SENAI CIMATEC

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL E TECNOLOGIA INDUSTRIAL

Mestrado em Modelagem Computacional e Tecnologia Industrial

Dissertação de Mestrado

Modelagem de Algoritmos de Distribuição Espacial de Grafos: Uma extensão da UML para Aplicações de Visualização de Redes Sociais e Complexas

Apresentada por: Claudinei Carlos dos Santos Costa
Orientador: Hernane Borges de Barros Pereira

Maio de 2018

Claudinei Carlos dos Santos Costa

Modelagem de Algoritmos de Distribuição Espacial de Grafos: Uma extensão da UML para Aplicações de Visualização de Redes Sociais e Complexas

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Área de conhecimento: Interdisciplinar

Orientador: Hernane Borges de Barros Pereira
SENAI CIMATEC

Salvador
SENAI CIMATEC
2018

Ficha catalográfica elaborada pelo Centro Universitário SENAI CIMATEC

C837m Costa, Claudinei Carlos dos Santos

Modelagem de algoritmos de distribuição espacial de grafos: uma extensão da UML para aplicações de visualização de redes sociais e complexas / Claudinei Carlos dos Santos Costa. – Salvador, 2017.

162 f. . il. color.

Orientador: Prof. Dr. Hernane Borges de Barros Pereira.

Dissertação (Mestrado em Modelagem Computacional e Tecnologia Industrial) – Programa de Pós-Graduação, Centro Universitário SENAI CIMATEC, Salvador, 2017.
Inclui referências.

1. Modelagem de software. 2. Algoritmos de visualização de grafos.. 3. Modelagem computacional. I. Centro Universitário SENAI CIMATEC. II. Pereira, Hernane Borges de Barros. III. Título.

CDD: 005.1

NDI - 06

Nota sobre o estilo do PPGMCTI

Esta dissertação de mestrado foi elaborada considerando as normas de estilo (i.e. estéticas e estruturais) propostas aprovadas pelo colegiado do Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial e estão disponíveis em formato eletrônico (*download* na Página Web http://ead.fieb.org.br/portal_faculdades/dissertacoes-e-teses-mcti.html ou solicitação via e-mail à secretaria do programa) e em formato impresso somente para consulta.

Ressalta-se que o formato proposto considera diversos itens das normas da Associação Brasileira de Normas Técnicas (ABNT), entretanto opta-se, em alguns aspectos, seguir um estilo próprio elaborado e amadurecido pelos professores do programa de pós-graduação supracitado.

Dedico este trabalho à minha família, amigos e especialmente a Clamilton Carlos dos Santos Costa. O apoio de vocês foi fundamental para que a realização deste sonho fosse possível.

Agradecimentos

Agradeço a todos que contribuíram de forma direta ou indireta para que a realização deste sonho fosse possível. Agradeço à minha família e amigos, pois são combustível para me seguir em frente. Gostaria de agradecer ao meu orientador Hernane Borges de Barros Pereira pela paciência e ensinamentos. Obrigado Mestre.

Não citarei nomes para não cometer injustiça, mas preciso, neste momento, fazer um agradecimento especial. Clamilton Costa, onde você estiver, muito Obrigado.

Salvador, Brasil
dia 29 de Maio de 2018

Claudinei Carlos dos Santos Costa

Resumo

A visualização de informações em redes sociais e complexas apoia os pesquisadores no processo de análise e identificação de informações que determinem o comportamento das redes, a este processo é dado o nome de inspeção visual. A área de pesquisa Visualização da Informação possui uma subárea, denominada Visualização de Redes, que se ocupa em definir premissas, técnicas e regras estéticas para a construção de leiautes de grafos de redes sociais e complexas desta forma, as informações contidas nas redes poderão ser observadas de forma mais rápida e clara. O objetivo deste trabalho foi a proposição de uma extensão para a linguagem unificada de modelagem (UML), denominada de *NET-UML*, que auxiliará a construção de ferramentas que tenham como uma de suas funcionalidades a visualização de redes sociais e complexas. Para a construção deste trabalho realizamos revisão sistemática onde verificamos, ao longo dos anos, crescimento no número de pesquisas que discutem algoritmos de distribuição espacial de redes e a inexistência de técnicas ou métodos específicos que apoiem a construção de ferramentas computacionais para o domínio das aplicações de visualização de redes sociais e complexas. Com esta lacuna, aspectos sintáticos e semânticos, inerentes ao domínio das redes sociais e complexas, deixarão de ser documentados ou serão pouco percebidos por estarem generalizados/misturados a outros conceitos. Para categorizar/documentar as propriedades e comportamentos de alguns algoritmos de distribuição espacial de grafos em duas dimensões, realizamos estudo sobre os algoritmos de distribuição espacial de grafos mais utilizados para a inspeção visual de redes sociais e complexas. Este estudo serviu de subsídio para a criação do modelo *NET-UML*, testado durante a construção da ferramenta *Open Source SC NET DRAW*. Foi realizado também, estudo e categorização de algoritmos de distribuição espacial de grafos, utilizados para inspeção visual de redes sociais e complexas para entendimento das propriedades e comportamentos específicos e comuns a categoria que os algoritmos pertencem. Os resultados obtidos foram satisfatórios, pois a *NET-UML* potencializa, através do uso de representação icônica, a abstração dos objetos e funcionalidades necessárias para aplicações computacionais de visualização de redes sociais e complexas.

Palavras-chave: Modelagem de Software, Algoritmos de Visualização de Grafos, Modelagem Computacional.

Abstract

The visualization of information in social and complex networks supports the researchers in the process of analysis and identification of information that determine the behavior of networks, to this process is called visual inspection. The Information Visualization research area has a subarea, called Network Visualization, which is responsible for defining premises, techniques and aesthetic rules for the construction of complex and social network graphs in this way, the information contained in the networks can be observed more quickly and clearly. The purpose of this work was to propose an extension to the unified modeling language (UML), called NET-UML, which will help the construction of tools that have as one of their functionalities the visualization of social and complex networks . For the construction of this work, we performed a systematic review where, over the years, we verified a growth in the number of researches that discuss algorithms of spatial distribution of networks and the lack of specific techniques or methods that support the construction of computational tools for the domain of the applications of social and complex networks. With this shortcoming, syntactic and semantic aspects, inherent to the domain of social and complex networks, will no longer be documented or will be poorly perceived because they are generalized/mixed with other concepts. In order to categorize/document the properties and behaviors of some algorithms of spatial distribution of graphs in two dimensions, we performed a study on the algorithms of spatial distribution of graphs most used for the visual inspection of social and complex networks. This study served as a basis for the creation of the NET-UML model, which was tested during the construction of the Open Source SC NET DRAW tool. It was also carried out, study and categorization of algorithms of spatial distribution of graphs, used for visual inspection of social and complex networks to understand the properties and behaviors specific and common to the category that the algorithms belong to. The results obtained were satisfactory, because NET-UML potentiates, through the use of iconic representation, the abstraction of the objects and functionalities necessary for computational applications of visualization of social and complex networks.

Keywords: Software Modeling, Graph Visualization Algorithms, Computational Modeling.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

API	Application Program Interface.
GEO-OMT .	Geographic Object Modeling Technique.
OMT	Object Modelling Technique.
OO	Orientação a Objetos.
OSE	Object Oriented Software Engineering.
PERT	Técnica de Avaliação e Revisão de Programas.
SIG	Sistema de Informação Geográfica.
SCRCC	Sistema de Criação de Redes de Colaboração Científica.
UML	Unified Modeling Language.

Capítulo Um

Introdução

As pesquisas sobre redes sociais e complexas vêm evoluindo ao longo do tempo e têm como suporte o uso de ferramentas computacionais. Tais ferramentas propiciam análise estatística e inspeção visual dos grafos originados pelas redes. A tarefa de abstração e modelagem para construção destas ferramentas não considera a semântica e a sintaxe necessárias para o domínio de redes sociais e complexas. Leva em conta somente questões relacionadas com o paradigma de desenvolvimento utilizado com isso, detalhes importantes podem passar despercebidos.

1.1 Definição do Problema

Segundo Mazza (2009), o processo de transformar dados em sabedoria se dá a partir da transformação de dados em informação, informação em conhecimento e conhecimento em sabedoria, obviamente, assumindo a existência de conhecimentos anteriores a cerca do assunto de interesse, neste contexto, o uso de elementos pictográficos apoiam o processo de geração de conhecimento, uma vez que elementos visuais são rapidamente entendidos e processados pelo cérebro humano.

A área de conhecimento Visualização da Informação possui uma subárea chamada de Visualização de Redes, que se ocupa da discussão de premissas e técnicas que apoiam a representação de informação em redes. Com base nestes conceitos são criados leiautes de distribuição e visualização de grafos utilizados para análise de redes sociais e complexas (Mazza, 2009).

A partir da análise dos códigos fontes e estrutura de dados utilizadas na construção das ferramentas e bibliotecas *Open Source* especialistas em visualização de redes sociais e complexas, desenvolvidas com o uso do paradigma da orientação a objetos, verificamos que as mesmas formam modeladas com o uso de técnicas que não consideram de forma adequada a semântica e a sintaxe inerentes ao domínio que tais ferramentas discutem. As ferramentas utilizadas como referência para a construção deste trabalho foram *Gephi* versão 0.9.2, *iGraph* versão 0.7.1, jGraph versão 0.9.0 e Graph Sharp versão 1.0, porém as mesmas não disponibilizam em seus repositórios ou sites a documentação técnica utilizada para a construção dos seus respectivos modelos de análise.

1.2 Objetivo

O objetivo geral desta pesquisa é propor uma extensão para a linguagem unificada de modelagem (UML), de modo a potencializar o desenvolvimento de ferramentas computacionais que possuam como uma de suas funcionalidades a visualização de redes sociais e complexas a partir da distribuição espacial de seus respectivos grafos.

Os objetivos específicos são:

- Investigar a categorização de algoritmos de distribuição espacial de grafos de redes sociais e complexas;
- Analisar ferramentas de visualização de redes sociais e complexas e as técnicas de modelagem utilizadas para sua construção;
- Criar estereótipos através de modelo icônico para classificar objetos relacionados com distribuição espacial de grafos de redes sociais e complexas e;
- Construir ferramenta *Open Source* para visualização de redes sociais e complexas em ambiente bidimensional, como forma de testar o modelo proposto pela presente pesquisa.

1.3 Importância da Pesquisa

A construção de ferramentas computacionais requer exercícios de abstração dos aspectos do mundo real para o mundo computacional. Quanto mais especializada for a ferramenta maior deve ser a análise conceitual, sintática e semântica. O crescimento do número de pesquisadores que utilizam redes sociais e complexas como ferramenta de pesquisa, provoca a necessidade de oferta de ferramentas computacionais que sejam capazes de apoiar a realização destes trabalhos. A existência de novas técnicas e linguagens para construção da modelagem de ferramentas especialista em visualização de redes sociais e complexas, potencializará a evolução e surgimento de novas ferramentas computacionais para este domínio de aplicação.

As técnicas de modelagem de ferramentas computacionais existentes não caracteriza impeditivo para a realização de novos estudos e proposição de novas técnicas de modelagem umas vez que através de pesquisa de revisão sistemática não identificamos trabalhos anteriores que discutissem técnicas, metodologias ou linguagens para modelagem deste tipo de aplicação.

Além de propor a extensão *NET-UML*, está pesquisa se propõe a desenvolver um software

Open Source com o uso da *NET-UML* como forma evidenciar a sua importância. O desenvolvimento do software utilizará técnicas que possibilitem a evolução do mesmo por outros pesquisadores.

1.4 Motivação

Com a realização da revisão sistemática para construção deste trabalho, apresentada no Capítulo 2, foi possível verificar o crescimento no número de pesquisas que discutem visualização de grafos de redes sociais e complexas e as proposições de novos algoritmos de distribuição leiautes de grafos, como consequência do surgimento de ferramentas computacionais que dão suporte aos pesquisadores de redes sociais complexas em tarefas analíticas através da inspeção visual.

A construção de ferramentas computacionais necessita de linguagens, técnicas e metodologias que apoiem o entendimento dos requisitos e conceitos envolvidos no ambiente onde tais ferramentas estão inseridas assim, o produto final atenderá plenamente aos seus objetivos. Existem linguagens genéricas capazes de apoiar o desenvolvimento de qualquer tipo de ferramenta computacional da mesma forma que técnicas especializadas que cobrem de forma específica determinados tipos de aplicações. Neste cenário, temos a linguagem unificada de modelagem (UML), que pode apoiar o desenvolvimento de ferramentas computacionais que utilizam o paradigma OO para qualquer domínio de aplicação e o modelo GEO-OMT, especializado para o desenvolvimento de sistemas de informação geográfica (SIG).

Durante a revisão sistemática foi observado a inexistência de linguagens, técnicas ou metodologias específicas, que apoiem a construção de ferramentas que tenham como funcionalidades a visualização de redes sociais e complexas. Esta lacuna demostra a oportunidade para o desenvolvimento de ferramentas especialistas que cubram as questões que o domínio de redes sociais e complexas necessitam haja visto o potencial para surgimento de novas ferramentas computacionais para este domínio de aplicação.

1.5 Aspectos Metodológicos

Após a definição do tema para realização da presente pesquisa e construção da dissertação, foi inciada a revisão sistemática, apresentada no Capítulo 2, objetivando identificar trabalhos anteriores sobre o tema escolhido e ter direcionamento para a condução dos trabalhos. Em paralelo à revisão sistemática, foi construída ferramenta Sistema de Criação de Redes de Colaboração Científica (SCRCC), que possui funcionalidades de recuperação de

dados de produção bibliográfica de pesquisadores a partir de currículos lattes; criação de redes unimodais e bimodais de coautoria, palavras chave e área de conhecimento além da importação de arquivos de redes criados nos software Gephi e Pajek.

A construção da SCRCC possibilitou o entendimento dos conceitos e a sintaxe envolvida neste domínio de aplicação de redes e proporcionou publicações de artigos em eventos científicos como IV Workshop de Pesquisa Tecnologia e Inovação (PTI), e VIII Encontro de Física Aplicada (DOI: 10.5151/phypro-viii-efa-46).

Durante a realização da revisão sistemática, foi identificado trabalho correlato que discute modelagem de ferramentas para domínio dos sistemas de informações geográficas (SIG) que faz uso do paradigma da orientação a objetos (OO). O mesmo serviu como inspiração para a realização da presente pesquisa. Apresentamos no Apêndice B, discussão sobre o trabalho correlato, modelo GEO OMT.

Após análise do trabalho correlato, foi realizado estudo sobre algoritmos de distribuição de grafos para entender as técnicas de visualização de informação em redes e critérios estéticos que devem ser respeitados.

A sexta etapa da pesquisa foi a construção do modelo *NET-UML*, realizada após entendimento das questões conceituais, comportamentais e técnicas relacionadas com a teoria e visualização de redes sociais e complexas e modelagem da ferramentas computacionais desenvolvidas a partir do paradigma da orientação a objetos. O modelo *NET-UML* será discutido no Capítulo 4, Seção 4.1.

Após a construção do modelo *NET-UML*, foi desenvolvida a ferramenta *Open Source SC NET DRAW* como forma de testar o modelo proposto. Os resultados obtidos serão apresentados no Capítulo 5.

Apresentamos na Figura 1.1 de forma sequencial, os passos para a realização da presente pesquisa.

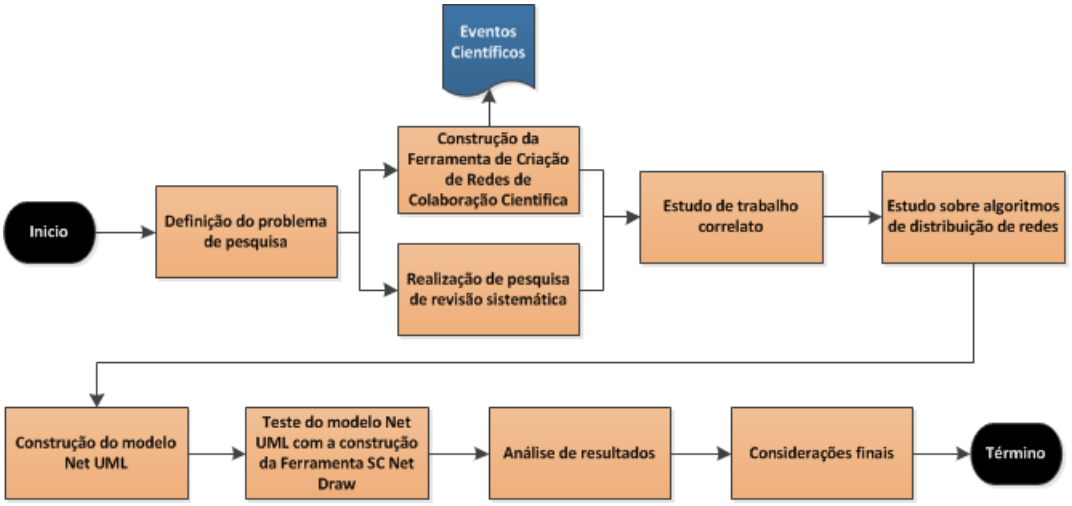


Figura 1.1: Passos para realização da pesquisa. Fonte: Próprio autor

1.6 Organização da Dissertação de Mestrado

A presente pesquisa está organizada da seguinte forma:

- **Capítulo 2 - Revisão Sistemática**: Capítulo que apresenta revisão sistemática de trabalhos desenvolvidos por outros pesquisadores de redes sociais e complexas, especificamente no que tange a técnicas de modelagem ferramentas de visualização de redes sociais e complexas, construídas a partir do paradigma de desenvolvimento orientação a objetos;
- **Capítulo 3 - Algoritmos de Leiaute de Grafos**: Capítulo responsável por apresentar conceitos de visualização da informação, teoria dos grafos, redes sociais e complexas e algoritmos de distribuição de grafos, bem como, estratégias de categorização dos algoritmos de distribuição de grafos apresentados neste trabalho, a partir de análises quantitativa e qualitativa;
- **Capítulo 4 - *NET-UML***: Este capítulo discute conceitos relacionados com o paradigma de desenvolvimento orientado a objetos e a linguagem de modelagem unificada (UML) e, por fim, apresenta proposta de extensão da linguagem UML, denominado *NET-UML*, para uso na criação de modelos de análise de software de visualização de redes sociais e complexas;
- **Capítulo 5 - Resultados e Discussões**: Neste capítulo serão apresentados os resultados obtidos no desenvolvimento da ferramenta de visualização de redes sociais e complexas, denominada *SC NET DRAW*, a partir do uso da *NET-UML* e

comparação do modelo de análise desta com o modelo de análise de ferramentas de visualização de redes .

- **Capítulo 6 - Considerações Finais**: Capítulo responsável por apresentar as considerações finais, contribuições da pesquisa e atividades futuras a serem desenvolvidas a partir dos resultados obtidos.

Capítulo Dois

Revisão Sistemática Sobre Técnicas de Modelagem de Ferramentas de Visualização de Redes Sociais e Complexas

A realização de pesquisas suportadas por redes sociais e complexas nos ajuda a perceber o comportamento de diversos tipos de sistemas, seja através da análise estatística ou da inspeção visual das redes (Albert; Barabasi, 2002), (Newman, 2010) e (Barabási, 2010).

A área de visualização da informação discute técnicas de apresentação de informações contidas em redes sociais e complexas, a partir da criação de imagens de grafos, que apoiam a inspeção visual das redes (Mazza, 2009).

Existem ferramentas computacionais (e.g *Tulip*, *Gephi*, *iGraph* e *Graph Shrap*) desenvolvidas a partir do paradigma da orientação a objetos que propiciam a inspeção visual e manipulação de redes sociais e complexas. No processo de desenvolvimento de tais ferramentas, é esperado que sejam utilizadas técnicas que considerem a semântica e à sintaxe que o domínio deste tipo de aplicação está inserido.

Larman (2007) e Wazlawick (2011) defendem em suas obras que o desenvolvimento de software orientado a objetos deve considerar os aspectos conceituais inerentes ao domínio onde o mesmo está inserido. Neste contexto, a Linguagem de Modelagem Unificada (UML) fornece vasto ferramental para a construção dos artefatos necessários para o desenvolvimento de software cobrindo os aspectos estruturais, comportamentais e de interação.

Apresentamos neste capitulo revisão sistemática de trabalhos desenvolvidos por outros pesquisadores de redes sociais e complexas.

O objetivo desta revisão sistemática é a investigação de algoritmos de distribuição espacial de grafos utilizados no estudo de redes sociais e complexas, suas propriedades e comportamentos, bem como a verificação de técnicas de modelagens especificas para a construção de ferramentas computacionais visualização de redes sociais e complexas.

Foram levantados artigos, teses e livros que serviram de suporte para o desenvolvimento da presente pesquisa. A revisão sistemática teve como plataforma de pesquisa os portais de periódicos *ACM Digital*, *Web of Science*, *Science Direct* e IEEE.

2.1 Metodologia de Revisão Sistemática

Mancini e Sampaio (2007) definem revisão sistemática como um tipo de pesquisa que busca na literatura evidências de pesquisas anteriores sobre determinado tema. Esse mecanismo se da a partir do uso de técnicas de busca, seleção, avaliação e crítica dos trabalhos encontrados, indicando o cenário atual das pesquisas sobre o tema.

