Análise e comparação de frameworks para edição e visualização de grafos

Lucas Fernando Souza de Castro[1], Gleifer Vaz Alves [1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR) – Ponta Grossa – PR – Brasil

l.castropg@gmail.com, gleifer@utfpr.edu.br

Abstract. *This work is related to a project, which has as main idea the construction of computational tools for proof-graphs. Thus, the chief goal of our work is the analysis and comparison of frameworks based on Java for editing and visualizing graphs. With this in mind, we have build a graph editor with a two-fold goal: comparing the Java frameworks and use it as a prototype for our proof-graph tool.*

Keywords: *Graph, Framework, Proof-Graph, Java*

Resumo. *Este trabalho está inserido dentro de um projeto que tem por objetivo a construção de ferramentas computacionais para grafos-de-prova. Assim, a meta do presente trabalho é analisar e comparar frameworks baseados para a linguagem Java para edição e visualização de grafos. Para isso, foi construído um editor de grafos com dois objetivos: comparar os frameworks Java e servir de protótipo para a ferramenta edição grafos-de-prova.*

Palavras-chave: *Grafo, Framework, Grafo-de-Prova, Java*

1. Introdução

Em Ciência da Computação ou em demais áreas co-relacionadas, grafos podem ser definidos como uma forma de representação de dados [Harary 1969]. Sua aplicabilidade abrange inúmeras áreas, como: Química, Engenharia, Matemática, e principalmente a área da Computação, onde existem *n* sub-áreas as quais grafos aplicam-se, como no caso da Inteligência Artificial, Redes, Jogos, Compiladores, Lógica, entre outras sub-áreas relacionadas.

Devido o uso de grafos tornar-se comum em várias áreas, extensões e novas formas de grafos surgiram. Entre as diferentes aplicações de grafos, surge o uso na área de teoria da prova. Especificamente para representação de provas da Dedução Natural. Em sua tese, [de Oliveira 2001], faz uso de conceitos da teoria dos grafos para criação de um sistema de provas chamado N-Grafos, onde se utiliza a noção de grafo direcionado para representar os grafos-de-prova, que de uma maneira geral são grafos rotulados com fórmulas lógicas. Os N-Grafos caracterizam-se como um sistema que mescla regras da Dedução Natural e do Cálculo de Sequentes, além disso fazem uso da estrutura de prova com múltiplas conclusões, o que viabiliza o uso em aplicações computacionais concorrentes. Em [Alves 2009], o autor define a propriedade de normalização para os N-Grafos. A partir desses resultados almeja-se criar ferramentas capazes de editar, manipular, visualizar e verificar propriedades dos N-Grafos. Um dos primeiros resultados em direção a esse objetivo é apresentado em [dos Santos 2012], onde um *schema* XML é definido para representar computacionalmente os N-Grafos.

Dando continuidade a obtenção de resultados para atingir os objetivos citados anteriormente. Aqui demonstra-se uma análise de dois *frameworks* destinados à visualização e manipulação em grafos: JUNG e JGraph. A análise tem como meta comparar ambos os *frameworks* e eleger um que atenda aos requisitos necessários para a elaboração de um editor para N-Grafos. Como metodologia a esta análise, foi desenvolvido na linguagem Java um editor *desktop* para grafos, usufruindo das funções de visualização da JUNG e JGraph. A escolha da linguagem Java é derivada do grande uso desta linguagem no meio acadêmico e profissional. A escolha pela JUNG e JGraph é resultante de dois motivos: a JUNG foi utilizada em [Kaspczak and Rodrigues 2011] como sendo *framework* para a visualização dos grafos. Já a JGraph é citada em diversos fóruns de tecnologia como sendo um *framework* amplamente utilizado para a visualização de grafos.

Ambos *frameworks* foram submetidos a análises e tiveram sua avaliação baseada nos resultados demonstrados através do desempenho e do suporte aos requisitos no editor desenvolvido. A partir da análise de desempenho, foi utilizado a escala Likert para a visualização e interpretação dos resultados.

O restante deste artigo subdivide-se na seguinte forma: a Seção 2 apresenta a descrição dos *frameworks*, a Seção 3 demonstra a análise e comparação e os resultados obtidos, e por fim, a Seção 4 as considerações finais.

