLEAN, A.; YOUNG, R. M.; BELLOTTI, V.; MORAN, T., Questions, Options, and Criteria: Elements of Design Space Analysis. In: Design Rationale: Concepts, Techniques, and Use, Lawrence Erlbaum associates, Mahwah, NJ, 1996.

[MACORATTI, 2006] MACORATTI. J. C.: Padrões de Projeto : O modelo MVC - Model View Controller, 2007. Disponível em: http://www.macoratti.net/vbn_mvc.htm. Acesso em 24 de junho de 2013.

[MACORATTI, 2007] MACORATTI. J. C.: Trabalhando com aplicações em 3 camadas, 2007. Disponível em: http://www.macoratti.net/vb_3cm.htm. Acesso em: 17/05/2013.

[MARTINIE, 2010] MARTINIE C., WINCKLER M., PALANQUE P., CONVERSY S,. DREAMER: a Design Rationale Environment for Argumentation, Modeling and Engineering Requirements. In 28th ACM International Conference on Design of Communication (ACM SIGDOC'10) will be held in São Carlos-São Paulo, Brazil on September 26-29, 2010, at BROA Golf Resort. ACM Press. pp. 73-80. ISBN: 978-1-4503-0403-0.

[MSDN, 2013] Microsoft, Assinaturas MSDN no Brasil http://msdn.microsoft.com/pt-br/subscriptions/buy.aspx. Acesso em 24 de junho de 2013.

[MORAN, 1996] Moran, T. P., & Carroll, J. M. (Eds.) (1996). Design rationale: Concepts, techniques, and use. Mahwah, NJ: Lawrence Erlbaum Associates.

[MOUTA, 2010] MOUTA, A. E. B. Dominando OpenSwing; 1ª. Edição, Rio de Janeiro, Editora Ciência Moderna Ltda, 2010.

[MONK, 1995] MONK, S.; SOMMERVILLE, I.; PENDARIES, J. M.; DURIN, B. Supporting design rationale for system evoluation. In Proceedings of the Fifth European Software Engineering Conference, 1995.

[NGUYEN, 1998] NGUYEN, L.; SWATMAN, P. A.; SHANKS, G. Supplementing Process-Oriented with Structure-Oriented Design Explanation within Formal Object-oriented Method. Software Engineering Conference, 1998. Proceedings. 1998 Australian, no.pp.118-132, 9-13 Nov 1998.

[NGUYEN, 2000] Nguyen, L. and Swatman, P.A. Complementary Use of ad hoc and post hoc Design Rationale for Creating and Organising Process Knowledge. Proceedings of the Hawaii International Conference on System Sciences HICSS-33, January 4-7, Maui, Hawaii, USA, 2000.

[PFLEEGER, 2004] PFLEEGER, S. L., Engenharia de Software, Teoria e Prática. Pearson Brasil, 2004.

[PRESSMAN, 2011] PRESSMAN, R. S. Engenharia de Software: uma abordagem profissional, 7ª Edição, McGraw-Hill-Bookman, Porto Alegre, 2011.

[PALANQUE, 1999] Palanque P., Christelle F., Exploitation des notations de Design Rationale pour une conception justifiée des applications interactives. In 11th French-speaking conference on human computer-interaction, IHM'99; November 22-26, 1999. Montpellier, France. pp. 33-40.

[PETERS, 2001] PETERS, J.F., PEDRYCZ, W. Engenharia de software: teoria e prática, Editora Campus, Rio de Janeiro, 2001.

[ROMAN, 1985] ROMAN, G.-C. A taxonomy of current issues in requirements engineering, Computer, volume: 18 Issue: 4 pp 14 - 23, ISSN: 0018-9162, 1985.

[SOMMERVILLE, 2011] SOMMERVILLE, I. Engenharia de software, 9ª Edição, Ed. Pearson Prentice Hall, São Paulo, 2011.

[THAYER, 1990] THAYER, Richard H. Thayer, Merlin Dorfman; System and software requirements engineering, 1990

[VINCENT, 2012] VINCENT C., BLANDFORD A., LI Y. QOC-E: A Mediating Representation To Support The Development Of Shared Rationale and Integration Of Human Factors Advice, Symposium on Human Factors and Ergonomics in Health Care, 2012.

[VISUAL C#, 2013] Microsoft, Visual C# Developer Center http://msdn.microsoft.com/pt-br/vcsharp. Acesso em 24 de junho de 2013.

[VISUAL STUDIO, 2013] Microsoft, Visual Studio no MSDN. Disponível em: http://msdn.microsoft.com/pt-br/vstudio. Acesso em 24 de junho de 2013.