Mancini e Sampaio (2007) propõem como etapas de um revisão sistemática: definição da pergunta cientifica; definição das bases de dados de consulta; estabelecimento dos critérios de buscas e seleção; realização das buscas nas bases de dados definidas; aplicação de critérios de seleção e exclusão de trabalhos; análise critica dos trabalhos selecionados e; apresentação da conclusão a cerca dos resultados obtidos.

Apresentamos na Figura 2.1, de forma sequencial, os passos para a realização de uma revisão sistemática.

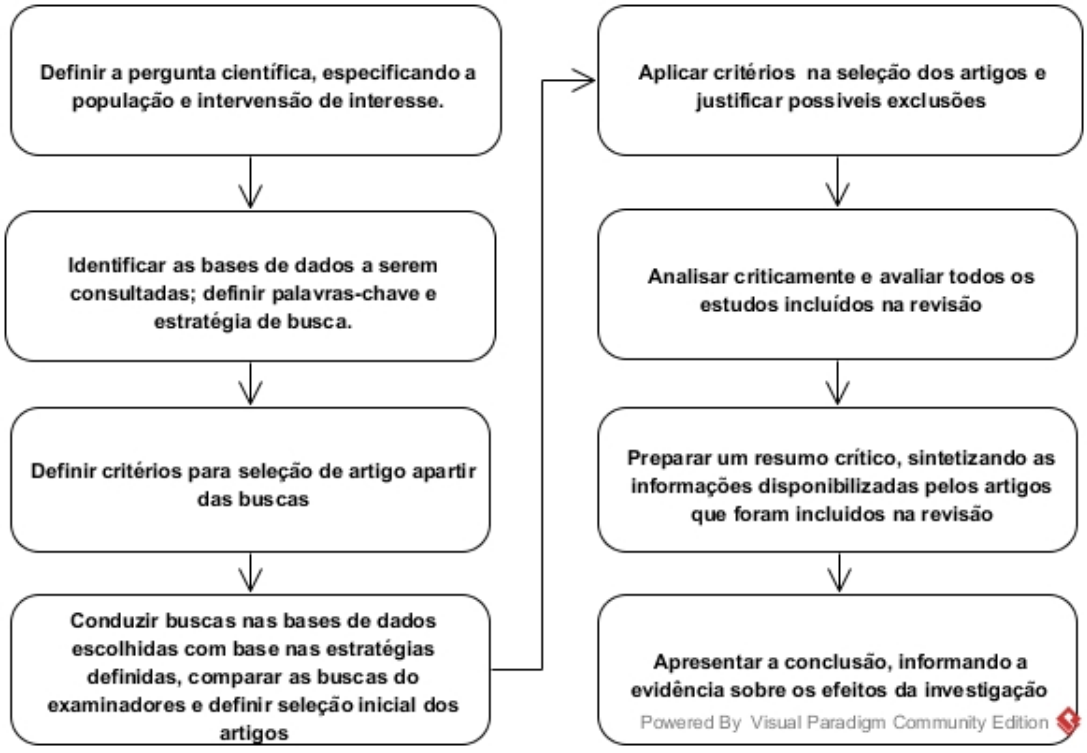


Figura 2.1: Passos da uma revisão sistemática.Fonte: Adaptado de Mancini e Sampaio (2007)

2.1.1 Questão de Pesquisa

O objetivo desta revisão sistemática foi realizar uma análise sobre algoritmos de leiautes de redes e suas implementações em ferramentas especialistas com o intuito de responder

a seguinte questão: Existem técnicas de modelagem para construção de ferramentas de visualização de redes sociais e complexas?

2.1.2 Selecionando a Evidência Científica

Foram utilizados como fonte para a realização da pesquisa os portais de periódicos *ACM Digital*, *Science Direct*, *Web of Science* e *IEEExplorer*, os Livros: Projeto de Algoritmos (Ziviane, 2011); Engenharia de Software Uma abordagem profissional (Presman, 2011); Utilizando UML e Padrões (Larman, 2007); Análise e projeto de sistemas de informação orientados a objetos (Wazlawick, 2011); *Complex Networks – An Algorithmic Perspective*(Ercies, 2015); *Handbook of Graph Drawing and Visualization*(Tamassia, 2013) e; Análise de Redes Sociais Uma Visão Computacional (Gabardo, 2015).

Dado a abrangência da pesquisa foi necessário criar três grupos de descritores para que fosse possível identificar estudos que os discutissem: algoritmos de visualização de redes sociais e complexas; técnicas e metodologias de visualização de redes sociais e complexas e; técnicas de modelagem de ferramentas de visualização de redes sociais e complexas.

Os descritores *Algorithm of visualization of Graph*, *Algorithm of visualization of Network*, *Algorithm of visualization of Complex Network* e *Algorithm of visualization of Social Network* foram utilizados para com o intuito de identificar trabalhos que discutissem algoritmos de visualização de redes sociais e complexas.

Para a identificação de trabalhos que falassem de técnicas e metodologias de visualização de redes sociais e complexas foram utilizados os descritores *Draw network*, *Graph drawing*, *Layout draw graph*, *Graph visualization*, *Network visualization*, *Social Network Visualization*, *Complex Network Visualization*, *Graph Drawing and Visualization*, *NetWork Drawing and Visualization*, *Graph Drawing and Visualization*, *Complex NetWork Drawing and Visualization* e *Social NetWork drawing and visualization.*

Os descritores *Modeling of Network Visualization Software*, *Modeling of Graph Visualization Software*, *Modeling of Social Network Visualization Software* e *Modeling of Complex Network Visualization Software* foram utilizados para identificar trabalhos que discutissem técnicas de modelagem de ferramentas de visualização de redes sociais e complexas.

2.1.3 Critérios de Inclusão e Exclusão

Para a inclusão ou exclusão de pesquisas nesta revisão, foram definidos os seguintes critérios:

2.1.3.1 Critérios de Inclusão

- Trabalhos sobre técnicas de modelagem de software de manipulação de grafos, redes sociais e ou complexas - critério que apoia a identificação de técnicas de modelagem especificas para software de manipulação de redes.
- Trabalhos sobre a teoria dos grafos, redes sociais e complexas - critério que apoia a identificação dos aspectos conceituais da teoria das redes, necessários para proposição da semântica e sintaxe do modelo proposto na presente pesquisa.
- Trabalhos sobre visualização de grafos, redes sociais e complexas - critério que apoia a identificação de produções que discutam aspectos conceituais e estéticos presente na visualização de informação em redes, necessários para implementação de ferramentas neste domínio de aplicação.
- Trabalhos sobre algoritmos de visualização e distribuição de redes sociais e complexas - critério que apoia a identificação de algoritmos específicos de distribuição espacial de redes sociais e complexas, para entendimento dos aspectos semânticos, comportamentais e propriedades que podem emergir da visualização das redes sociais e complexas, a partir da distribuição proporcionada por tais algoritmos.

2.1.3.2 Critérios de Exclusão

- Trabalhos que não falem sobre técnicas de modelagem de software de manipulação de grafos, redes sociais e ou complexas;
- Trabalhos que não falem sobre a teoria dos grafos, redes sociais e complexas.
- Trabalhos que não falem sobre visualização de redes sociais e complexas.
- Trabalhos que não falem sobre algoritmos de visualização e distribuição de redes sociais e complexas.

2.1.4 Condução da Revisão da Literatura

O levantamento dos trabalhos aderentes à pesquisa foi realizado entre 01 a 31/05/2017 e, como mencionado anteriormente, as pesquisas foram realizadas nos portais de periódicos *ACM Digital*, *Science Direct*, *Web of Science*, *IEEExplorer*.

Foram levantados inicialmente 3913 trabalhos. De posse destes, foi realizada verificação de duplicação. Com isso, o número de publicações reduziu para 2665.

Apresentamos na Tabela 2.1.4 e na Figura 2.2, respectivamente, quantitativos de publicações por descritores e portais de pesquisas. Os descritores *Algorithm of visualization of Graph*, *Algorithm of visualization of Network*, *Algorithm of visualization of Complex Network*, *Algorithm of visualization of Social Network*, *Layout draw graph*, *Complex Network visualization*, *NetWork drawing and visualization*, *Graph drawing and visualization*, *Complex NetWork drawing and visualization*, *Social NetWork drawing and visualization*, *Modeling of network visualization software*, *Modeling of graph visualization software*, *Modeling of social network visualization software* e *Modeling of complex network visualization software* não apresentaram resultados enquanto os descritores *Graph Drawing* e *Network visualization* apresentaram grande número de publicações.

Tabela 2.1: Publicações por descritores e portais de pesquisa.

Descritor	**Web of Science**	**Science Direct**	**IEEE**	**ACM DIGITAL**
Algorithm of visualization of Graph	0	0	0	0
Algorithm of visualization of Network	0	0	0	0
Algorithm of visualization of Complex Network	0	0	0	0
Algorithm of visualization of Social Network	0	0	0	0
Draw network	6	0	0	3
Graph drawing	1297	166	145	141
Layout draw graph	0	0	0	0
Graph visualization	624	37	190	112
Network visualization	760	50	166	114
Social Network visualization	54	2	16	21
Complex Network visualization	1	0	0	0
Graph drawing and visualization	8	0	0	0
NetWork drawing and visualization	0	0	0	0
Graph drawing and visualization	0	0	0	0
Complex NetWork drawing and visualization	0	0	0	0
Social NetWork drawing and visualization	0	0	0	0
Modeling of network visualization software	0	0	0	0
Modeling of graph visualization software	0	0	0	0
Modeling of social network visualization software	0	0	0	0
Modeling of complex network visualization software	0	0	0	0
Total	2750	255	517	391

Fonte: Próprio autor

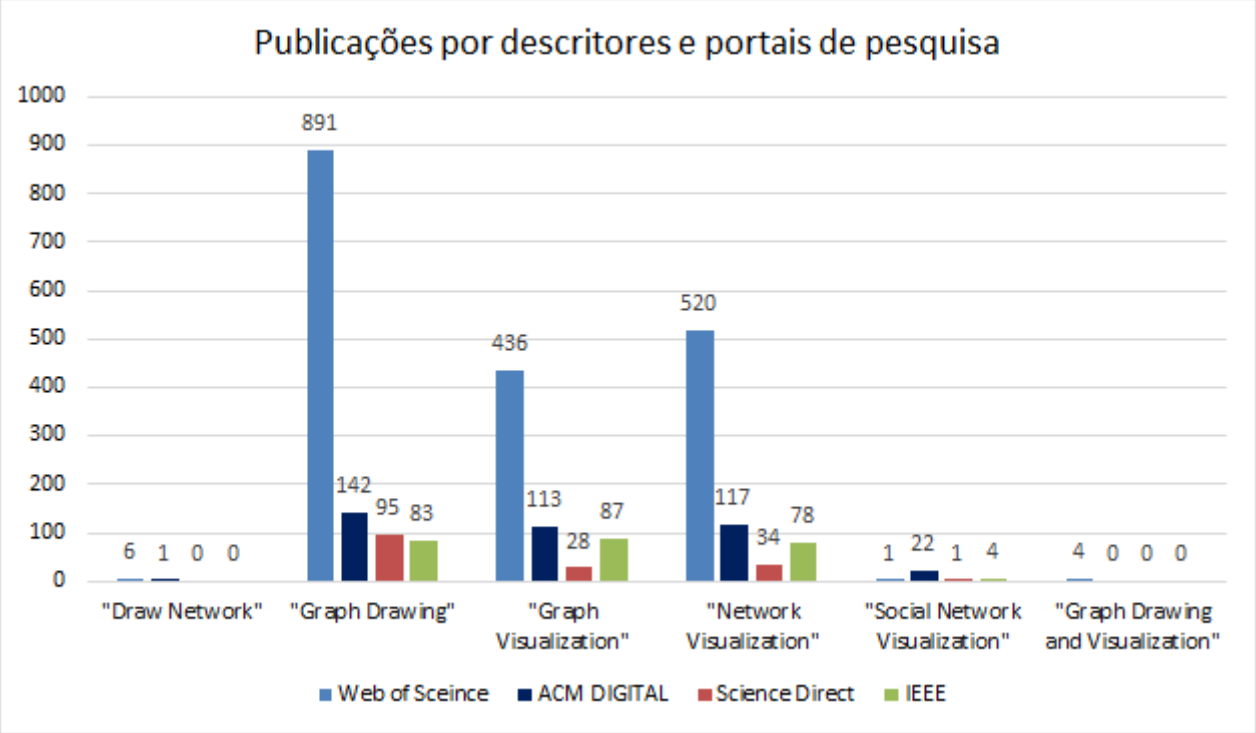


Figura 2.2: Publicações por descritores e portais de pesquisa. Fonte: Próprio autor

Não foram aplicados filtros temporais para busca das publicações. Desta forma foi observado que desde 1977 existem publicações sobre o tema tendo como a obra mais antiga *Automated Display Techniques for Linear Graphs* de (Maguire, 1977) apresentada na *4th Annual Conference on Computer Graphics and Interactive Techniques*, que discutiu técnicas automatizadas para apresentação de grafos lineares.

Foi observado na ultima década grande crescimento no número de publicações totalizando 1845(69% das publicações). Este número pode justificar o surgimento e aprimoramento das novas técnicas e ferramentas de visualização de redes sociais e complexas.

Apresentamos na Figura 2.3 o número de publicações por descritores e ano de publicação, nas quais é possível observar que entre 2004 e 2017 houve aumento no número de publicações no tema de visualização de redes sociais e complexas.

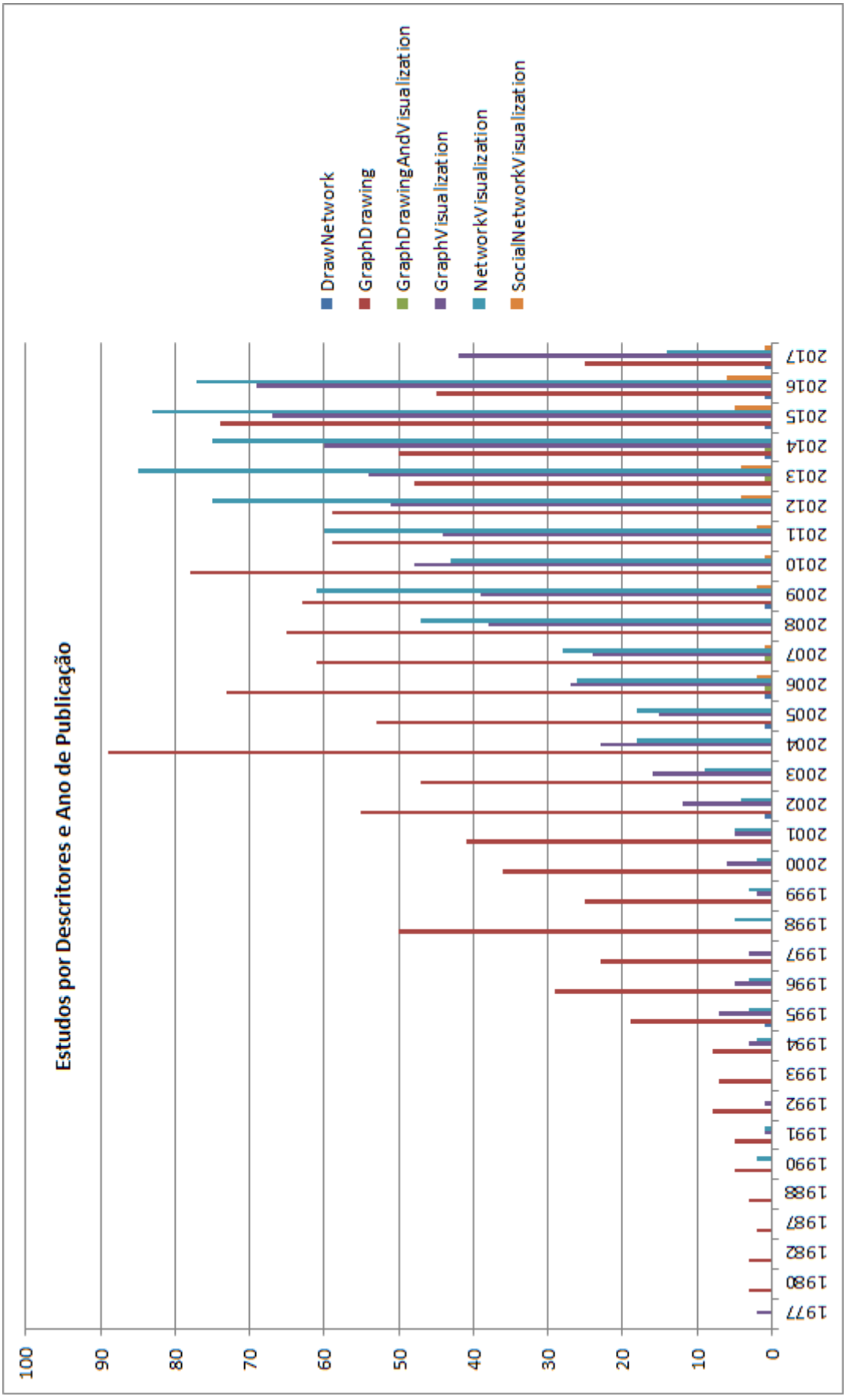


Figura 2.3: Estudos por descritores e ano de publicação.Fonte: Próprio autor

Após a eliminação de trabalhos duplicados, foi realizada verificação de aderência dos trabalhos restantes conforme descrito nos critérios de inclusão e exclusão, sendo identificadas 46 publicações com total aderência ao objeto de pesquisa deste trabalho.

2.1.5 Resultados

As publicações selecionadas foram categorizadas como algoritmos de leiaute de grafo e modelagem, sugerindo como as mesmas serão utilizadas no desenvolvimento da presente pesquisa.

O maior número dos trabalhos selecionados discute algoritmos de leiaute de grafo, o que contribui para o entendimento das técnicas computacionais envolvidas na visualização de grafos enquanto as publicações classificadas como modelagem contribuíram para verificação das questões teóricas e estéticas. Apresentamos na Tabela 2.2 quantitativo de publicações selecionadas por categoria e portal de periódicos.

Tabela 2.2: Publicações selecionadas por categoria.

Categoria	IEEE	Não indexado	ScienceDirect	WebOfScience	Total
Algoritmo de Leiaute de Grafo	12	0	6	14	32
Modelagem	3	1	2	8	14
Total	15	1	8	22	46

Fonte: Próprio autor

Foi verificado que 56% das publicações selecionadas possuem classificação Quális Capes nas áreas de avaliação Interdisciplina e Ciência da Computação, conforme apresentamos nas tabelas 2.3 e 2.4, sendo que destas 80% têm Quális A1 ou A2, o que indica a relevância das pesquisas relacionadas.

Apesar de 44% das publicações selecionadas não possuírem Quális Capes, isso não que dizer que são trabalhos de pouca relevância, pois, dentre estes estão publicações como Eades, Lin e Tamassia (1990), intitulado *An Algorithm for Drawing a Hierarchical Graph*, tido como referência quando se fala em algoritmos de desenho de grafos hierárquicos e McGuffin (2012), intitulado *Simple algorithms for network visualization: A tutorial* que apoia iniciantes em pesquisas de algoritmos de distribuição de grafos.

Tabela 2.3: Periódicos por Quális na Area de Avaliação Interdisciplinar.

Periódico	**A1**	**A2**	**B1**	**Total Geral**
BIOINFORMATICS (OXFORD. PRINT)	1	0	0	1
COMPUTER NETWORKS (1999)	0	1	0	1
FUTURE GENERATION COMPUTER SYSTEMS	0	1	0	1
INFORMATION SCIENCES	1	0	0	1
LECTURE NOTES IN COMPUTER SCIENCE	0	0	2	2
PLOS ONE	1	0	0	1
SOFTWARE, PRACTICE And EXPERIENCE (PRINT)	0	4	0	4
Total Geral	3	6	2	11

Fonte: Próprio autor

Tabela 2.4: Periódicos por Quális na Area de Avaliação Ciência da Computação.

Periódico	**A1**	**A2**	**B1**	**Total Geral**
ALGORITHMICA	0	1	0	1
COMPUTATIONAL GEOMETRY	0	4	0	4
IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS	6	0	0	6
JOURNAL OF DISCRETE ALGORITHMS (PRINT)	0	0	1	1
JOURNAL OF VISUAL LANGUAGES AND COMPUTING	0	0	2	2
Total Geral	6	5	3	14

Fonte: Próprio autor

2.2 Considerações do Capítulo

A revisão sistemática da literatura apresentada neste capítulo nos ajudou a perceber a evolução das pesquisas sobre visualização de redes através de algoritmos de distribuição de grafos e, por conseguinte, o surgimento de ferramentas de visualização e manipulação de redes como Gephi, Pajek, Ucinet, dando suporte aos pesquisadores que fazem uso de redes sociais e complexas como ferramenta de trabalho. A Figura 2.3 nos ajuda a perceber este comportamento.