2. Frameworks de visualização e manipulação em grafos

Nesta seção é apresentada uma breve descrição de dois *framework*s destinados a visualização e manipulação de grafos, JUNG e JGraph. Inicialmente é narrado seu histórico e após, uma introdução ao funcionamento da visualização do grafo em ambos *framework*s.

2.1. JUNG - Java Universal Network Graph

JUNG é um *framework* específico para a manipulação, análise e visualização de grafos para a linguagem Java. Joshua O'Madadhain, Danyel Fisher, e Scott White, são seus idealizadores e desenvolvedores. JUNG, *Java Universal Network Graph*, teve sua primeira versão lançada em agosto de 2003.

A arquitetura da JUNG é projetada para suportar uma variedade de representações de entidades e suas relações, tais como grafos direcionados e não direcionados, grafos multi-modais e hipergrafos. A JUNG fornece recursos adicionais como: algoritmos de teoria dos grafos, mineração de dados e análise de redes sociais, tais como rotinas de agrupamento e geração de gráficos aleatórios [Jung 2009].

O *framework* fornece ao desenvolvedor bibliotecas específicas para a implementação de algoritmos aplicados a grafos, como busca do menor caminho e verificação de conectividade.

Para a visualização e renderização do grafo são utilizadas as bibliotecas *Swing* e *Awt*, contidas no pacote *javax.swing* e *java.awt*, respectivamente. A biblioteca *Swing* é responsável pelos criação componentes gráficos do Java, porém é independente de plataforma. Já a *Awt* é utilizada para criação de interfaces com o usuário e para a pintura de gráficos e imagens, porém delegando estas funções ao sistema operacional. Na figura 1 é apresentado um grafo direcionado utilizando o *framework* JUNG para a visualização.

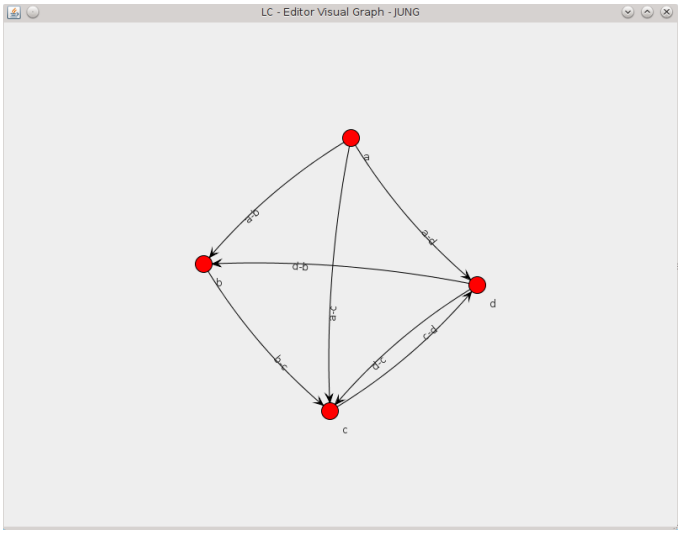

Figura 1. Renderização do grafo utilizando JUNG

A disposição dos vértices e arestas na JUNG, é gerada de forma automática através do uso de *layouts* pré-definidos. A figura 1 demonstra o uso do layout: KKLayout, juntamente com ilustração da renderização do grafo realizada dentro do editor para grafos desenvolvido, o qual é demonstrado na seção 3.2. Além do KKLayout, há outros *layouts* fornecidos pelo *framework* como: IsoMLayout, CircleLayout, FrLayout e SpringLayout. Segundo, [Kaspczak and Rodrigues 2011] e [JUNG 2009] estes são alguns *layouts* fornecidos pelo *framework*.

- KKLayout: "Algoritmo: Kamada-Kawai" - Usa um *layout* de nó, não respeita as chamadas de filtro.
- ISOMLayout: "Mapa de Auto-Organização Meyer" - Implementa um algoritmo que realiza uma auto-organização de *layout.*
- CircleLayout: " *Layout* simples de posições randômicas de vértices em um círculo" - Possui uma implementação onde os vértices ficam igualmente espaçados em um círculo regular.
- FRLayout: "Algoritmo: Fruchterman-Reingold – Os nós ficam dispostos de acordo com um algoritmo que possui como conceito base que nos que situam-se próximos uns dos outros acabam por repelir-se.
- SpringLayout: Possui nós filhos associados com determinados tipo de regras.

O uso de *layouts* pré-definidos proporciona uma maior flexibilidade na exibição do grafo, pois, com uso destes, não é necessário desenvolver algoritmos de visualização.

A partir de sua criação no ano de 2003 até agosto de 2012 foram lançadas 31 versões, chegando a atualização 2.0.1, lançado em janeiro de 2010. Nota-se que a partir desta data, não foram lançadas atualizações assim a JUNG caracteriza-se por ser um *framework* desatualizado em relação as bibliotecas Java, como por exemplo, a *Swing*. Destaca-se que em 2010, a versão do JDK (*Java Development Kit*) estava na versão 1.5, ao passo que atualmente o JDK está na versão 7.0.

2.2. JGraph

JGraph, é um *framework* destinado a manipulação de grafos e diagramas, porém o seu ponto forte é a manipulação e criação de grafos visuais. Em 2001, foi anunciado sua primeira versão [JGraph 2012]. A JGraph não possui desenvolvedores fixos, e sim uma equipe de desenvolvimento.

O *framework* utiliza as bibliotecas *javax.swing e java.awt* para a renderização dos elementos gráficos e de interação com o usuário, semelhante a JUNG.

O *framework* JGraph é disponível tanto para o desenvolvimento web como para desktop. Para a web, é possível utilizá-lo usando JavaScript, e desktop utilizando Java. Para a dominação web utilizando JavaScript, o *framework* denomina-se mxGraph e para desktop JGraphX.

A figura 2 demonstra a utilização do editor de grafos realizando a renderização do grafo utilizando o *framework* JGraph.

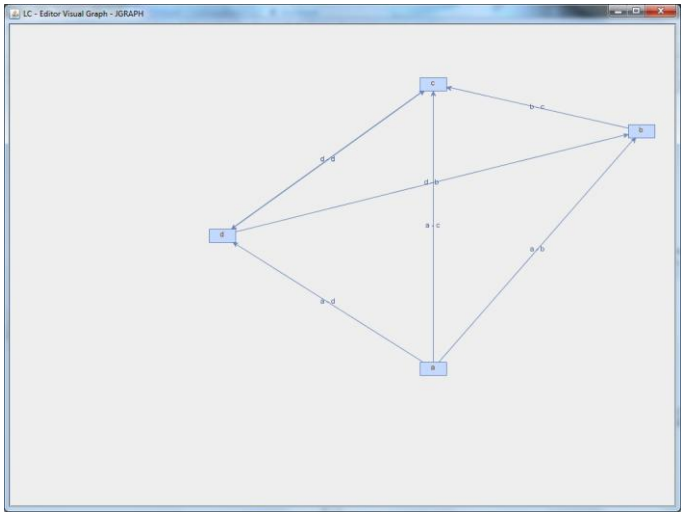

Figura 2. Renderização do grafo utilizando JGraph

A disposição dos vértices e arestas é gerada de forma manual, usando coordenadas cartesianas, X e Y. A JGraph oferece layouts de visualização, porém estes não são mencionados na documentação oficial nem ao menos em seus exemplos de implementação. A "descoberta" que o *framework* possuía suporte ao layouts foi através de uma análise em sua árvore de diretórios, onde foi assimilado a implementação de alguns algoritmos de visualização, *layouts.*

3. Análise e Comparação dos Frameworks

Nesta seção é apresentado o relatório da análise e comparação dos *frameworks* mencionados na seção 2. Na subseção 3.1 é apresentado um levantamento de requisitos, na qual foram definidos critério de avaliação para os *frameworks* de visualização e manipulação em grafos. Na subseção 3.2 é apresentado um editor visual de grafos, o qual foi desenvolvido para comparar o funcionamento de ambos *frameworks*. Na subseção 3.3 é apresentada a análise e comparação referente a cada *framework*.