Durante a realização da revisão sistemática, através do uso dos descritores citados na Seção 2.1.2, não identificamos trabalhos que discutam técnicas de modelagem específicas para construção de ferramentas de visualização de redes sociais e complexas. Analisamos também as ferramentas de código aberto *Gephi*, *iGraph*, *JGraph* e *Graph Sharp* e observamos que as estas ferramentas foram desenvolvidas a partir do paradigma OO e fizeram uso da linguagem unificada de modelagem (UML), mas, não identificamos nenhuma técnica de modelagem especifica para o domínio das aplicações de redes sociais e complexas.

A não identificação de trabalhos que discutam técnicas de modelagem especificas para construção de ferramentas de visualização de redes, sugere oportunidade para realização da presente pesquisa.

Identificamos durante a pesquisa, o trabalho correlato de Borges (1997a) intitulado Modelagem de Dados Geográficos: Uma Extensão do Modelo OMT para Aplicações Geográficas que será apresentado com maior detalhe também no Capítulo 4.

Capítulo Três

Algoritmos de Leiaute de Grafos

A análise visual de dados, apoiada pela cognição, em contraponto à análise textual, faz com que o processo de transformação dos dados em conhecimento seja mais rápido e intuitivo, uma vez que a percepção de informações contidas em pictogramas é percebida de forma mais rápida pelo cérebro humano (Mazza, 2009).

Os algoritmos de distribuição espacial grafos, também conhecidos como algoritmos de distribuição de rede ou algoritmos de leiaute de rede, organizam os elementos da rede de forma visual de acordo com o tipo de análise que se deseja fazer, indo além da criação de um grafo como figura plana, mas conseguindo criar uma figura que favoreça a geração de conhecimento a cerca do objeto de análise.

O objetivo deste capítulo é apresentar os algoritmos de distribuição espacial de grafos utilizados na análise de redes sociais e complexas de forma categorizada. Para tanto, faz-se necessário, discutir conceitos que norteiam a visualização de informação em redes como visualização da informação, teoria dos grafos e redes sociais e complexas.

Os algoritmos discutidos neste capítulo foram selecionados a partir da verificação dos estudos de Khokhar (2015), Brandes e Wagner (2013), Csardi e Nepusz (2006).

3.1 Visualização da Informação

A utilização de elementos pictográficos apoia o processo de transformação de dados em informação e esta em conhecimento, uma vez que os elementos visuais são rapidamente entendidos e processados pelo cérebro (Mazza, 2009). A visualização da informação possui uma subárea chamada de visualização de redes, que se ocupa da discussão de premissas e técnicas que apoiam a representação de informação e a partir destes conceitos são criados os algoritmos de distribuição e visualização de redes.

3.2 Teoria dos Grafos

Grafo é um elemento matemático muito utilizado para representação e estudo das características das redes sociais e complexas. Tem representação matemática $G = (V, E)$ em que V representa o conjunto dos elementos individualizados chamados de vértices e E

representa as arestas, que são as conexões entre os vértices (Barabási, 2010; Gabardo, 2015).

Os estudos sobre a teoria dos grafos foi inaugurado por Euler (1736) em artigo publicado em 1736, conhecido como *Solutio problematis ad geometriam situs pertinentis*, que apresentou novos conceitos matemáticos para representar a solução do problema da travessia das sete pontes de *Konisgber* sem passar pela mesma ponte mais de uma vez.

Euler (1736) representou as localidades como vértices e as pontes como arestas e provou, matematicamente, com uso de métricas de grafos, a impossibilidade de realizar tal travessia sem cruzar, pelo menos, umas das pontes mais de uma vez, pois, para que fosse possível realizar os vértices, deveriam ter grau ou número de conexões par para um número impar de arestas. O mesmo não ocorreu com as pontes de *Konisgber* conforme apresentado na Figura 2.1. As propriedades de grafos discutidas neste trabalho serão apresentadas na Seção 3.2.1.

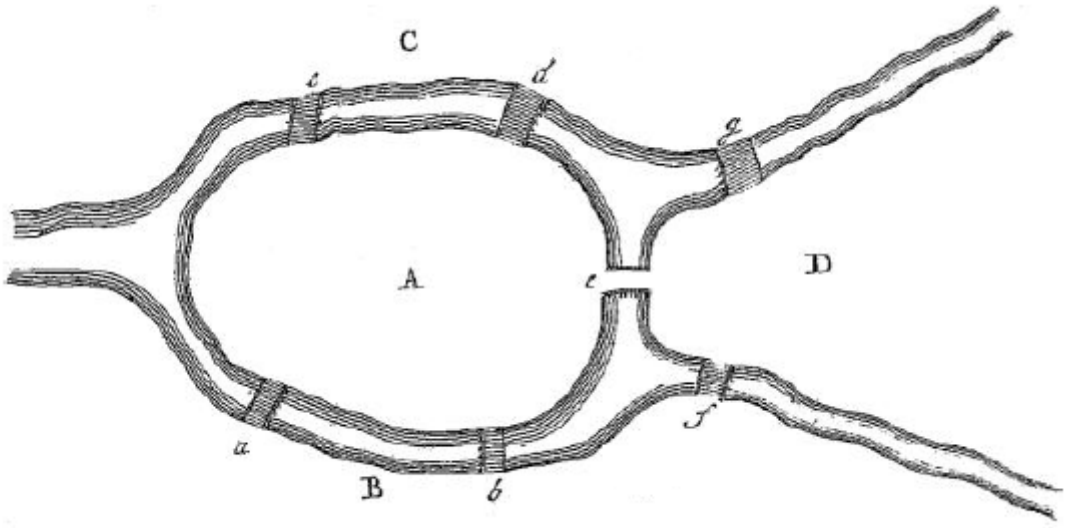


Figura 3.1: Pontes de *Konisgber*.Fonte: Euler (1736)

Existem diversos tipos de grafos, porém neste trabalho discutiremos apenas leiaute de distribuição para grafos dirigidos: grafos em que se faz necessário indicar o sentido das arestas; grafos não dirigidos onde não se faz necessário indicar o sentido das arestas e grafos ponderados nos quais são atribuídos pesos para as arestas, os grafos ponderados podem ser dirigidos ou não dirigidos. Apresentamos na Figura 3.2 grafos direcionados, não direcionados e ponderados. Todos os grafos possuem seis vértices com identificação alfanumérica de A a F, porém é possível verificar que as arestas dos grafos direcionados e ponderados possuem setas indicando o sentido da relação, o que não ocorre nas arestas do grafo não direcionado. Observa-se também que as arestas do grafo ponderado possuem

um valor indicando o peso da relação entre os vértices.

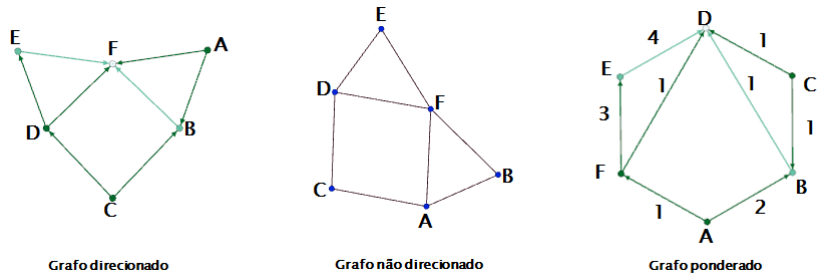


Figura 3.2: Tipos de grafos. Fonte: Próprio autor

3.2.1 *Propriedades de Grafos*

Os grafos apresentam propriedades matemáticas ou métricas que apoiam a análise de redes sociais e complexas na identificação dos padrões existentes na rede que o mesmo está representando. Algumas destas métricas são utilizadas também para criação da representação visual dos grafos. Abaixo apresentamos métricas de grafo utilizadas pelos leiautes de distribuição de grafos discutidos neste trabalho (Gabardo, 2015).

- Ordem – número de vértices de um grafo , denotado por $n = |V|$ (Gabardo, 2015; Boaventura, 2012).
- Tamanho – número de arestas ou conexões existentes em um grafo, denotado por $m = |E|$ (Gabardo, 2015; Boaventura, 2012).
- Grau – número de conexões de um vértice. Em um grafo direcionado existem dois tipos de Graus. Grau de Entrada, conexões que chegam no vértices, e Grau de Saída, conexões que saem do vértice (Gabardo, 2015; Boaventura, 2012).
- Grau Médio – em um grafo não dirigido o grau médio representa o número médio de conexões dos vértices da rede conforme a Equação 3.1 (Gabardo, 2015; Boaventura, 2012).

$$G_M = \frac{2.E}{V} \tag{3.1}$$

- Densidade – é o resultado da relação entre a ordem e o tamanho do grafo, obtido pela razão entre o número de arestas existentes e o número máximo de arestas possíveis (Gabardo, 2015; Boaventura, 2012). Apresentamos na Equação 3.2 do cálculo da densidade.

$$D = \frac{2.m}{n(n-1)} \tag{3.2}$$

- Distância Geodésica – menor caminho entre dois vértices de um grafo. Em um grafo podem existir mais de um caminho entre dois vértices, esta propriedade também é conhecida como caminho mínimo (Angelis, 2005; Boaventura, 2012).

- Diâmetro – é a maior distância geodésica existente em um grafo (Angelis, 2005).

- Centralidade de grau – propriedade de escopo local, relacionada ao vértice e sua vizinhança, que indica quão bem relacionado é determinado vértice pois, mede a proximidade do vértice em relação ao demais (Gabardo, 2015). A centralidade de grau é obtida pelo somatório das conexões que o vértice possui conforme apresentamos na Equação 3.3 onde i é o vértice que se deseja medir a centralidade de grau, a_{ij} é o elemento da matriz de adjacência (ver Seção 3.2.2) cujo valor será somado para a obtenção do valor centralidade de grau (Gabardo, 2015; Freeman, 1978-9). Apresentamos na Equação 3.3 o cálculo da centralidade de grau.

$$C_D(i) = \sum_{j=1}^{n} a_{ij} \tag{3.3}$$

- Centralidade de proximidade – propriedade de escopo global, relacionada com o vértice e os demais vértices da grafo, mede a proximidade de um determinado vértice aos demais vértices da grafo e seu cálculo é dado pela soma das distâncias geodésica em relação aos demais vértices das rede. esta propriedade só deve se calculada para grafos que possui apenas um componente pois, não é possível calcular a distância entre vértices que estão em diferentes componentes. O vértice com maior valor de centralidade de proximidade é aquele que dará o menor número de saltos para alcançar os demais vértices do grafo. O cálculo da centralidade de proximidade é dado pela Equação 3.4 onde n é o número de vértices do grafo, $\sum_{j=1}^{n} d(ij)$ é o somatório das distâncias geodésicas do vértice de acordo a quantidade de conexões entre os vertices i e j (Gabardo, 2015; Freeman, 1978-9). Apresentamos na Equação 3.4 o cálculo da centralidade de proximidade.

$$C_C(i) = \frac{n-1}{\sum_{j=1}^{n} d(ij)} \tag{3.4}$$

- Centralidade de intermediação – propriedade de escopo global baseada na quantidade de distâncias geodésicas entre vértices que passam por um determinado ponto do grafo, definindo o quão é estratégica a posição do vértice, o que sugere, que quanto maior for o valor desta propriedade, maior será o controle do fluxo de informações que o vértice poderá exercer neste grafo (Gabardo, 2015). O cálculo da centralidade de intermediação é dado pela Equação 3.5 onde $\theta ij(v)$ é o número das distâncias geodésicas entre os vértices i e j que passam pelo vértice v e θij é o número de distâncias geodésicas entre i e j (Gabardo, 2015; Freeman, 1978-9). Apresentamos

na Equação 3.5 o cálculo da centralidade de proximidade.

$$Cb(v) = \sum \frac{\theta ij(v)}{\theta ij} \tag{3.5}$$

Apresentamos na Tabela 3.1 lista dos algoritmos utilizados neste trabalho e as propriedades de grafo utilizadas para a construção do leiaute ou observadas na inspeção visual pós distribuição dos vértices no leiaute. Os algoritmos serão discutidos na Seção 3.4. As propriedades utilizadas na construção ou observadas pós execução dos algoritmos foram marcadas com o número 1. O número 0 indica que a propriedade não foi observada ou utilizada pelo algoritmo.

Ordem e Tamanho são as propriedades utilizadas por todos os leiautes ,pois representam respectivamente o número de vértices e arestas do grafo. Ademais, para construção dos leiautes foram utilizadas as propriedades: grau para os leiautes que evidenciam a existência grupos (e.g. *K-Core*, *Star*) ou hierarquias (e.g. *Tree*, *Sugiyama*), distância geodésica e diâmetro para o cálculo das distâncias teóricas entre os vértices, utilizado pelo leiaute *Kamada Kawai*.

A inspeção visual dos leiautes descritos na Tabela 3.1 nos possibilita observar propriedades como Centralidade de Grau (e.g. leiaute *Star*); Centralidade de Proximidade (e.g leiaute *K-Core*); Centralidade de Intermediação (e.g leiautes *Force Atlas*, *Kamada Kawai* e *Fruchterman Reingold Layout*) e Densidade (e.g leiautes *Contraction* e *Expansion*).

Tabela 3.1: Algoritmos de distribuição X Propriedade de grafos.

Algoritmo	Ordem	Tamanho	Grau	Densidade	D. Geodésica	Diâmetro	C. Grau	C. Proximidade	C. Intermediação
Ciruclar Layout	1	1	0	0	0	0	0	0	0
Star Layout	1	1	1	0	0	0	1	0	0
K-Core Layout	1	1	1	0	0	0	1	1	0
Tree Layout	1	1	1	0	0	0	0	0	0
Sugiyama Layout	1	1	1	0	0	0	0	0	0
ClockWise Rotate	1	1	0	0	0	0	0	0	0
Contraction	1	1	0	1	0	0	0	0	0
Counter ClockWise Rotate	1	1	0	0	0	0	0	0	0
Expansion	1	1	0	1	0	0	0	0	0
Force Atlas Layout	1	1	0	0	0	0	1	1	1
Fruchterman Reingold Layout	1	1	1	0	0	0	1	1	1
Kamada Kawai Layout	1	1	0	0	1	1	1	1	1

Fonte: Próprio autor

3.2.2 *Formas de Representação de Grafos*

Os dados de grafos podem ser representados e terem seus dados mantidos de diversas formas, desde imagens e arquivos até estruturas de dados robustas que propiciam a manipulação computacional através de algoritmos. Para a construção dos algoritmos de distribuição de grafos apresentados neste trabalho, utilizamos como estrutura de dados matriz de adjacência, listas de adjacências e tipo de dado grafo. A escolha se deu por estes atenderem de forma satisfatória os requisitos dos leiautes de distribuição de grafos e o paradigma de desenvolvimento de software utilizado (Gabardo, 2015).

As matrizes de adjacência são matrizes de duas dimensões de tamanho (n x n) onde n é a quantidade de vértices do grafo. Nesta estrutura, os vértices são representados por rótulos alfanuméricos e a existência de conexão entre os vértices é definida de forma binária. A existência de conexão é representada por 1 e a ausência de conexão é representada por 0. Lista de adjacência é uma matriz quadrada cuja a definição das suas dimensões se assemelha a da matriz de adjacência, porém suas linhas são utilizadas para representar listas individuais por vértice indicando quais outros vértices estes possui relação (Gabardo, 2015). O tipo de dados grafo é uma abstração do elemento matemático grafo que possui coleções de objetos vértices e arestas. Este tipo de dado será apresentado com mais detalhes no capítulo 4

Apresentamos na Figura 3.3 um grafo não direcionado que possui seis vértices representados pelas letras A, B, C, D, E e F e 8 arestas presentadas pelos números de 1 a 8. As Figuras 3.4, 3.5 e 3.6 são respectivamente a matriz de adjacência, lista de adjacência e o objeto do grafo representando as estruturas de dados que utilizamos no presente trabalho.

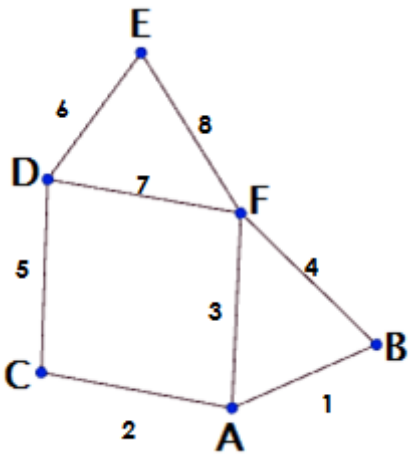


Figura 3.3: Grafo não direcionado.Fonte: Próprio autor

	A	B	C	D	E	F
A	0	1	1	0	0	1
B		0	0	0	0	1
C			0	1	0	0
D				0	1	1
E					0	1
F						0

Figura 3.4: Matriz de Adjacência.Fonte: Próprio autor

A	B	C	F	
B	A	F		
C	A	D		
D	C	E	F	
E	D	F		
F	A	B	D	E

Figura 3.5: Lista de adjacência.Fonte: Próprio autor

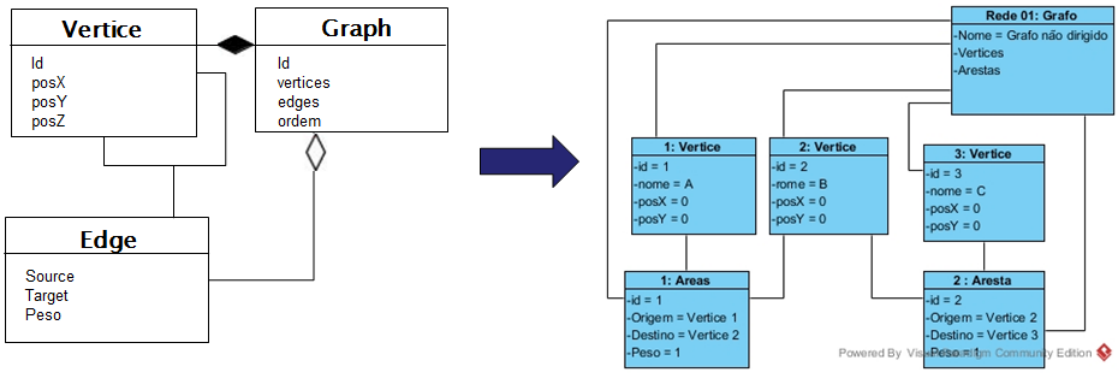


Figura 3.6: Objeto Grafo. Fonte: Próprio autor

3.3 Redes Sociais e Complexas

Uma rede pode ser conceituada como uma coleção de elementos que possuem conexão entre si. O estudo dos padrões existentes nas conexões entre os elementos que compõem uma rede pode gerar conhecimento a cerca do sistema em análise. Os estudos de redes

são aplicados em diversas áreas do conhecimento como física, biologia, ciências sociais e computação (Newman, 2010).

Redes sociais são aquelas onde a conexão entre os atores é formada por uma relação social, como alunos de um grupo de pesquisa, funcionários de uma empresa ou amigos de infância. Ao falar em redes sociais, naturalmente nos vem à mente as redes de relacionamento como *Facebook*, *Istagram*, *Twitter*, entre outras, porém o estudo das redes sociais vai além e a análise do comportamento e identificação dos fenômenos contidos são feitos por meio de métricas ou propriedades que buscam eliminar a subjetividade implícita nas redes sociais (Gabardo, 2015; Figueiredo, 2011).

Uma rede é classificada como complexa quando a sua estrutura tipológica apresenta um padrão não trivial, em comparação a modelos comuns. É considerada como um sistema complexo, dado a quantidade de elementos e interações existentes, bem como o número de propriedades que emergem, reforçando a característica da não trivialidade e auto-organização (Figueiredo, 2011).

Diante da diversidade de conceituações pertencentes às redes sociais e complexas na literatura científica, apresentamos a seguinte:

> Muitas redes sociais e complexas possuem propriedades não triviais, inexistentes em modelos mais simples. Essas redes são sistemas complexos que, em geral, envolvem inúmeros elementos organizados em estruturas que podem existir, ou coexistir, em diferentes escalas. Seus processos de ação e de organização não são usualmente descritos por regras simples ou redutíveis a apenas um nível explanatório. Frequentemente, suas características principais emergem de interações entre suas partes constituintes e não podem ser previstas a partir de uma compreensão isolada de cada uma destas partes (Pereira, 2013).

3.4 Algoritmos de Distribuição de Redes Sociais e Complexas

Os algoritmos de distribuição espacial de grafos dão suporte à análise de redes sociais e complexas a partir da construção de elementos que apoiam a inspeção visual do grafo que representa a rede. O leiaute do grafo deve destacar as informações relevantes, dado que o mesmo servirá de ferramenta analítica para visualização dos dados e comunicação dos eventos contidos na rede, de acordo o contexto da pesquisa (Brandes; Wagner, 2013).

A construção de um leiaute de grafo deve levar em consideração critérios estéticos que apoiem a exibição das informações contidas no grafo.

Segundo Purchase (1997), a construção de leiautes de grafos deve levar em consideração: a diminuição do número de curvas e cruzamento de arestas, ou seja quanto menor for o número de curvas e cruzamento de arestas maior será a compreensão da informação

contida no grafo; alto grau de simetria, já que vértices e arestas devem ser desenhados com tamanho uniforme e; distribuição uniforme dos vértices, considerando que os vértices devem ser distribuídos de maneira uniformemente na área disponível para o desenho do leiaute.