3.1. Levantamento de Requisitos

Os requisitos levantados aplicam-se ao contexto de desenvolvimento do editor para n-grafos. O futuro editor irá demandar requisitos visuais como requisitos de eventos em tempo de execução, como por exemplo: uso de *listeners* à eventos de mouse e teclado. Os requisitos demonstrados a seguir foram itens de avaliação na qual os *frameworks* apresentados foram submetidos a análise e comparação de seus funcionamentos e suporte.

- **Atualizações:**
  - Lançamentos periódicos de atualizações *(releases)*;
- **Comunidadede desenvolvimento/usuários ativa:**
  - Fóruns de comunicação entre usuários e desenvolvedores;
  - Manuais de implementação e uso;
  - Modelos e exemplos de implementação;
  - Portal de desenvolvedores atualizado;
- **Desempenho;**
  - Desempenho em tempo de renderização do grafo;
- **Eventos de interação com o usuário;**
  - Suporte ao uso de *listeners* para eventos de mouse e teclado;
- **Flexibilidade de alteração na estrutura visual do grafo;**
  - Adição/remoção de estruturas visuais de vértices/arestas;
  - Ajuste de tamanho visual do vértice em tempo de execução;
  - Alteração de origem/destino de arestas;
  - Criação de vértices e arestas com rótulos personalizáveis em tempo de execução;
  - *Layouts* de visualização;
  - Movimentação da estrutura visual do vértice;
- **Fácil Implementação;**
  - Clareza em codificação;
  - Documentação oficial detalhada;
  - Modelos e exemplos de implementação;

Com o objetivo de analisar os *frameworks* e avaliá-los conforme os requisitos acima, foi utilizado a atribuição de pesos, de 0 a 10. Com base nisto, cada item recebeu um valor de relevância de acordo com a sua funcionalidade e importância dentro das delimitações de requisitos. Uma escala psicométrica denominada escala Likert foi utilizada para a atribuição dos pesos de cada item dos requisitos. Esta escala é amplamente utilizada em diversas situações como por exemplo, em pesquisas de opinião [Trochim 2006].

De acordo com grau de importância que cada requisito exerce, foram definidos cinco níveis de relevância, os quais foram aplicados aos requisitos. Cada requisito recebe um peso referente ao grau de importância que exercerá sobre o futuro editor. A Tabela 1 descreve estes níveis, os quais serão representados por siglas.

A Tabela 2 demonstra os pesos aplicados aos requisitos exibidos na subseção 3.1 utilizando os níveis de importância exibidos na Tabela 1.

Com base nos requisitos descritos na Tabela 2, é necessário avaliar os *frameworks* de forma individual, esta análise apresentada adiante, na subseção 3.3. Para realizar a análise, foi utilizado novamente a escala Likert, porém com três níveis, onde cada framework recebe a nota de avaliação nos itens de cada requisito, através de uma porcentagem de acordo com o desempenho e suporte do mesmo. Os três níveis de avaliação são descritos na Tabela 3.

SIGLA	IMPORTÂ NCIA	PESO
EI	Extremamente Importante	10
MI	Muito Importante	7.5
I	Importante	5
PI	Pouco Importante	2.5
SI	Sem Importância	0

Tabela 1. Definição dos pesos

REQUISITO	EI	MI	I	PI	SI	PESO
Lançamentos periódicos versões		x				7.5
Fóruns de comunicação entre usuários e Desenvolvedores		x				7.5
Manuais de implementação e uso	x					10
Modelos e exemplos de implementação	x					10
Portal de desenvolvedores atualizado		x				7.5
Desempenho em tempo de renderização do grafo	x					10
Suporte ao uso de listeners para eventos de mouse e teclado		x				7.5
Adição/remoção de estruturas visuais de vértices/arestas	x					10
Ajuste de tamanho visual do vértice em tempo de execução			X			5
Alteração de origem/destino de arestas	x					10
Criação de vértices e arestas com rótulos personalizáveis em tempo de execuçã;	x					10
Layouts de visualização;		x				7.5
Movimentação da estrutura visual do vértice		x				7.5
Clareza em codificação		x				7.5
Documentação oficial detalhada	x					10