Fruchterman e Reingold (1991) adicionam aos critérios discutidos por Purchase (1997) a proximidade de vértices. Segundo Fruchterman e Reingold (1991) vértices que possuem conexão devem ser posicionados de forma próxima enquanto os sem conexão não devem ser posicionados tão próximos.

Wetherell e Shannon (1979) proproem três critérios específicos para desenho de árvores: vértices no mesmo nível na árvore devem ser posicionados na mesma linha, e os níveis subsequentes devem ser posicionados em linhas paralelas; um vértice filho esquerdo deve ser posicionado a esquerda do vértice pai e o filho direito à direita e o vértice pai deve ser posicionado de forma central as seus filhos.

A implementação dos critérios estéticos discutidas acima se da a partir do uso dos mesmos como regras para definição dos leiautes de grafo. Na Seção 3.4.2, apresentaremos algoritmos de distribuição de grafos onde poderão ser observados a existência de algoritmos de atração e repulsão que objetivam respectivamente aproximar e distanciar vértices de acordo à existência de conexões ou até mesmo algoritmos que organizam os vértices em camadas ou níveis de acordo com o seu relacionamento hierárquico.

3.4.1 Categorização de Algoritmos de Distribuição de Redes Sociais e Complexas

Existe grande variedade de algoritmos de leiaute de grafo, dado às aplicações que o grafo possui em áreas como matemática, física, engenharia, computação, entre outras. A categorização dos algoritmos de redes sociais e complexas, neste trabalho, se deu a partir de análises quantitativas baseadas em métodos de classificação, análise de agrupamento e verificação de referenciais teóricos conforme descrito a seguir.

3.4.1.1 Categorização por Análise de Agrupamento

A análise de agrupamento é um método de classificação com o objetivo de organizar objetos em grupos de acordo com as características que os mesmos possuem, identificando as similaridades ou dissimilaridades (Linden, 2009).

Para realizar a análise de agrupamento dos algoritmos de distribuição de redes sociais e complexas foi criada uma tabela contendo as características dos algoritmos, cujos valores foram colocados de forma binária onde 1 representa a existência da característica e 0 a ausência da mesma. Foram elencados 18 algoritmos e 9 características conforme apresentado na Tabela 3.2.

A definição dos atributos para criação da tabela de classificação levou em conta as regras utilizadas pelos algoritmos para posicionamento e organização dos vértices no leiaute e a restrição dos algoritmos quanto ao direcionamento das arestas. Deste modo, as características escolhidas foram as seguintes:

- Aleatório – identifica se os vértices foram posicionados de forma aleatória.
- Camadas – identifica se os vértices serão agrupados em camadas.
- Direção – restringe o algoritmo a grafos direcionados ou não direcionados.
- Distância Teórica – define se o posicionamento do vértices será dado por meio de cálculo de distância entre os vértices.
- Grupos – Identifica se os vértices serão reunidos em grupos.
- Relação de Adjacência – Define se o posicionamento dos vértices levará em consideração a relação de vizinhança entre vértices.
- Sistema Físico – define se o posicionamento do vértices será dado por meio de princípios da física.
- Sistema Trigonométrico – define se o posicionamento do vértices será dado por meio de princípios trigonométrico.
- Transformação Geométrica – define se o posicionamento do vértices será dado por meio de transformações geométricas nas posições atuais dos vértices.

Para representar os agrupamentos utilizaremos o dendrograma, por este apresentar uma árvore de classificação onde os objetos mais semelhantes são agrupados de forma recursiva em subgrupos até que não existam mais objetos a serem verificados e uma rede bimodal indicando os algoritmos e suas características.

Tabela 3.2: Algoritmos de distribuição de grafos organizados por características.

Algoritmo	Aleatório	Camadas	Grupos	Direção	Distância Teórica	Sistema Físico	Transf. Geometrica	Trigonometrico	Relação de Adjacencia
Aleatório	1	0	0	0	0	0	0	0	0
Arc Diagram	0	0	1	1	0	0	0	1	1
Árvore	0	1	1	1	0	0	0	0	1
Circular With Forces	0	0	0	0	0	1	0	1	1
Circular With Radial	0	0	1	0	0	0	0	1	0
ClockWise Rotate	0	0	0	0	0	0	1	1	0
Contraction	0	0	0	0	0	0	1	0	0
Counter ClockWise Rotate	0	0	0	0	0	0	1	1	0
Expansion	0	0	0	0	0	0	1	0	0
Force Atlas	0	0	0	0	0	1	0	0	1
Fruchterman Reingold	0	0	0	0	0	1	0	0	1
Layout Circular	0	0	0	0	0	0	0	1	0
Layout Kamada Kawai	0	0	0	0	1	0	0	0	1
Layout K-Core	0	1	1	0	0	0	0	1	0
Layout Star	0	0	0	0	0	0	0	1	0
Matriz de Adjacência	0	0	0	0	0	0	0	0	1
Sugiyama	0	1	1	1	0	0	0	0	1
Yifan Hu	0	0	0	0	0	1	0	0	1
Total	1	3	5	3	1	4	4	8	9

Fonte: Próprio autor

Criamos algoritmo específico para o software estatístico R com o objetivo de realizar a análise de agrupamento e criar o dendrograma. A seguir apresentamos o algoritmo utilizado para realizar a análise de agrupamento, o dendrograma resultante da análise de agrupamento e a rede bimodal originada da relação entre os algoritmos e suas características, respectivamente Algoritmo 1, Figuras 3.7 e 3.8 (R Core Team, 2016).

Algorithm 1 Algoritmo de Análise de Agrupamento

```
1: library(xlsx);
2: arquivo ← read.xlsx(file.choose(), 1);
3: distancia ← dist(arquivo[,-1], "euclidian");
4: arquivo[,1] ;
5: agrup ← hclust(distancia, "complete");
6: plot(agrup, main="Algoritmos Dist. de Grafos", cex=1, hang=1.5, ylab="Distâncias",
   xlab="Caracteristicas", labels=arquivo[,1]);
```

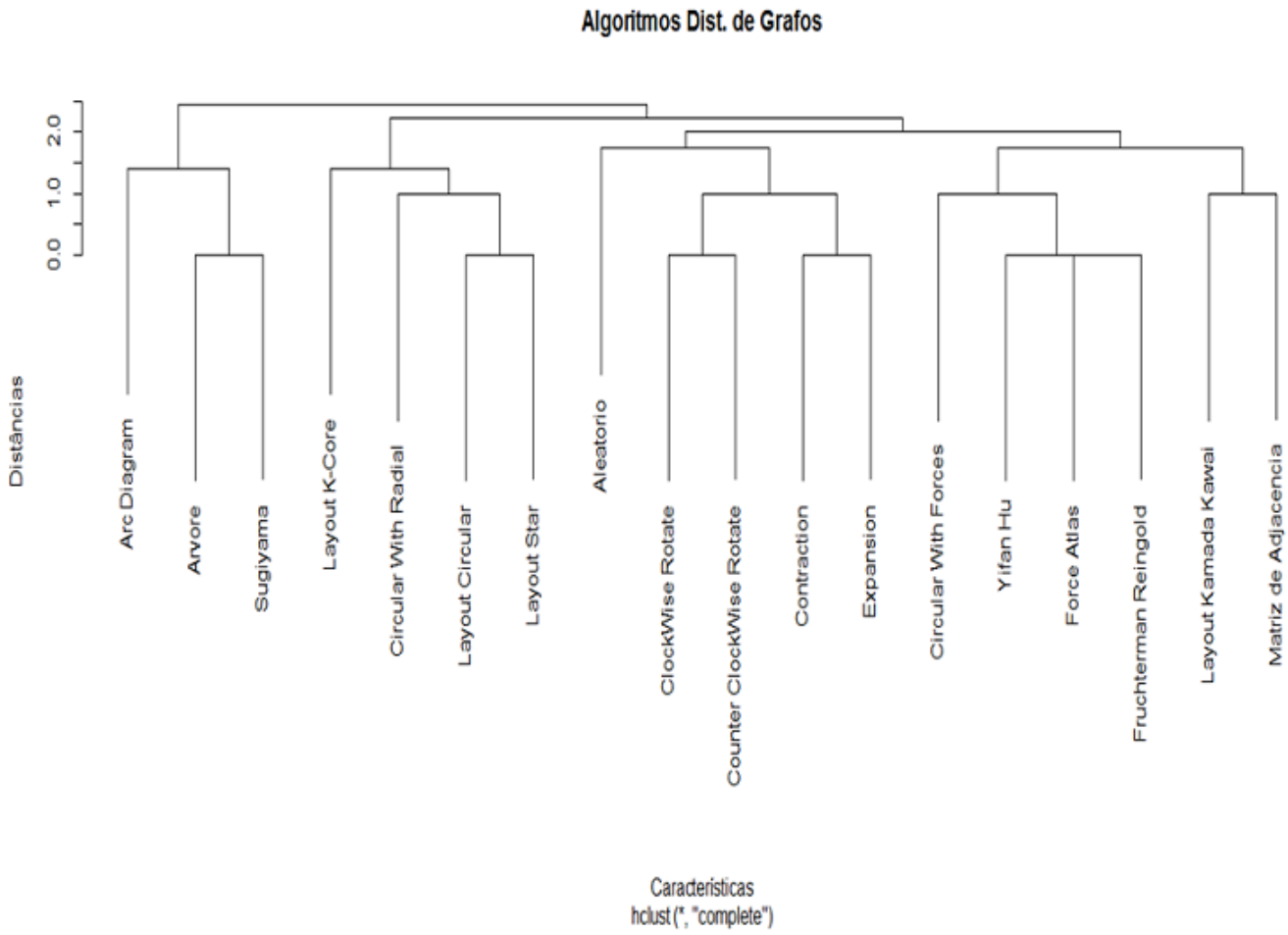


Figura 3.7: Dendrograma Algoritmos de Distribuição de Grafos. Fonte: Próprio autor

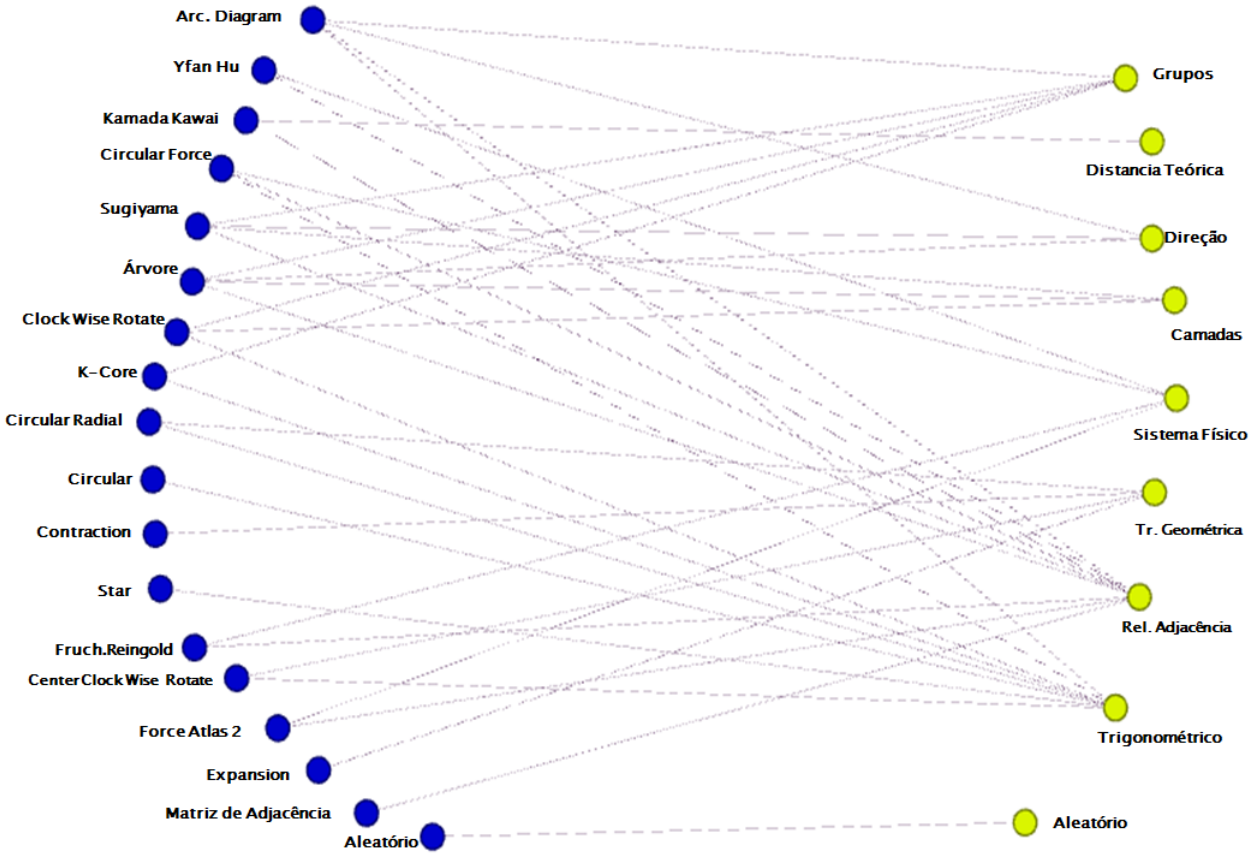


Figura 3.8: Rede Bimodal Algoritmos e Características. Fonte: Próprio autor

Observamos no dendrograma da Figura 3.7 que foram criados dois grandes grupos de algoritmos. O primeiro grupo contém os algoritmos *Arc Diagrama* e os algoritmos hierárquicos Árvores e *Sugiyama*. Apesar de, na literatura, encontrar o *Arc Diagrama* classificado como algoritmo circular, acreditamos que a análise de agrupamento o colocou junto com os algoritmos hierárquicos devido a estes possuírem em comum com os demais algoritmos do grupo, o atributo de restrição quanto ao uso para grafo direcionado e uso da relação de adjacência para posicionamento dos vértices.

O segundo grupo possui duas subdivisões. A primeira agrupa os algoritmos cujo posicionamento dos vértices se dá por princípios trigonométricos e também pela existência de grupos. A segunda subdivisão cria três subgrupos, o primeiro com os algoritmos que definem o posicionamento dos vértices a partir de transformações geométricas somando a operações trigonométricas (e.g. *ClockWise Rotate* e *Counter ClockWise Rotate*); o segundo subgrupo com os algoritmos que definem o posicionamento dos vértices apenas com o uso de transformações geométricas (e.g. *Contraction e Expansion*) e, por fim, o terceiro sub grupo com os algoritmos definidos na literatura como dirigidos por força, que possuem em comum a característica do uso da vizinhança dos vértices e divergem pelo fato

de utilizarem princípios físicos (e.g. *Force Atlas 2*, *Fruchterman Reingold* , *Circular With Forces* e *Yifan Hu*) e cálculo de distâncias teóricas (e.g. *Kamada Kawai*) para determinar o posicionamento final dos vértices.

Rede de dois modos são redes onde as entidades ou atores que possuem conexão entre si são de categorias diferentes (Tomaél; Marteleto, 2013). Apresentamos na Figura 3.8, rede de dois modos que apresenta conexões, quando existem, entre os algoritmo de distribuição espacial de redes e as características utilizadas pelos mesmos para posicionamento e organização dos vértices e arestas no leiaute da rede. Ao realizar inspeção visual na rede, foi possível observar a relação entre os algoritmos e suas características (e.g. algoritmo *Arc. Diagram* que se conecta as características Trigonométricas, Grupos, Direção e Relação de Adjacência) e também as entidades que não possuem conexões como o caso dos algoritmos de distribuição Aleatória e Matriz de Adjacência. Apesar de apresentar a relação entre os algoritmos e as características, a rede de dois modos não é uma ferramenta eficaz para definir a categorização dos algoritmos. Por este motivo optamos pelo uso da classificação por análise de agrupamentos.

A classificação por análise de agrupamento se assemelha a categorização dos algoritmos encontrada na literatura conforme apresentado a seguir.

3.4.1.2 Categorizações Propostas na Literatura

Nas obras *Handbook of Graph Drawing and Visualization*, editada por Tamassia (2013) e *Gephi Cookbook* de Khokhar (2015) são propostas categorizações de algoritmos levando em conta, respectivamente, conceitos de matemática discreta e análise de redes sociais e complexas.

Tamassia (2013) categoriza os algoritmos de distribuição de grafos em:

- Circular - classe de algoritmo onde os vértices são organizados em *clusters* e posicionados em um circunferência e as arestas são desenhadas como linhas retas (e.g Leiaute Circular, *Circular With Forces*, *Circular With Radial*), esta classe de algoritmos será melhor discutida na Seção 3.4.2.1 (Six; Tollis, 2013).
- *Force-Directed* - classe de algoritmo onde o posicionamento dos vértices é determinado utilizando somente informações contidas no grafo, desconsiderando questões relacionadas ao domínio de análise, são conhecidos também como incorporadores de mola e os leiautes produzidos são agradáveis e sem cruzamento de arestas (e.g. *Fruchterman e Reingold*, *Kamada e Kawai*), esta classe de algoritmos será melhor discutida na Seção 3.4.2.3 (Kobourov, 2013).

- *Hierarchical* - classe de algoritmo onde o posicionamento dos vértices deve representar o relacionamento hierárquico existente, restrito a grafos direcionados (Healy; Nikolov, 2013; Rusu, 2013) (e.g. *Sugiyama*, Árvore).

Khokhar (2015) discute o uso de algoritmos na ferramenta *Gephi*, mas não propõem de forma explicita classificação dos mesmos, porém é possível observar, a sugestão de agrupamento da classe algoritmos onde o posicionamento dos vértices e ajustado para facilitar análises, como algoritmos de transformação geométrica (e.g. *ClockWise Rotate*, *Contraction*, *Expansion*).

É possível verificar semelhanças nas categorizações por meio da análise de agrupamento e por meio da verificação do referencial teórico, apesar de, na literatura, algoritmos como *Circular With Forces* e *Circular With Radial* serem classificados como circulares enquanto na análise de agrupamento, respectivamente, estarem próximos a algoritmos dirigidos por força e hierárquicos. Para o desenvolvimento do presente trabalho, será utilizada a categorização por meio do referencial teórico por estas terem a indicação do uso dos algoritmos em análise de redes sociais e complexas.

3.4.2 Leiautes de Distribuição de Grafos

Discutiremos a seguir, os leiautes de distribuição de grafos usados para a proposição da *NET-UML*. Utilizamos como estrutura de dados as estruturas apresentadas na Seção 3.2.2, haja visto que todos os leiautes terão obrigatoriamente como parâmetro de entrada o objeto grafo. Os leiautes *Tree* e *Sugiyama* adicionalmente farão uso da matriz e lista de adjacência conforme visto a seguir.

3.4.2.1 Algoritmos Circulares

Os algoritmos circulares se caracterizam por dividir o grafo em *clusters*, posicionar os vértices na borda de uma circunferência e desenhar as arestas como linhas retas.

O posicionamento dos vértices é calculado antecipadamente sem a necessidade de realizar muitas iterações. Possuem como ponto negativo a possibilidade de apresentar elevando número de cruzamentos de arestas a medida que o grafo cresce em número de vértices e arestas. Dentre os diversos algoritmos circulares existentes, para a composição deste trabalho foram estudados os algoritmos: Circular Básico, Estrela e H-Core (Six; Tollis, 2013; Vilas Bôas, 2016; Alvarez-hamelin et al., 2005).

Leiaute Circular Básico

Neste leiaute os vértices são colocados em um formato circular cujo o posicionamento de cada um dos vértices é dado pelo cálculo do deslocamento em radianos dos ângulos atribuídos a cada um dos vértices. Seu uso é indicado para inspeção visual de grafos ou dígrafos que possuam pequeno número de vértices e arestas, pois a medida que a rede cresce em número de vértices e arestas a legibilidade do leiaute diminui dado o aumento do número de cruzamentos de arestas.

Este leiaute é útil para identificação de *clusters* existentes na rede e possui complexidade computacional $\theta(n)$ sendo n o número de vértices (Six; Tollis, 2013; Csardi; Nepusz, 2006). O pseudocódigo do algoritmo Circular Básico e seu detalhamento estão no Apêndice C, Seção C.1.

A Figura 3.9 exibe uma distribuição Circular Básica de uma rede hipotética com 10 vértices e 25 arestas onde é possível observar a formação de dois grupos de vértices, o grupo 1 formado pelos vértices a, b, c e d e o grupo 2 formado pelos vértices e, f, g, h, i e j.

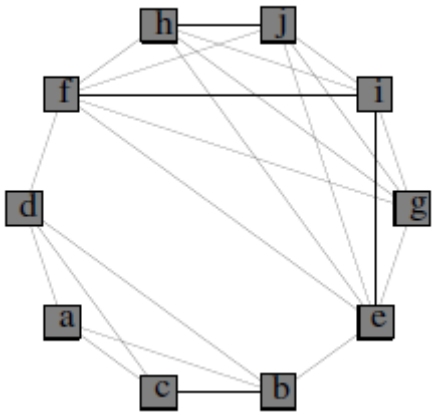


Figura 3.9: Distribuição Circular Básica de uma rede com 10 vértices e 25 aresta.Fonte: (Six; Tollis, 2013)

Leiaute Estrela

Neste leiaute, os vértices são desenhados em formato de estrela inscrita em uma circunferência. O vértice como maior grau será posicionado ao centro da circunferência enquanto os demais serão desenhados em um formato circular.