Tabela 2. Requisitos para análise dos *frameworks* com base na escala Likert

SIGLA	RELEVÂ NCIA	PORCENTAGEM NOTA
AS	Atende Satisfatoriamente	100%
AP	Atende Parcialmente	50%
NA	Não Atende	0%

Tabela 3. Níveis de atribuição de notas nas avaliações dos *frameworks*

3.2. Editor Visual para Grafos

Afim de comparar os dois *frameworks* apresentados, JUNG e JGraph, foi desenvolvido uma aplicação em Java SE, onde foi implementado um editor visual para grafos, LC - Editor Visual Graph. O editor por sua vez, além do objetivo de comparação, servirá como um protótipo para o desenvolvimento do futuro editor para os n-grafos.

REQUISITO	FUNCIONALIDADE -	AS	AP	NA	NOTA
Lançamentos periódicos de versões (Peso: 7.5)	Lançamentos praticamente quinzenais.	X			7.5
Fóruns de comunicação entre usuários e desenvolvedores (Peso: 7.5)	Possui, porém há poucos tópicos com assuntos avançados.		x		3.75
Manuais de implementação e uso (Peso: 10)	Possui, porém, apenas com funcionalidades básicas.		x		5
Modelos e exemplos de implementação (Peso: 10)	Possui.	X			10
Portal de desenvolvedores atualizado (Peso: 7.5)	Possui.	X			7.5
Desempenho em tempo de renderização do grafo (Peso: 10)	Excelente tempo de resposta.	X			10
Suporte ao uso de listeners para eventos de mouse e teclado (Peso: 7.5)	Possui.	X			7.5
Adição/remoção de estruturas visuais de vértices/arestas (Peso: 10)	Possui	X			10
Ajuste de tamanho visual do vértice em tempo de execução (Peso: 5)	Possui. Não é necessário implementação adicional para habilitar.	X			5
Alteração de origem/destino de arestas (Peso: 10)	Possui.	X			10
Criação de vértices e arestas com rótulos personalizáveis em tempo de execução (Peso: 10)	Possui. Não é necessário implementação adicional para habilitar.	X			10
Layouts de visualização (Peso: 7.5)	Possui, porém, não são consta na documentação oficial.		x		3.75
Movimentação da estrutura visual do vértice (Peso: 7.5)	Possui.	X			7.5
Clareza em codificação (Peso: 7.5)	Código “legível”; Manipulação direta de objetos	X			7.5
Documentação oficial detalhada (Peso: 10)	Documentação oficial relata apenas funcionalidades básicas.		x		5

Tabela 4. Resultado avaliação - JGraph

Os dois *frameworks* possuem suas próprias estruturas de dados para o armazenamento do grafo em tempo de execução. Diante disso, foi desenvolvido uma estrutura de dados independente de *framework* baseada em uma lista de adjacências, sendo assim, ao grafo ser renderizado em um determinado *framework*, é realizado uma cópia da estrutura de dados do editor para a estrutura do *framework*, e vice-versa.

Toda edição do grafo, adição/remoção de vértices e arestas é realizada dentro da estrutura de dados do editor, pois assim ao grafo ser renderizado em um dos dois *frameworks*, toda a estrutura de dados referente ao *framework* será atualizada.

REQUISITO	FUNCIONALIDADE - JUNG	AS	AP	NA	NOTA
Lançamentos periódicos de versões (Peso: 7.5)	Sem atualizações de janeiro/2010 à agosto/2012.			x	0
Fóruns de comunicação entre usuários e desenvolvedores (Peso: 7.5)	Não possui.			x	0
Manuais de implementação e uso (Peso: 10)	Possui, além de outros manuais não oficiais detalhados.	X			10
Modelos e exemplos de implementação (Peso: 10)	Possui.	X			10
Portal de desenvolvedores atualizado (Peso: 7.5)	Possui, porém sem atualizações.		x		3.75
Desempenho em tempo de renderização do grafo (Peso: 10)	Tempo de resposta alto em relação à JGraph.		x		5
Suporte ao uso de listeners para eventos de mouse e teclado (Peso: 7.5)	Possui, porém não consta na documentação oficial.		x		3.75
Adição/remoção de estruturas visuais de vértices/arestas (Peso: 10)	Possui, porém não consta na documentação oficial.		x		5
Ajuste de tamanho visual do vértice em tempo de execução (Peso: 5)	Possui, porém não consta na documentação oficial.	x			2.5
Alteração de origem/destino de arestas (Peso: 10)	Possui.	x			10
Criação de vértices e arestas com rótulos personalizáveis em tempo de execução (Peso: 10)	Não há nada documentado ou publicado.			x	0
Layouts de visualização (Peso: 7.5)	Possui.	x			7.5
Movimentação da estrutura visual do vértice (Peso: 7.5)	Há suporte, porém não consta na documentação oficial.		x		3.75
Clareza em codificação (Peso: 7.5)	Código “legível”; Manipulação direta de objetos.	x			7.5
Documentação oficial detalhada (Peso: 10)	Documentação oficial relata apenas funcionalidades básicas e intermediárias.		x		5

Tabela 5. Resultado avaliação - JUNG

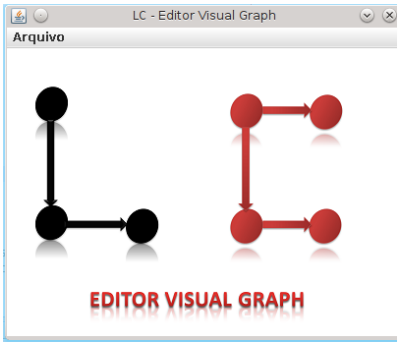

Figura 3. Tela Inicial - LC

3.3. Análise e Comparação: JGraph e JUNG

Esta subseção demonstra a análise realizada nos *frameworks* aqui apresentados JGraph e JUNG utilizando a escala Likert, com base nos requisitos definidos na subseção 3.1. As versões dos *frameworks* analisados foram as mais recentes de acordo com data do trabalho. A versão da JUNG encontra-se na versão 2.0.1 e a JGraph 1.10.2.1.

A Tabela 4 ilustra a análise referente a JGraph, já a Tabela 5 demonstra a análise referente ao *framework* JUNG. O gráfico exibido na figura 4 ilustra os resultados obtidos na análise, com base nas tabelas 4 e 5.