Como o leiaute discutido na seção anterior, o uso do leiaute estrela é indicado para grafos

e dígrafos que possuam poucos vértices e arestas, possuiu complexidade computacional $\theta(n)$ sendo n o número de vértices. Grafos que representam redes de citações podem utilizar este leiaute como forma de representação e identificação do vértice que possui a maior centralidade de grau. O pseudocódigo do algoritmo Estrela e seu detalhamento estão no Apêndice C, Seção C.2.

A Figura 3.10 exibe a distribuição estrela de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos. O vértice que possui maior grau é posicionado ao centro da circunferência.

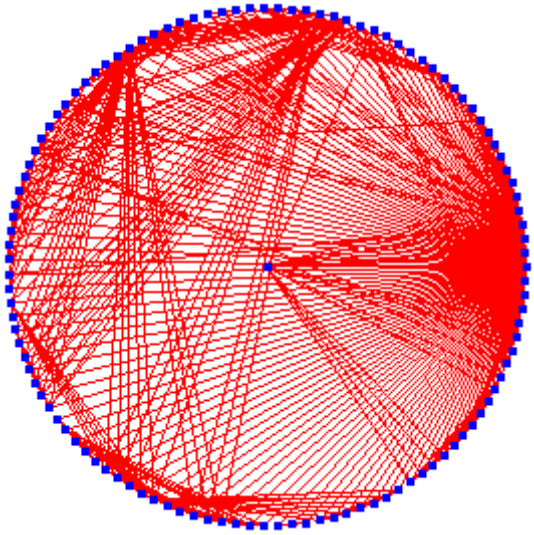


Figura 3.10: Distribuição Star de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

Leiaute *K-Core*

O algoritmo *K-Core* se caracteriza por organizar os vértices em camadas em formato de circunferências. Os vértices que possuem maior grau são posicionados nas camadas mais internas, próximas do núcleo central da circunferência enquanto os que possuem menor grau são posicionados nas camadas mais externas.

A inspeção visual deste leiaute ajuda a identificar os vértices que possuem maior centralidade de proximidade e, com isso, quais vértices possuem maior poder de transmitir informações na rede. Possui complexidade computacional $\theta(\mathrm{n}^2)$ sendo n o número de vértices. O pseudocódigo do algoritmo *K-Core* e seu detalhamento estão no Apêndice C, Seção C.3.

A Figura 3.11 exibe a distribuição *K-Core* de uma rede com 155 vértices extraída do

trabalho *K-Core Decomposition: A Tool For The Visualization of Large Scale Networks*, proposto por Alvarez-Hamelin et al. (2005).

Conforme a proposta do leiaute, os vértices que possuem os maiores números de conexões são agrupados no centro da circunferência enquanto os demais vão sendo colocados nas camadas mais externas.

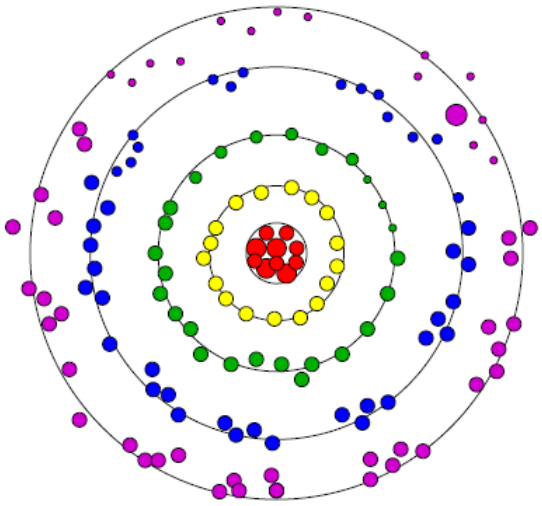


Figura 3.11: Distribuição K-Core de uma rede com 155 vértices.Fonte: (Alvarez-hamelin et al., 2005)

3.4.2.2 Algoritmos Hierárquicos

O estudo de dígrafos em muitos casos revela a existência de hierarquias e estas podem ser representadas por algoritmos que evidenciem a existência deste comportamento através do desenho de dígrafos por camadas ou níveis (Healy; Nikolov, 2013; Rusu, 2013).

O uso de dígrafos é muito útil para representar conexões como diagramas de relacionamento entre classes existentes na orientação a objetos, relacionamentos entre tabelas de bancos de dados e em análise de redes sociais onde a descoberta de hierarquia é importante (Healy; Nikolov, 2013). Dentre os algoritmos hierárquicos, utilizaremos para a composição deste trabalho o *Tree* e *Sugiyama*.

Leiaute *Tree*

Segundo Rusu (2013), árvores são estruturas de dados comumente utilizadas quando se deseja modelar relações hierárquicas, cuja a representação visual têm aplicações em áreas como engenharia de software (e.g. modelagem OO), gestão empresarial (e.g. organogra-

mas de empresas), inteligência artificial (e.g. representação do conhecimento), biologia (e.g. árvores evolutivas), dentre outras.

Eades et al. (2010) propuseram convenções que apoiam a criação de leiautes de árvores através da proposição das seguintes regras: desenho de polilinhas, as arestas são escritas em uma sequência conectada de segmentos de linha; desenho ortogonal, as arestas são escritas como segmentos horizontais e verticais; ordenação hierárquica de vértices, as folhas devem sempre ser escritas próximas e abaixo de seu vértice raiz; desenho planar, escrita de vértices e arestas de forma a evitar cruzamentos e; desenho em linha reta, as arestas devem ser escritas como linhas retas favorecendo assim a percepção das hierarquias.

Existe na literatura diversos tipos algoritmos de leiaute de distribuição de grafos em árvore, para a construção deste trabalho utilizamos o algoritmo proposto por Reingold e Tilford (1981) no artigo intitulado *Tidier Drawings of Trees*. Este algoritmo utiliza como princípios estéticos a escrita dos vértices de mesmo nível na mesma linha; os vértices filhos são posicionados em linhas paralelas próximos aos seus pais dando a ideia de subárvores, levando em consideração as regras descritas na seção anterior (Wetherell; Shannon, 1979; Narkhede; Inamdar, 2014; Reingold; Tilford, 1981).

O algoritmo proposto por Reingold e Tilford (1981) possui complexidade computacional $\theta(n\ log(n))$ com n sendo o número de arestas. No que tange redes sociais e complexas o uso deste algoritmo é indicado para análise de dígrafos com um número pequeno de vértices e arestas e em estudos onde se deseja investigar hierarquias, uma vez que a área disponível para o desenho da árvore é um limitador. O pseudocódigo do algoritmo de árvore proposto por Reingold e Tilford (1981) e seu detalhamento estão no Apêndice C, Seção C.4.

A Figura 3.12 exibe a distribuição *Tree* de uma dígrafo hipotético que possui 8 vértices e 7 arcos onde os vértices de mesmo nível são posicionados no mesmo eixo em relação ao eixo y e os vértices filhos são posicionados abaixo do vértice pai sempre da esquerda para direita afim de proporcionar ao final um leiaute agradável para inspeção visual e percepção da hierarquia.

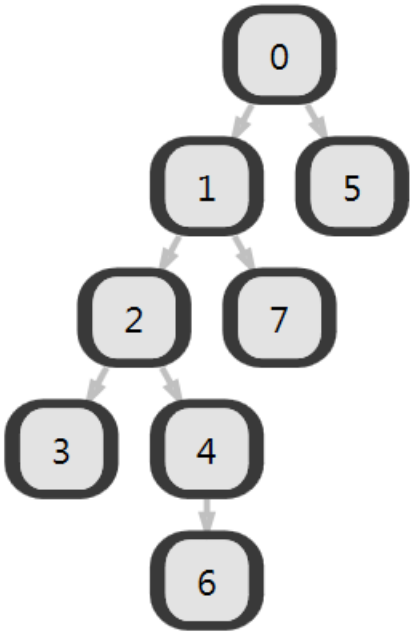


Figura 3.12: Distribuição *Tree* de uma rede com 8 vértices e 7 arestas.Fonte: Próprio autor

Leiaute *Sugiyama*

Sugiyama e Toda. (1980) **propuseram**, no trabalho intitulado *Methods for Visutal Understanding of Hierarchical System Structures*, algoritmo para distribuição espacial de dígrafos, a partir do uso de métodos teóricos e heurísticos que fossem capazes de posicionar os vértices a partir de uma ordenação em níveis, representando hierarquias, com poucos cruzamentos de arestas e posicionamento horizontal que prezasse pela simetria e com isso, resultasse em um leiaute agradável favorecendo assim, a inspeção visual.

Os métodos teóricos são responsáveis pela aderência do leiaute a representação das hierarquias, enquanto os métodos heurísticos se ocupam de garantir a performance do algoritmo.

Além dos usos gerais quando se fala em hierarquias, a representação visual proporcionada pelo Leiaute *Sugiyama* é indicada também para uso em modelagem estruturais como a Técnica de Avaliação e Revisão de Programas também conhecida como (PERT) uma vez, que se faz necessário representar a relação de precedência existente nas atividades de um projeto, neste cenário os vértices representam atividades e as arestas a relação entre as mesmas (Sugiyama; Toda., 1980).

O algoritmo proposto por Sugiyama e Toda. (1980) pode ser dividido em 4 etapas conforme descrevemos a seguir:

Na etapa 1 são removidos os ciclos e é criada uma estrutura hierárquica conforme a teoria dos grafos, neste trabalho utilizamos o algoritmo de Prim para criação de uma árvore geradora mínima conforme implementação proposta por Ziviane (2011). Caso a hierarquia obtida possua arestas longas, são criados vértices falsos como forma de equilibrar a hierarquia existente. Nesta etapa são atribuídos níveis a cada um dos vértices e por consequência a definição do posicionamento vertical dos mesmos; A etapa 2 se encarrega da redução do número de cruzamento de arestas a partir do uso de técnicas de ordenação, neste trabalho utilizamos a ordenação baricêntrica; A etapa 3 têm a responsabilidade de determinar o posicionamento horizontal dos vértices em linhas retas, com diminuição do comprimento de arestas quando houver conexão entre vértices de mesmo nível e; A etapa 4, com a função de eliminar os vértices falsos, criados na etapa 1, e recriar as arestas que os conectam, ajustando o posicionamento final dos elementos do dígrafo.

Este algoritmo possui complexidade computacional $theta(n.m\ log(m))$, sendo n é número de vértices e m é o número de arestas. O pseudocódigo do algoritmo *Sugiyama* e seu detalhamento estão no Apêndice C, Seção C.5.

A Figura 3.13 exibe a distribuição *Sugiyama* de uma dígrafo hipotético, que possui 15 vértices e 21 arcos, apresentado por Sugiyama e Toda. (1980) no trabalho *Methods for Visutal Understanding of Hierarchical System Structures* onde é possível observar o posicionamento dos vértices em camadas com pequeno número de cruzamento de arcos, obtido a partir do uso da ordenação, conforme passos do algoritmo *Sugiyama* discutido na seção anterior.

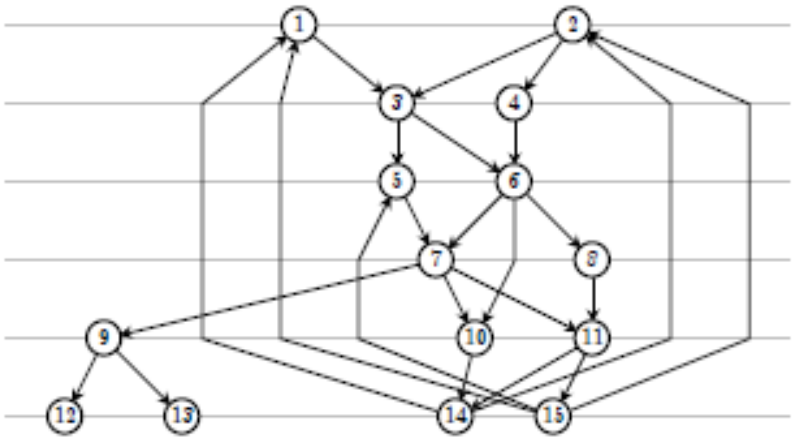


Figura 3.13: Distribuição *Sugiyama* de uma rede com 15 vértices e 21 arcos.Fonte: (Sugiyama; Toda., 1980)

3.4.2.3 Algoritmos Dirigidos por Força

Algoritmos dirigidos por força se caracterizam por utilizar princípios físicos e propriedades do próprio grafo para calcular a distribuição dos vértices no leiaute (Kobourov, 2013).

O seu uso é indicado para grafos não direcionados e os leiautes resultantes tendem a ser mais agradáveis para a inspeção visual, devido os mesmos prezarem pela simetria e diminuição do cruzamento de arestas. Dentre os algoritmos dirigido por força discutiremos neste trabalho os algoritmos *Force Atlas 2*, *Kamada Kawai* e *Fruchterman Reingold* (Kobourov, 2013; Jacomy et al., 2014; Fruchterman; Reingold, 1991; Eades et al., 2010).

Leiaute *Kamada Kawai*

Kamada e Kawai (1989) propuseram no artigo intitulado *An Algorithm For Drawing General Undirected Graphs* um algoritmo de leiaute para grafos não direcionados que utiliza os princípios da lei de *Hooke*, no qual o posicionamento dos vértices é obtido por meio do cálculo da distância Euclidiana entre os vértices através da introdução de um sistema dinâmico virtual, onde as conexões entre os vértices são representadas por molas que definem a posição final dos vértices quando a energia do sistema é considerada mínima.

A inspeção visual de grafos gerada pelo *Kamada Kawai* possibilita a identificação de arranjos de vértices que podem sugerir a existência de comunidades bem como os hubs existentes na rede.

A distribuição dos vértices no *Kamada Kawai* é realizada a partir da execução dos seguintes passos: definição do posicionamento inicial dos vértices de forma aleatória e cálculo das distâncias geodésica entre os vértices; definição da distância ideal (L) entre os vértices, obtida levando em conta a área disponível e o diâmetro da rede e; realização de cálculo do posicionamento dos vértices a partir da minimização da energia existente entre os mesmos adicionada pelo sistema dinâmico virtual que coloca os vértices em posições estáveis através do uso do método *Newton-Raphson*.

O *Kamada Kawai* possui complexidade computacional $\theta(n^2)$ sendo n o número de vértices (Csardi; Nepusz, 2006). O pseudocódigo do algoritmo *Kamada Kawai* e seu detalhamento estão no Apêndice C, Seção C.6.

A Figura 3.14 exibe a distribuição *Kamada Kamai* de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos.

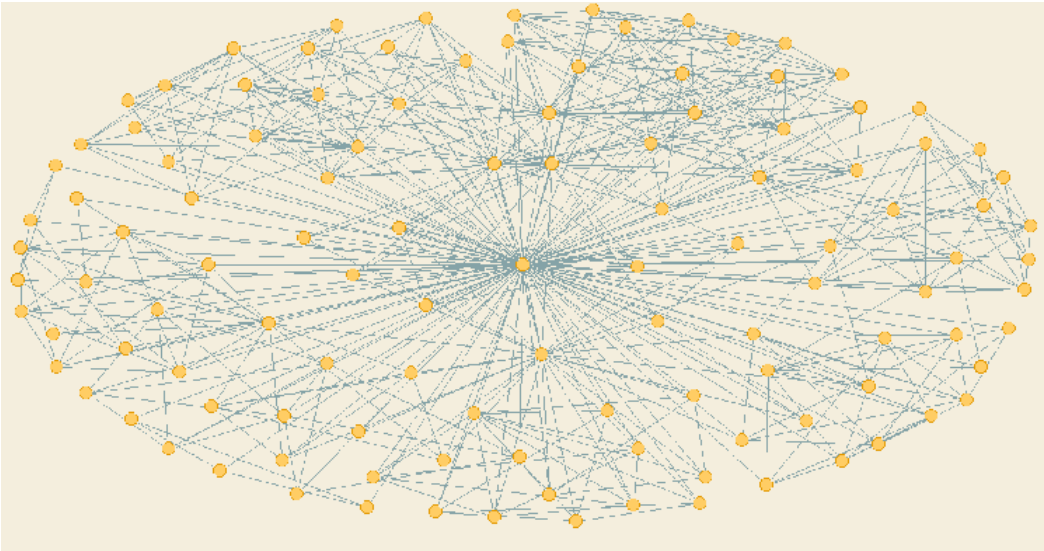


Figura 3.14: Distribuição *Kamada Kawai* de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

Leiaute *Fruchterman Reingold*

Fruchterman e Reingold (1991) propuseram um algoritmo para grafos não dirigidos onde os vértices conectados devem ser posicionados próximos uns dos outros, como forma de evitar o cruzamento de arestas, e os vértices devem estar distribuídos uniformemente afim de apresentar um leiaute agradável para a realização da inspeção visual.

O posicionamento dos vértices no leiaute *Fruchterman Reingold* se dá através de estabelecimento de sistema físico criado a partir de cálculos de repulsão entre os vértices não adjacentes e atração de vértices vizinhos. Como ocorre no leiaute *Kamada Kawai*, o posicionamento do inicial dos vértices e dado de forma aleatória.

O cálculo da força de repulsão entre os vértices é dado pela fórmula ($F_r = k/d$) enquanto a força de atração entre os vértices é obtida por meio da formula ($F_a = -k * d^2$) onde k é a constante da força e d representa a distância entre os vértices de acordo a Equação 3.6.)

$$d = \sqrt{(dx^2 + dy^2)} \tag{3.6}$$

Após os cálculos de repulsão e atração, é efetuada verificação das máximas distâncias entre os vértices para atribuição do posicionamento dos elementos no leiaute do grafo.

Apesar de ser indicado para uso em grafos, este algoritmo pode também ser utilizado em dígrafos, sendo indicado para grafos que possuam entre 1 a 1 000 000 vértices, vale ressaltar que a sua performance de execução está relacionada ao tamanho do grafo. O

Fruchterman Reingold possui complexidade computacional $\theta(n^2)$ a cada iteração sendo n o número de vértices (Csardi; Nepusz, 2006). O pseudocódigo do algoritmo e seu detalhamento estão no Apêndice C, Seção C.7.

A Figura 3.15 exibe a distribuição *Fruchterman Reingold* de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos.

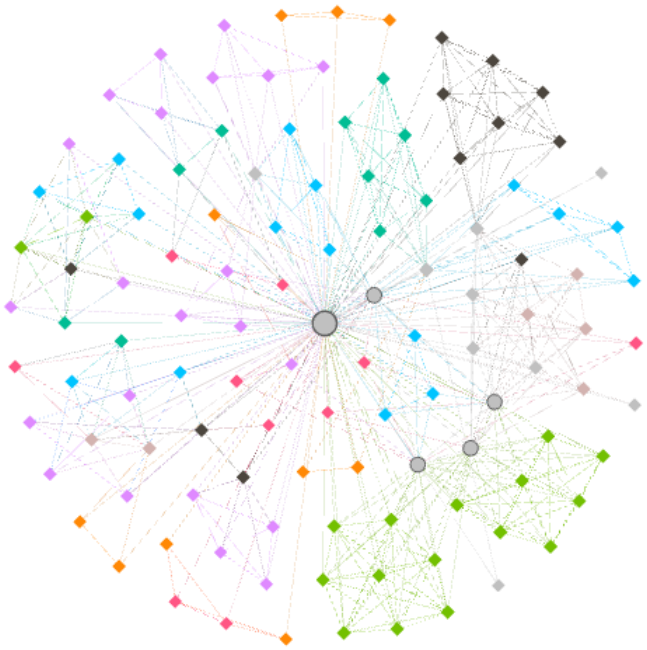


Figura 3.15: Distribuição *Fruchterman Reingold* de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

Force Atlas 2

Jacomy et al. (2014) propuseram algoritmo dirigido por força que, como os algoritmos *Kamada-Kawai* e *Fruchterman Reingold*, simula sistema físico para calcular a distribuição dos vértices de um grafo. Neste algoritmo os vértices se repelem como partículas carregadas enquanto as arestas se ocupam de atrair os vértices simulando um sistema de molas.

O *Force Atlas 2* prioriza a qualidade do leiaute de distribuição do grafo em detrimento da performance de execução, sendo classificado como um algoritmo continuo, pois dependente do usuário para que a sua execução seja finalizada, esta característica o diferencia dos algoritmos *Kamada-Kawai* e *Fruchterman Reingold* pois nestes, as forças de atração e repulsão são aplicadas de forma não homogênea.

Ainda segundo Jacomy et al. (2014), Fruchterman e Reingold (1991) não reproduziu fiel-

mente o modelo mola-elétrico, que utiliza para repulsão de partículas carregas a formula ($F_r = k/d^2$) e para atração ($F_a = kd$). Conforme apresentado nos algoritmo 16 e 17, Fruchterman e Reingold (1991) utilizou respectivamente para efetuar os cálculos de repulsão e atração dos vértices as fórmulas ($F_r = k/d$) e ($F_a = -k * d^2$, evidenciando assim, que apesar de utilizar a abstração de um sistema de molas não representou fielmente tal sistema.