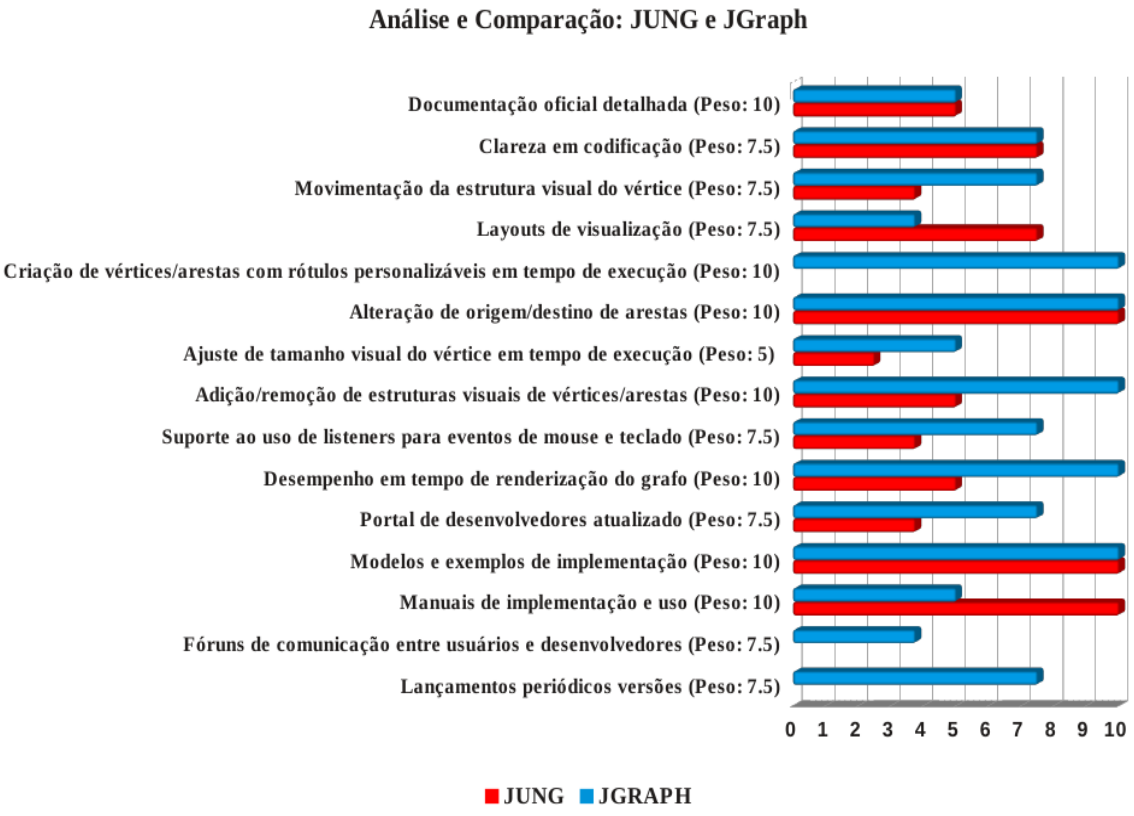

Figura 4. Gráfico: Análise e Comparação - JUNG e JGraph

4. Considerações Finais

Com o resultado da análise aqui apresentada, com base nos requisitos específicos listados na subseção 3.1, fica claro que o *framework* JGraph apresentou-se de forma satisfatória em muitos requisitos aqui levantados, sendo a maioria de peso 10, essenciais para a elaboração do editor para os N-Grafos. A JGraph como *framework* tem pontos a melhorar, porém de modo geral apresentou-se melhor do que a JUNG, sendo assim o *framework* selecionado para a próxima etapa do trabalho.

Além da análise e comparação dos *frameworks*. Destaca-se outra contribuição deste trabalho: a definição dos requisitos e técnicas para análise de *frameworks* para grafos. Portanto, o trabalho aqui apresentando poderá ser usado para futuras pesquisas que almejam criar aplicações computacionais que necessitem de uma ferramenta de visualização e edição para grafos.

Após a elaboração deste trabalho, foi constatado um novo *framework* para a manipulação e visualização de grafos, o Prefuse [Prefuse 2011]. Estudos já foram destinados a este *framework*, porém ainda são um pouco incipientes. Assim, será necessário repetir a metodologia aqui apresentada para avaliar esse novo *framework*. Tendo finalizada essa nova análise, a etapa subsequente será iniciada, a qual será responsável pela criação da ferramenta para N-Grafos.

Referências

Alves, G. V. (2009). *Transformations for proof-graphs with cycle treatment augmented via geometric perspective techniques*. PhD thesis, Universidade Federal de Pernambuco.

de Oliveira, A. G. (2001). *Proofs from a Geometric Perspective.* PhD thesis, Universidade Federal de Pernambuco.

dos Santos, D. V. (2012). Representação computacional para grafos de prova. Trabalho de conclusão de curso, Universidade Tecnológica Federal do Paraná.

Harary, F. (1969). *Graph Theory*. Addison-Wesley.

JGraph (2012). *JGraphX (JGraph 6) User Manual*. Disponível em: http://jgraph.com/doc/mxgraph/index javavis.html 2.2.3.4. Acessado em Agosto de 2012.

JUNG (2009). *JUNG 2.0 Tutorial*. Disponível em: http://www.grotto-networking.com/JUNG/JUNG2-Tutorial.pdf. Acessado em Agosto de 2012.

Kaspczak, A. and Rodrigues, L. G. (2011). Implementação de um algoritmo para verificação de ciclos em grafos-de-prova. Trabalho de conclusão de curso, Universidade Tecnológica Federal do Paraná.

Prefuse (2011). *The prefuse visualization toolkit.* Disponível em: http://presufe.org. Aces- sado em Agosto de 2012.

Trochim, W. M. (2006). Likert Scaling. Disponível em: http://www.socialresearchmethods.net/kb/scallik.php. Acessado em Agosto de 2012.