Segundo Noack (2007) e Jacomy et al. (2014), o cálculo da distância entre os vértices é a característica mais importante dentre os leiautes dirigidos por força e têm grande peso na espacialização dos vértices, haja visto a dependência que os sistemas físicos possuem deste cálculo para determinarem as forças de atração e repulsão. Esta dependência pode ser linear, exponencial e logarítmica. Ao levar esta questão em consideração o *Force Atlas 2* permite ao pesquisador fazer uso de diferentes tipos de cálculos de força de atração e repulsão dos vértices a partir da seleção de parâmetros. Esta tipo de algoritmo possui complexidade computacional $\theta(n * log(n))$ sendo n o número de vértices (Jacomy et al., 2014). O pseudocódigo do algoritmo *Force Atlas 2* e seu detalhamento estão no Apêndice C, Seção C.8.

A Figura 3.16 exibe a distribuição *Force Atlas 2* de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos.

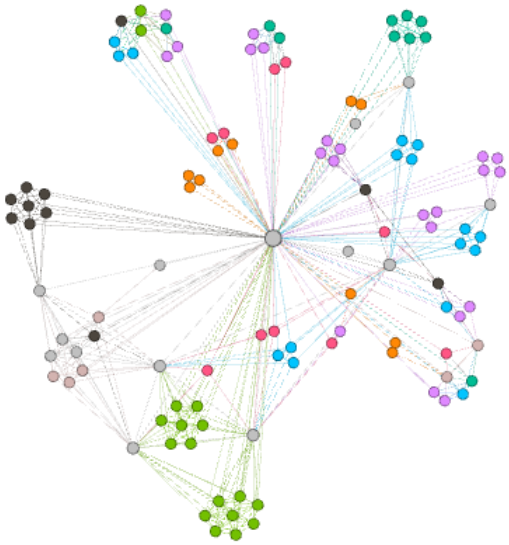


Figura 3.16: Distribuição *Force Atlas 2* de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

3.4.2.4 Algoritmos Transformação Geométrica

Esta classe de algoritmos se ocupa de aplicar transformação geométrica em um grafo ou dígrafo, onde prioritariamente, já foi aplicado outra categoria de leiaute de distribuição.

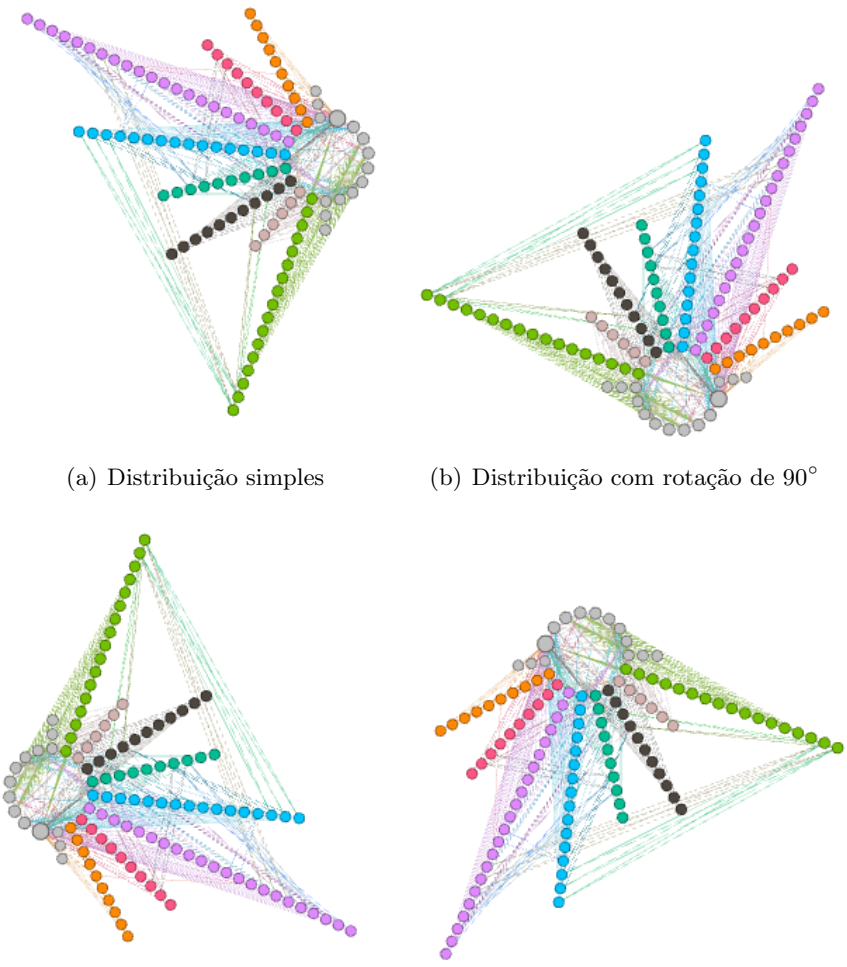
A aplicação dos algoritmos de transformação geométrica de contração e expansão é diferente das operações de zoom, pois estas se ocupam de aumentar ou diminuir o tamanho dos vértices e arestas dando a sensação de proximidade ou distanciamento enquanto os algoritmos de transformação geométrica criam uma nova distribuição especial dos grafos. Neste trabalho, discutiremos os algoritmos *Clockwise Rotate*, *Center Clockwise Rotate*, *Contraction e Expansion* (Khokhar, 2015).

Leiautes *Clockwise Rotate e Center Clockwise Rotate*

Os algoritmos *Clockwise Rotate e Center Clockwise Rotate* aplicam transformação geométrica no posicionamento dos vértices a partir de um ângulo informado, realizando respectivamente, operações de rotação no sentido horário e anti-horário.

A rotação no sentido horário é obtida ao se atribuir um ângulo com valor positivo na formula que faz o cálculo do posicionamento do vértice, enquanto as rotações no sentido anti-horário são obtidas ao se atribuir um ângulo de rotação com valor negativo (Khokhar, 2015). Esta tipo de algoritmo possui complexidade computacional $\theta(n)$ sendo n o número de vértices. O pseudocódigo do algoritmo *Clockwise Rotate e Center Clockwise Rotate* e seu detalhamento estão no Apêndice C, Seção C.9.

A Figura 3.17 exibe a distribuição *Clockwise Rotate* de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos. Inicialmente aplicamos o leiaute de distribuição *Radial Axis* e em seguida utilizamos a distribuição *Clockwise Rotate* para rotacionar os elementos do grafo em 90°, 180° e 270°.



(a) Distribuição simples (b) Distribuição com rotação de 90°

(c) Distribuição com rotação de 180° (d) Distribuição com rotação de 270°

Figura 3.17: Distribuição *Clockwise Rotate* de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

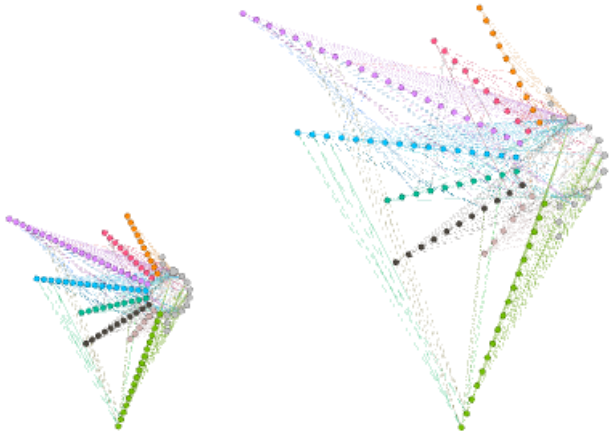
Leiautes *Contraction e Expansion*

Os algoritmos de contração e expansão aplicam transformação geométrica nos posicionamentos dos vértices do grafo aproximando ou distanciando o vértice do centro do plano onde os mesmo foram posicionados a partir de uma escala informada, realizando movimento diagonal dando a impressão de expansão ou retração da rede (Vilas Bôas, 2016), (Khokhar, 2015).

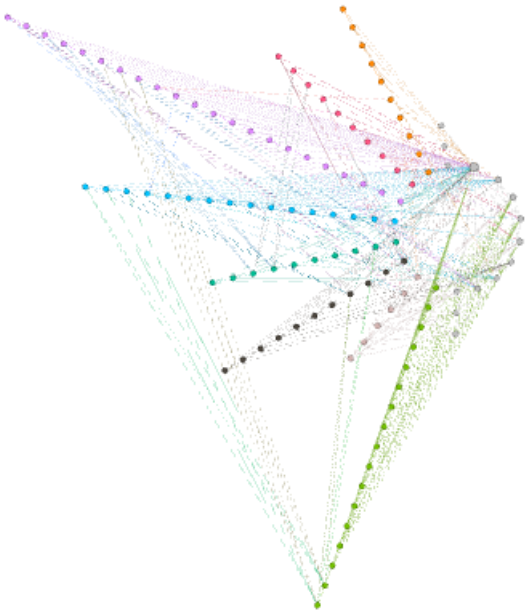
O algoritmo de contração normalmente é utilizado quando se possui uma rede muito esparsa e se deseja visualizar a mesma em uma única janela. Já o algoritmo de expansão é indicado quando se possui uma rede densa e se deseja analisar a rede de forma esparsa. Vale ressaltar que os algoritmos de expansão se diferem das operações de ampliação, pois enquanto estas diminuem ou aumenta os elementos do grafo de forma proporcional, nos leiautes de contração e repulsão, os tamanhos dos elementos do grafo se mantém de

forma inalterada. Somente o posicionamento dos elementos sofre alteração (Khokhar, 2015). O pseudocódigo do algoritmo *Contraction e Expansion* e seu detalhamento estão no Apêndice C, Seção C.10.

A Figura 3.18 exibe a distribuição *Expansion* de uma rede de coautoria de publicações de pesquisadores de uma programa de pós graduação interdisciplinar em eventos científicos e periódicos. Inicialmente aplicamos o leiaute de distribuição *Radial Axis* e em seguida utilizamos a distribuição *Expansion* para aumentar a distância entre os elementos do grafo como forma de apoiar a inspeção visual e descoberta de informações contidas no mesmo.



(a) Distribuição simples (b) Distribuição com fator de expansão 1.2



(c) Distribuição com fator de expansão 2.4

Figura 3.18: Distribuição Expansion de uma rede de coautoria de publicações em eventos e periódicos com 116 vértices e 483 arestas.Fonte: Próprio autor

3.4.3 Algoritmos de Distribuição de Grafo e Aplicações

A escolha do algoritmo de distribuição espacial do grafo depende do tipo de informação e das características da rede que está sendo analisada. Apresentamos a seguir tabela indicativa de cenários de uso dos algoritmos de visualização de rede discutidos neste trabalho.

Tabela 3.3: Algoritmos de Distribuição de Grafo e Aplicações.

Algoritmo	Tipo do grafo	Complexidade Computacional	Tamanho da Rede
Ciruclar *Layout*	Não direcionado	$\theta(n)$	1 a 1 000 000 arestas
Star Layout	Não direcionado	$\theta(n)$	1 a 1 000 000 arestas
K-Core Layout	Não direcionado	$\theta(n^2)$	1 a 1 000 000 arestas
Tree Layout	Direcionado	$\theta(n\ log(n))$	Indefinido
Sugiyama Layout	Direcionado	$\theta(n.m\ log(m))$	Indefinido
ClockWise Rotate	Direcionado e Não direcionado	$\theta(n)$	Indefinido
Contraction	Direcionado e Não direcionado	$\theta(n)$	Indefinido
Counter ClockWise Rotate	Direcionado e Não direcionado	$\theta(n)$	Indefinido
Expansion	Direcionado e Não direcionado	$\theta(n)$	Indefinido
Force Atlas 2 Layout	Não direcionado	$\theta(n*log(n))$	1 a 1 000 000 vértices
Fruchterman Reingold Layout	Não direcionado	$\theta(n^2)$	1 a 1 000 000 vértices
Kamada Kawai Layout	Não direcionado	$\theta(n^2)$	1 a 1 000 000 vértices

Fonte: Próprio autor

3.5 Considerações do Capítulo

Apresentamos neste capítulo conceitos relacionados a teoria dos grafos, redes sociais e complexas, algoritmos de distribuição de grafos e categorizações dos mesmos com o objetivo de observar a sintaxe e semântica relacionada, bem como, propriedades comuns e especificas que cada elemento estudado possui.

No próximo capítulo apresentaremos um proposição de extensão da UML para o desenvolvimento de ferramentas que possuam funcionalidade de visualização de redes sociais e complexas, levando em consideração a semântica e a sintaxe envolvidas neste domínio de aplicação.

Capítulo Quatro

Linguagem de Modelagem de Ferramentas de Visualização de Redes Sociais e Complexas

O paradigma da orientação a objetos, também conhecido como OO, é amplamente utilizado no processo de desenvolvimento de software. Nesse paradigma o processo a ser informatizado é tratado como uma coleção de objetos que se relacionam e por meio de troca de mensagens realiza os objetivos do sistema.

O uso da OO traz como vantagens elevado grau de abstração, encapsulamento e reutilização de objetos entre os projetos, proporcionando a criação de modelos de software próximos do mundo real pois, é possível transportar para os modelos aspectos semânticos do universo a ser informatizado (Wazlawick, 2011).

Ferramentas especialistas em distribuição, manipulação e análise de grafos originados por redes sociais e complexas foram desenvolvidas com base no paradigma OO, a exemplo do software *Gephi* e biblioteca *Graph Sharp*.

Existem vários tipos de ciclo de vida para construção de software OO (e.g iterativo e incremental, cascata, ágil, programação extrema) e fases clássicas que são comuns como: análise, projeto de software, implementação e testes.

Este trabalho discutirá questões relacionadas às fases e artefatos de software OO relacionados aos aspectos conceituais e semânticos: fase de análise com o diagrama de classes e aspectos sintáticos e comportamentais; fase de especificação de requisitos com os casos de uso e; fase de projeto de software com os diagramas de máquinas de estados, uma vez, que estas fases se ocupam respectivamente com aspectos conceituais e semânticos do domínio do problema que o software se propõe a solucionar.

O objetivo deste capítulo é apresentar uma proposta de extensão da linguagem UML, denominado *NET-UML*, para uso na criação de modelos de análise de software de criação e visualização de grafos de redes sociais e complexas.

4.1 NET-UML

A *NET-UML* têm como proposta contribuir, na fase de análise, com a adição de elementos que abstraem a semântica e a sintaxe existente no domínio das redes sociais e complexas

e nos leiautes de distribuição de grafos que as representam. Com isso, pretendemos fazer com que os modelos de domínio sejam mais ricos e traduzam de fato o que os objetos do mundo real representam, diminuindo assim, os riscos de uma análise que não seja fiel ao contexto que a solução computacional está inserida.

A motivação pela proposição da *NET-UML* foi o potencial de desenvolvimento de novas ferramentas para análise e inspeção visual de redes sociais e complexas a partir da OO.

4.1.1 Metodologia para Construção da NET-UML

A construção da *NET-UML* se deu a partir da junção de aspectos sintáticos e semânticos relacionados a teoria dos grafos e leiautes de distribuição comumente utilizados para análise de redes sociais e complexas, conforme visto em Brandes e Wagner (2013), Csardi e Nepusz (2006), Six e Tollis (2013), Khokhar (2015), e apresentado no Capítulo 3, Seção 3.4, com os aspectos relacionados análise OO cobertos pela UML e pelo modelo GEO-OMT, utilizado para construção de sistemas de informação geográficas (SIG), respectivamente apresentados nos Apêndices A e B.

Uma vez definido como domínio de aplicações computacional, as redes sociais e complexas, no que tange especificamente as ferramentas de visualização e inspeção visual de redes, dividimos os elementos do modelo em estruturais, responsável pela representação sintática e conceitual do domínio das redes sociais, conforme apresentamos na Seção 4.1.5.1 do presente Capítulo e elementos comportamentais, responsável pela semântica associada a distribuição espacial dos grafos e comportamentos ou funcionalidades que ferramentas computacionais deste domínio devem implementar de forma mandatória, conforme apresentamos nas Seções 4.1.3, 4.1.4, 4.1.5.2 e 4.1.6.

Após elencar os elementos do modelo seus atributos, comportamento e relacionamentos foram definidos esteriótipos, através de representações pictográficas, para os elementos do tipo classe como forma de destacar a sintaxe e semântica associada ao elemento, conforme apresentamos na Seção 4.1.2.

Apresentamos na Figura 4.1 os passos para construção do modelo NET-UML e seus componentes que serão melhor discutido nas seguintes seções.

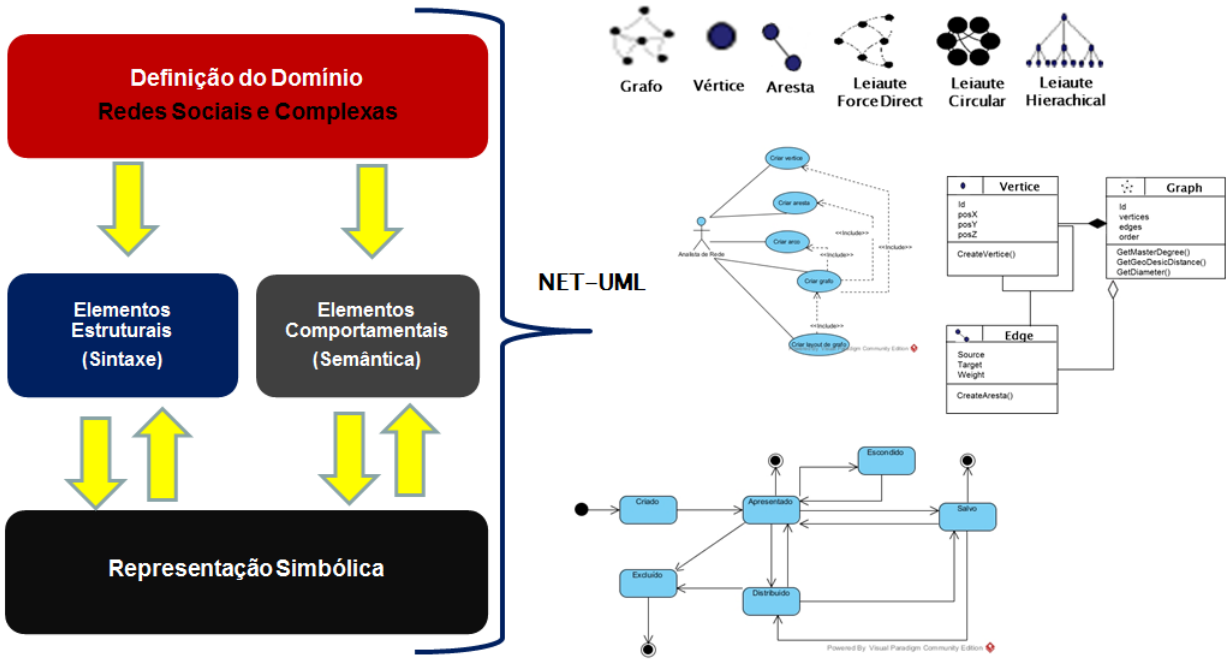


Figura 4.1: Metodologia para Construção da *NET-UML*. Fonte: Próprio autor

4.1.2 *Componentes da NET-UML*

A *NET-UML* possui como características:

- Aderência ao paradigma OO utilizando os conceitos de classe, herança, objetos e relacionamentos.
- Individualização dos objetos a partir da definição de representação simbólica especifica, potencializada pelo uso de esteriótipos, mecanismo de extensão da UML.
- Visão contextualizada e segmentada do modelo do software a partir, da separação das classes convencionais, das classes do domínio da teoria e visualização dos grafos, destacando os relacionamentos que podem ocorrer entre estes diferentes objetos.
- Sugere comportamentos padrões e mandatórios para os elementos da teoria dos grafos, através da proposição explicita de casos de uso e diagramas de máquina de estados o domino das aplicações das redes sociais e complexas.

A *NET-UML* especializa a estrutura das classes destacando no lado superior esquerdo, um esteriótipo que ao invés de utilizar o simbologia convencional (e.g. *<<Document>>*,

<<*Control*>>, <<*Entity*>>), utiliza pictogramas que representam os objetos no contexto sintático e semântico da teoria dos grafos, mantendo os demais componentes do elemento classe que são nome, atributos e métodos.

O uso da representação pictográfica foi inspirado no modelo *GEO-OMT*. Este modelo têm por objetivo apoiar na percepção da natureza conceitual e comportamental do elemento a ser modelado, facilitando as tarefas de abstração e análise.

Apresentamos na Figura 4.2, proposição de representação do elemento classe no modelo *NET-UML*.

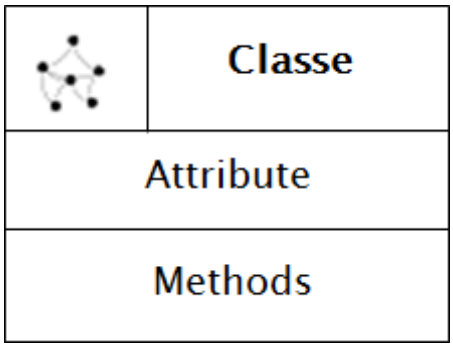


Figura 4.2: Elemento classe no modelo *NET-UML*. Fonte: Próprio autor

A *NET-UML* é composta por dois grupos de classes:

- *Conventional Class* - representa os objetos básicos de uma software desenvolvido a partir da OO e que estão fora do domínio conceitual das redes.
- *Network Class* - representa os objetos que estão no domínio sintático e semântico das redes sendo subdivida em *Object Graph* e *Object Layout*, respectivamente, agrupando as classes que representam o elemento matemático grafo e as classes que representam os leiautes de distribuição de grafos.

Apresentamos na Figura 4.3 o meta modelo da *NET-UML* e suas divisões e seus relacionamentos.

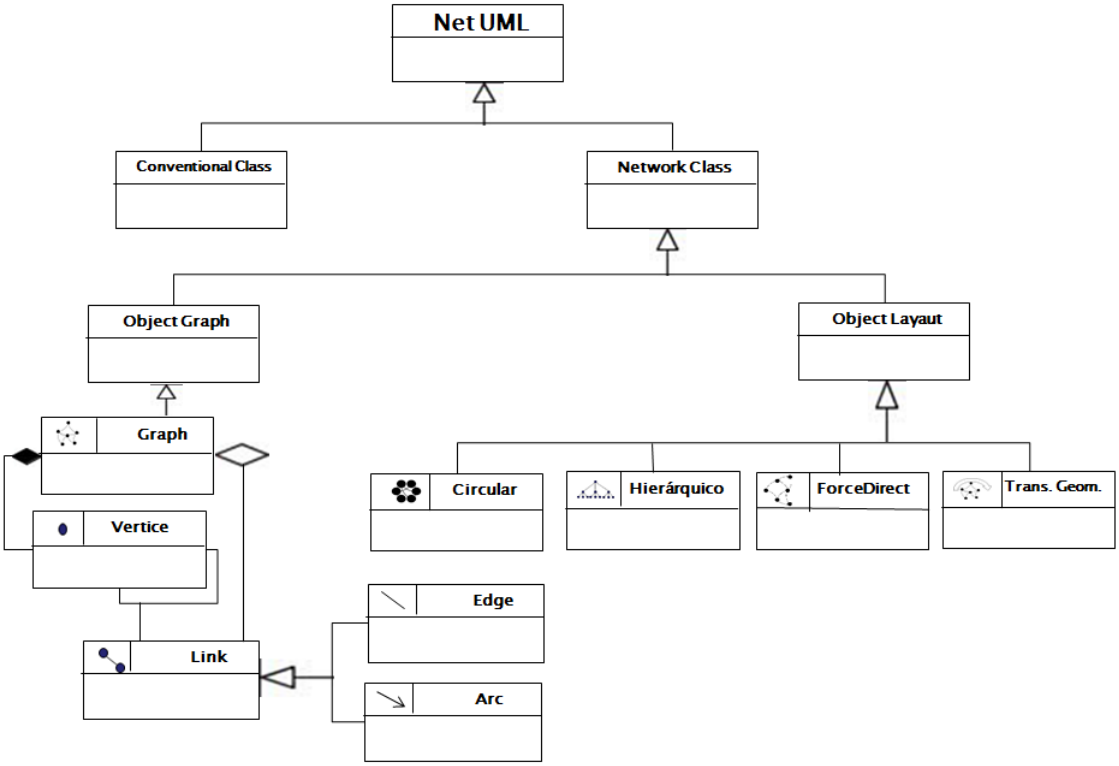


Figura 4.3: Meta modelo da *NET-UML*. Fonte: Próprio autor

4.1.3 Casos de Usos Mandatórios

Casos de uso são narrativas textuais utilizadas para descrever: as funções do sistema; suas delimitações; atores responsáveis e; o fluxo de operações necessárias para a execução (Larman, 2007; Tacla, 2007; Wazlawick, 2011). Possuem características comportamentais auxiliando assim a definição da semântica dos elementos da *NET-UML*.

A *NET-UML* reafirma a necessidade da existência de funcionalidades de criação de grafos e seus elementos, de acordo com o conceito do domínio das redes, definido como casos de usos obrigatórios a serem implementados a criação de vértices, arestas, arcos, grafos e leiautes de distribuição de grafos.

Apresentamos na Figura 4.4 o diagrama de casos de uso padrão da *NET-UML*.

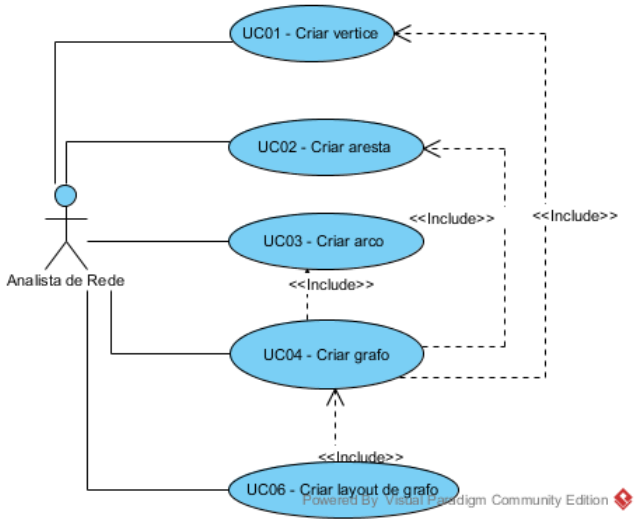


Figura 4.4: Diagramas de Casos de Uso. Fonte: Próprio autor

4.1.4 Diagrama de Máquina de Estados Mandatórios

A *NET-UML* sugere a existência dos diagramas de máquina de estado para os objetos vértices e grafo, haja visto, que estes são elementos com maior complexidade, no contexto das aplicações de visualização de redes sociais e complexas, cujo o seu comportamento na aplicação deve acontecer de forma a apoiar à inspeção visual.

Após análise dos algoritmos de distribuição espacial de grafos e ferramentas de análise e visualização de redes discutidos neste trabalho, identificamos os estados descritos a seguir para o objeto grafo conforme apresentamos na Figura 4.5:

- Criado - estado primário de um grafo cuja os vértices e arestas/arcos (caso existam) foram definidas.
- Apresentado - estado que exibe os elementos do grafo no leiaute podendo ocorrer primariamente após o estado de Criado. Uma vez apresentado o grafo pode ser Escondido, Salvo, Distribuído ou ter seu ciclo de vida encerrado.
- Escondido - uma vez apresentado o grafo pode ser escondido ou colocado como invisível e novamente ser voltar ao estado de Apresentado.
- Distribuído - estado que representa a aplicação dos algoritmos de distribuição espacial e manipulação dos elementos do grafo. Ema vez Distribuído o grafo volta a ao estado de Apresentado podendo ir para o estado de Salvo ou Excluído.
- Salvo - neste estado as propriedades do grafo são gravadas para continuidade da

inspeção visual. Normalmente este estado sucede as operações de manipulação e distribuição de um grafo precedendo a finalização do ciclo de vida do objeto.

- Excluído - após ser apresentado ou distribuído o grafo pode ser excluído do ambiente de visualização e com isso ter seu ciclo de vida encerrado.

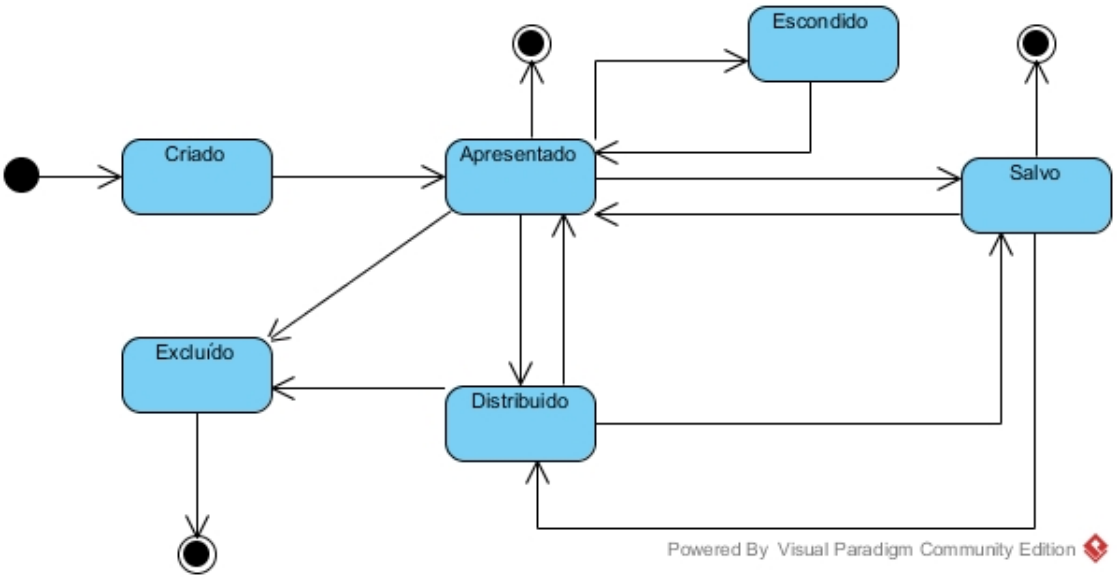


Figura 4.5: Diagrama de máquina de estados padrão para o objeto grafo de softwares de visualização de redes sociais e complexas. Fonte: Próprio autor

Conforme discutido em seções anteriores, o vértice é o único objeto da *NET-UML* que possui representação espacial, resultante da aplicação dos leiautes de distribuição de grafos, por isso possui atributos que definem seu posicionamento no plano (e.g. posição em x, y e z), o posicionamento das arestas e arcos é dado pela conexão dos vértices que as relacionam. O objeto vértice possui os seguintes estados conforme a Figura 4.6:

- Criado - estado primário de um vértice obtido após a criação da sua instancia.
- Apresentado - estado que o vértice alcança após a execução de um algoritmo de distribuição. Após apresentado um vértice pode ser escondido ou excluído logicamente e novamente voltar a ser exibido.
- Escondido - uma vez apresentado o vértice pode ser colocado como invisível, caso se deseje exibir apenas um conjunto de vértices (e.g. destacar um dado vértice e seus vizinhos e esconder os demais). Um vértice escondido pode voltar a ser exibido.
- Excluído Logicamente - estado que representa a exclusão parcial de um vértice do grafo porém, o mesmo pode voltar a fazer parte do grafo e com isso novamente ser apresentado.

- Excluído Fisicamente - estado que representa a exclusão definitiva de um vértice do grafo e com isso a finalização do ciclo de vida do objeto.

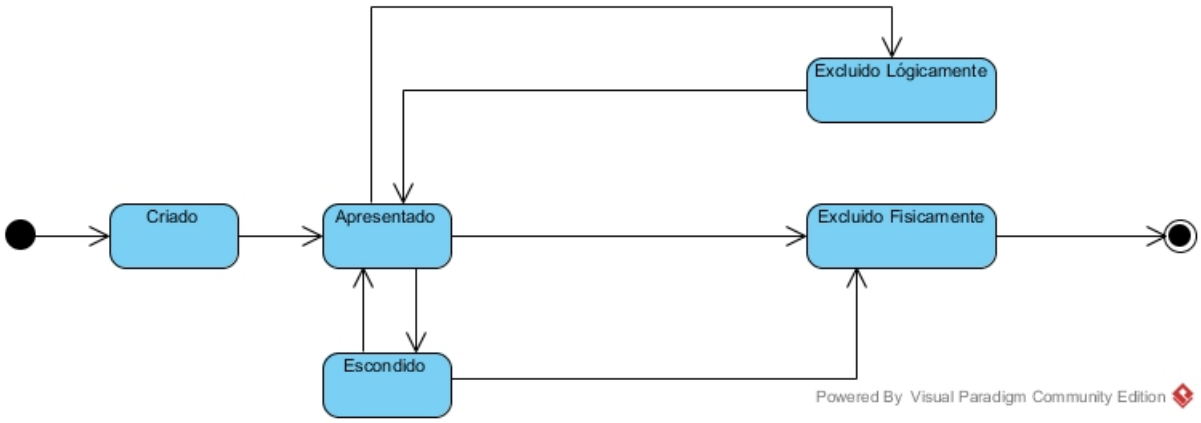


Figura 4.6: Diagrama de Estado do Objeto Vértice da *NET-UML*. Fonte: Próprio autor

4.1.5 Classes NET-UML

Conforme apresentado na Seção 4.1.2, a *NET-UML*, devido aos conceitos de teoria dos grafos, especializa o elemento classe proposto pela UML em duas novas categorias, *Object Graph* e *Object Layout*, através da adição de elementos pictográficos conforme apresentamos a seguir nas Seções 4.1.5.1 e 4.1.5.2.

4.1.5.1 Classes Object Graph

As classes *Object Graph* têm a função de representar os elementos matemáticos, vértice, aresta, arco e grafo que são utilizados para representar redes. As questões conceituais que provocaram o surgimento desta categoria de classes foram discutidas no Capítulo 3, Seção 3.2.

Apresentamos na Figura 4.7 as classes *Object Graph* e discutimos a seguir seus conceitos:

- Classe Vértice – representa o elemento fundamental que dá origem às arestas e aos grafos é o único objeto que possuirá atributo espacial que através da relação de dependências com as classes *Object Layout* determinará seu posicionamento ou distribuição no leiaute da rede.
- Classe *Edge* – representa o objeto formado pela relação entre os vértices quando neste não houver a necessidade de explicitar o direcionamento da relação.

- Classe *Arc* – representa o objeto formado pela relação entre os vértices quando neste há a necessidade de explicitar a direção da relação.
- Classe *Graph* – representa o elemento matemático grafo formado pelo relacionamento com as classes Vertice e Edge/Arc.

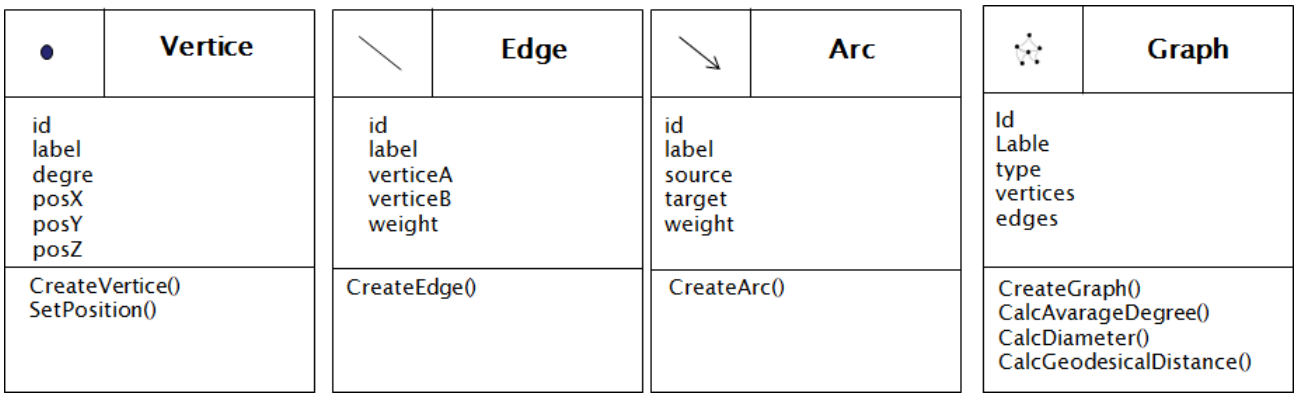


Figura 4.7: Classes Object Graph da *NET-UML*. Fonte: Próprio autor

4.1.5.2 Classes Object Layout

As classes *Object Layout* representam os diferentes leiautes de distribuição espacial dos grafos uteis para análise de redes sociais e complexas, elencados e categorizados no Capítulo 3, Seção 3.4.

A Figura 4.8 exibe as classes *Object Layout* a seguir são descritos seus conceitos:

- Classe Circular – representa as distribuições espaciais de grafo como leiaute Estrela, Circular Básico, *K-Core* dentre outros.
- Classe *Force Direct* – representa as distribuições espaciais de grafo com leiaute dirigido por forca como *Force Atlas 2*, *Kamada Kawai*, *Fruchterman Reingold* e *Open Ord*.
- Classe *Hierachical* – representa distribuições onde a representação de hierarquia é necessária para a análise da rede a exemplo os laiautes de *Tree* e *Sugiyama*.
- Classe *Geometric Transformation* – representam algoritmos espaciais que aplicam transformações geométricas em grafos que anteriormente já foi aplicado algum algoritmo de distribuição como Circular, *Force Direct* e *Hierachical*. Exemplos de

algoritmos de transformação geométrica são *ClockWise Rotate*, *Center ClockWise Rotate*, *Contraction* e *Expansion*.

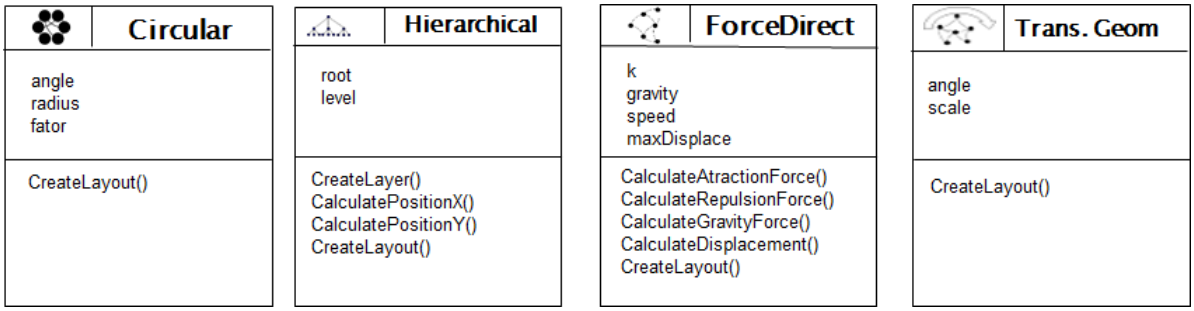


Figura 4.8: Classes Object Layout da *NET-UML*. Fonte: Próprio autor

4.1.6 Relacionamentos

O modelo *NET-UML* faz uso dos tipos de relacionamentos propostos pela UML e o relacionamento de generalização espacial proposto pelo GEO-OMT, devido aos mesmos atenderem as necessidades que as associações do paradigma OO e também as associações espaciais que ocorrem entre as classe *Object Graph* e *Object Layout*.

Existem restrições quanto ao relacionamento de agregação, composição e generalização espacial, pois estes somente podem ocorrer entre classes do mesmo tipo, ou seja, entre classes *Object Graph* e entre classes *Object Layout*. O relacionamento de dependência poderá acontecer entre qualquer tipo de classes sem restrições.

Apresentamos respectivamente nas Figuras 4.9 e 4.10 os relacionamentos entre classes suportados pela *NET-UML* e exemplo de diagrama de classes utilizando os elementos da *NET-UML*.

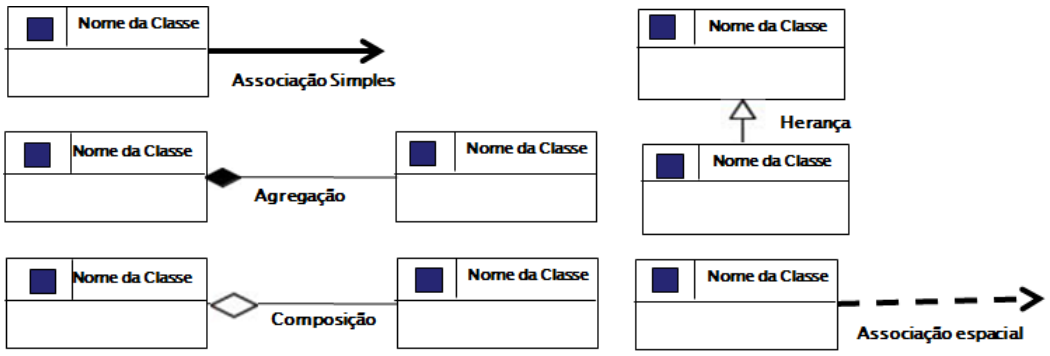


Figura 4.9: Relacionamentos suportados pela *NET-UML*. Fonte: Próprio autor

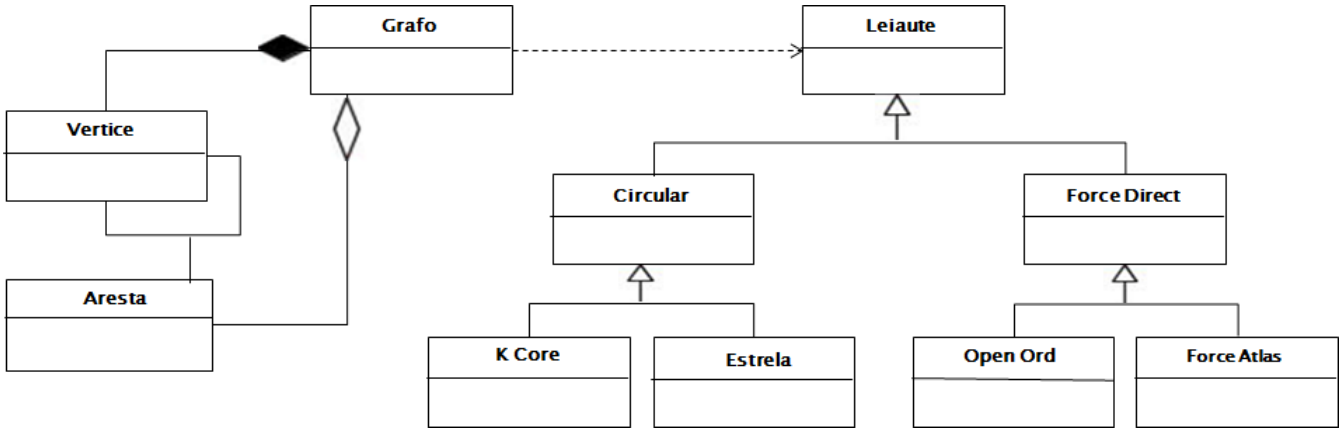


Figura 4.10: Relacionamentos entre classes do *NET-UML*. Fonte: Próprio autor

4.2 *NET-UML versus GEO-OMT*

Conforme apresentado no Apêndice B, o modelo *GEO-OMT* da suporte a construção de modelos de objetos para aplicações SIG, apoiando a modelagem de estruturas topológicas e de redes bem como relacionamentos espaciais (Borges, 1997a).

O modelo *NET-UML* apoia a etapa de análise de projeto de softwares de visualização de redes sociais e complexas, dando ênfase na separação dos objetos relacionados com a representação teórica (discutidos no Capítulo 3, Seções 3.2 e 3.3), dos objetos que representam os elementos comportamentais de distribuição de grafos (discutidos no Capítulo 3, Seção 3.4).

Devido ao contexto das aplicações SIG, todas as classes categorizadas como georreferenciadas possuem atributos espaciais, pois estas representam regiões da superfície terrestre e semanticamente são elementos espaciais (Borges, 1997a). No contexto das aplicações de visualização de redes sociais e complexas, apenas o elemento vértice possui representação espacial haja visto, que o posicionamento das arestas é resultado do posicionamento dos vértices e a distribuição destes depende do leiaute escolhido.

No modelo *NET-UML*, a separação dos objetos das classes *Object Layout* que representam os leiautes de distribuição de grafos é necessária, pois os diferentes tipos de leiautes apresentam princípios estéticos, técnicas e finalidades de acordo ao tipo de análise que se pretende realizar a partir da inspeção visual dos grafos. Já no modelo *GEO-OMT*, a divisão das classes conceituais (classes georreferenciadas) se da pela finalidade da representação geográfica conforme apresentado no Apêndice B, o *GEO-OMT* as classifica como Geo-Campos, que abstraem objetos distribuídos continuamente no espaço, e Geo-Objetos,

que abstraem objetos distribuídos individualizados no espaço (Borges, 1997a).

Quando discutimos relacionamento entre os objetos dos modelos *GEO-OMT* e *NET-UML*, observamos que o *GEO-OMT* possui maior complexidade devido ao grande número de elementos espaciais representado pelo modelo e suas características geográficas conforme visto na Apêndice B. O modelo *NET-UML* utiliza os relacionamentos propostos pela UML e faz uso do relacionamento espacial proposto pelo *GEO-OMT*.

4.3 Considerações do Capítulo

Apresentamos neste capítulo o modelo *NET-UML* como proposta de extensão para a UML, construído a partir da semântica e sintaxe inerente a teoria e leiautes de distribuição dos grafos.

Apresentaremos no próximo capítulo os resultados obtidos com a construção e experimentação do modelo *NET-UML* no desenvolvimento da ferramenta de visualização de redes sociais e complexas *Open Source SC NET DRAW*.

Capítulo Cinco

Resultado e Discussões

Abordaremos neste capítulo os resultados obtidos com uso da *NET-UML* para a construção da *SC NET DRAW* e verificação das técnicas de modelagem utilizadas para a construção do *Gephi*, *JGraph*, *iGraph* e *Graph Sharp*.

A análise dos resultados e comparações têm como objetivos: validar se o modelo *NET-UML* possui relevância; potencializa a construção de ferramentas computacionais que possuem funcionalidades de visualização de redes sociais e complexas e; verificar se as ferramentas pesquisadas foram construídas com o uso de técnicas de modelagem especificas para o domínio das redes sociais e complexas.

5.1 *SC NET DRAW - Ferramenta de Visualização de Redes Sociais e Complexas*

Como forma de testar o modelo proposto neste trabalho, desenvolvemos a ferramenta de visualização de redes sociais e complexas denominada *SC NET DRAW*, cujo o modelo de análise foi criado com o uso da *NET-UML* e os algoritmos de visualização espacial dos grafos disponibilizados na ferramenta foram discutidos no Capítulo 3.

A *SC NET DRAW* tem o objetivo de prover ao analista de redes sociais e complexas funcionalidades de criação, inspeção visual e calculo de propriedades fundamentais de grafos disponibilizando os seguintes leiautes de distribuição de grafos:

- Circulares.
  - Circular básico.
  - *K-Core*.
  - Estrela.
  - Ego.
- Dirigidos por força.
  - *Kamada Kawai*.
  - *Fruchterman Reingold*.
  - *Force Atlas 2*.

- Hierarquicos.

  Tree.

  Sugyama.

- Transformação geométrica.

  Contração.

  Expansão.

  Clockwise Rotate.

  Center Clockwise Rotate.

5.1.1 *Técnicas e Tecnologias Utilizadas*

A *SC NET DRAW* foi concebida para ser utilizada em ambiente desktop, em computadores com o sistema operacional *Windows 7 64 Bits* ou superior. Para a sua construção foram utilizadas as linguagem de desenvolvimento OO *Microsoft C Sharp*, as bibliotecas de funções gráficas *Open GL* e *TAO Framework*. Os modelos de análise foram criados com o uso da *NET-UML*, uma das contribuições da presente pesquisa.

Apresentamos na Figura 5.1 diagrama do projeto de software da *SC NET DRAW*, destacando as camadas e os elementos que as compõem.

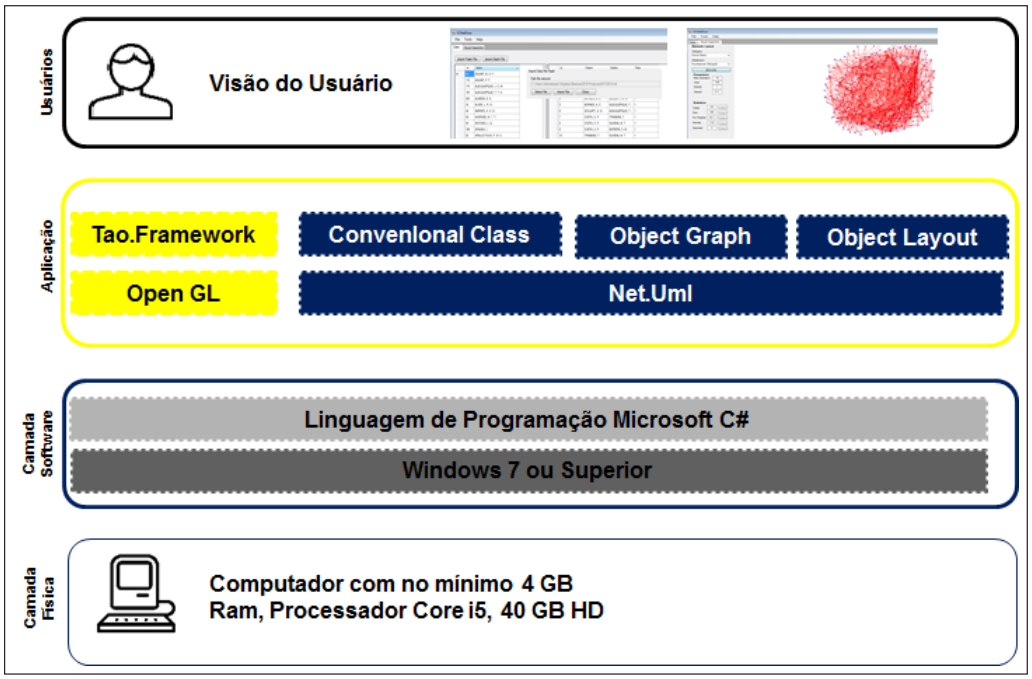


Figura 5.1: *SC NET DRAW* Macro Arquitetura. Fonte: Próprio autor

Como estruturas de dados para criação dos grafos e seus elementos foram utilizadas vetores de objetos vértices e aresta, respeitando a definição do elemento matemático grafo conforme discutido no Capítulo 3 e evidenciado na Seção 5.4.

Criamos também estrutura de dados de lista de adjacência para que fosse possível a implementação dos algoritmos de busca em profundidade e criação da árvore geradora mínima, necessários para criação dos leiaute *Tree* e *Sugyiama*, uma vez que a estrutura de vetor de objetos vértices e arestas não são suficientes para criação de tais algoritmos.

Conforme descrito no Capítulo 4 Seção 4.1.5.1, o objeto vértice é o único a possuir atributos espaciais e a partir da seleção de um algoritmo de distribuição de grafo são feitos cálculos de posicionamento dos vértices no plano cartesiano definido para apresentação do leiaute do grafo. As arestas não possuem atributos espaciais pois sua escrita e posicionamento no leiaute do grafo se da de acordo aos vértices que as conecta.

5.1.2 SC NET DRAW - Macro Requisitos e Diagrama de Casos de Uso

Conforme discutido na Seção 5.1, a *SC NET DRAW* tem o objetivo de fornecer ao analista de redes sociais e complexas funcionalidades de criação de grafos, calculo das propriedades fundamentais e inspeção visual dos grafos a partir da aplicação de leiautes de distribuição. Para alcançar estes objetivos foram definidos os seguintes macro requisitos:

- REQ001 - Criação de redes a partir da importação de dados de arquivos *NET* gerados pelo software *Pajek*.
- REQ002 - Criação de redes a partir da importação de dados de arquivos *CSV* criados pelo software *Gephi*.
- REQ003 - Criação de projeto de visualização de redes - funcionalidade responsável por gravar os projetos de rede de acordo com as manipulações realizadas pelo analista de rede.
- REQ004 - Inspeção visual - funcionalidade que possibilita ao analista de rede realizar inspeção visual nas redes através da seleção de leiaute de distribuição que mais se adeque aos objetivos da análise.
- REQ005 - Calculo das propriedades fundamentais dos grafos - funcionalidade que possibilita ao analista de rede calcular as propriedades fundamentais dos grafos como grau, tamanho da rede, densidade, grau médio, diâmetro e distância geodésica.

A construção do diagrama de casos de uso, levou em consideração a proposta da *NET-UML* no que se refere aos usos mandatórios que ferramentas de visualização de redes

devem possuir, adicionamos usos específicos para as funcionalidades de visualização de acordo com os algoritmos que serão disponibilizados pela *SC NET DRAW*, bem como os casos de usos relacionados a estratégia de criação das redes.

Apresentamos na Figura 5.2, diagrama de casos de uso com a descrição das funcionalidades relacionadas com sintaxe e conceitos da teoria dos grafos. Os casos de uso UC01, UC02, UC03 e UC04 representam as funcionalidades de criação do elemento matemático grafo a partir da criação do vértice, arestas ou arcos, já os casos de uso UC05 e UC06 representam as funcionalidades de importação de redes a partir de arquivos originados pelos software *Gephi* e *Pajek* possuindo assim, relação de inclusão com o caso de uso UC04, que representa a funcionalidade de criação de grafos.

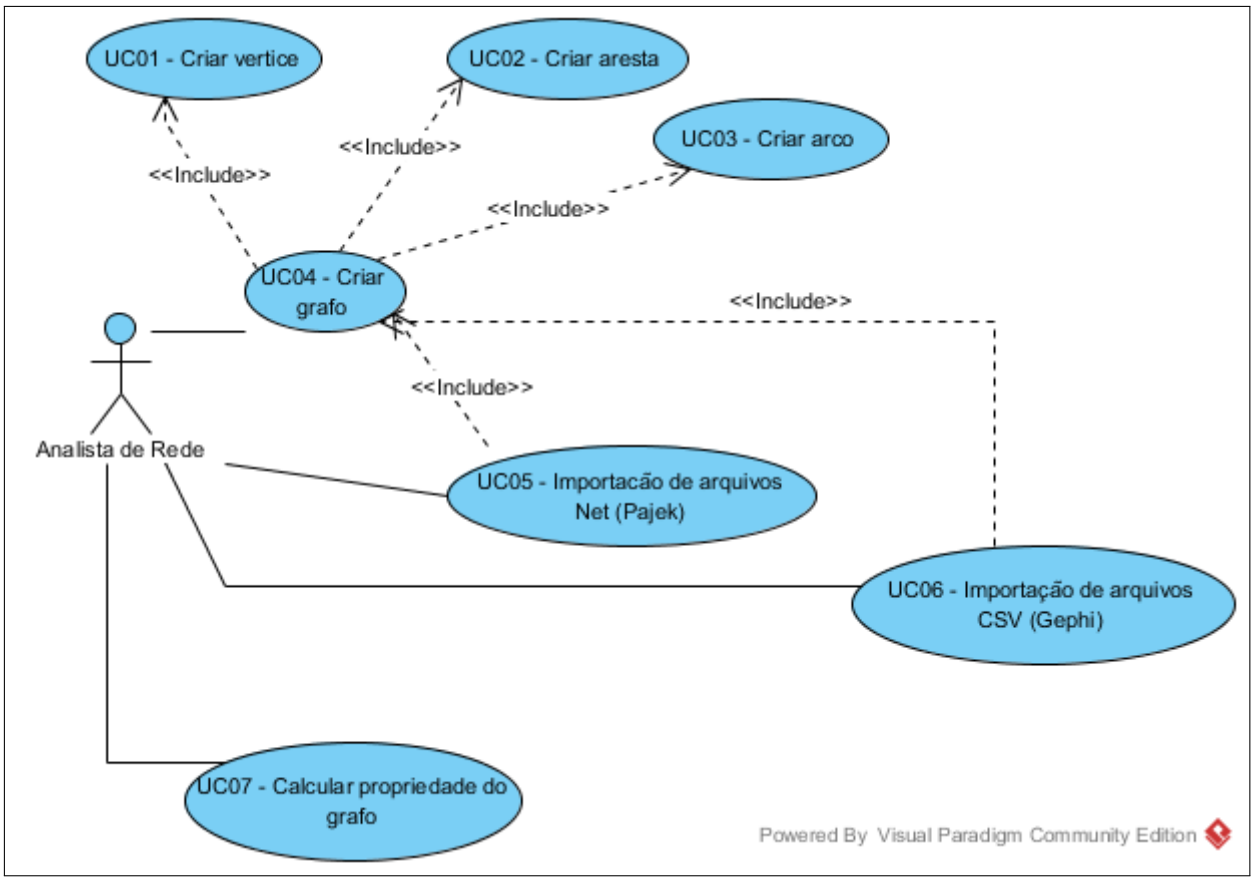


Figura 5.2: *SC NET DRAW* Casos de Uso de funcionalidade relacionadas com os aspectos sintáticos. Fonte: Próprio autor

Detalhamos as funcionalidades do macro requisito REQ004 - Inspeção visual, nos seguintes casos de uso genéricos de leiautes de distribuição de grafo: UC08 - Criar leiaute de circular; UC09 - Criar leiaute dirigido por força; UC10 - Criar leiaute hiaerárquico e; UC11 - Criar leiaute de transformação geométrica conforme apresentamos na Figura 5.3.

Especializamos os casos de uso genéricos de acordo com a categoria dos leiautes que representam, o caso de uso UC08 representa a funcionalidade de criação de leiautes circulares agregando os leiautes K-Core, Circular Básico e Star, o caso de uso UC09 representa as funcionalidade de distribuição de grafos dirigidos por força, agregando os leiautes *Fruchterman Reingold*, *Kamada Kawai* e *Force Atlas 2*, já o caso de uso UC10 representa a funcionalidade de distribuição de grafos por hierarquias, agregando os leiautes *Sugyama* e *Tree* enquanto o caso de uso UC11 dão conta dos leiautes de transformação geométrica, agregando as funções de rotação no sentido horário e anti horário bem como a operação de contração e expansão.

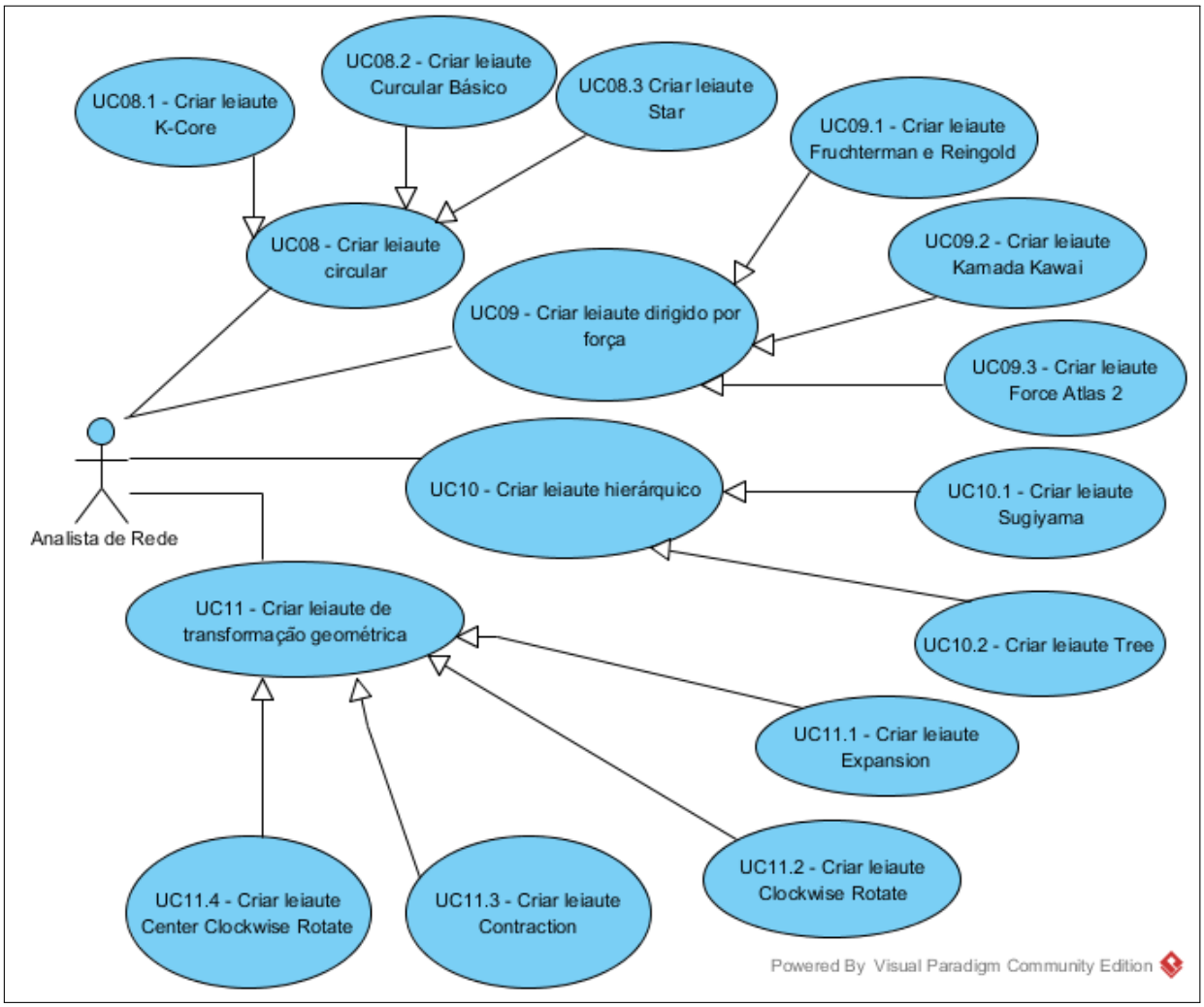


Figura 5.3: *SC NET DRAW* Casos de Uso de funcionalidade relacionadas com os aspectos semânticos. Fonte: Próprio autor

5.1.3 SC NET DRAW - Diagramas de Máquina de Estados

Utilizamos na construção desta ferramenta os diagramas de maquina de estado propostos pela *NET-UML* por concordar que os objetos mais complexos a serem implementados pela ferramenta são os objetos vértices e grafo, conforme discutido no Capítulo 4 Seção 4.1.4 Figuras 4.6 e 4.5.

Entender conceitualmente o comportamento das mudanças dos estados do objetos grafo e vértice, no contexto das ferramentas de visualização, é de fundamental importância para correta implementação das funcionalidades, haja visto a existência de diferentes estados e eventos que provocam tais mudanças sobretudo, as alterações de posicionamento dos vértices a partir da aplicação dos algoritmos de distribuição dos grafos.

Como exemplo, citamos o leiaute de distribuição espacial de grafos Ego, que a partir da seleção de um dado vértice, distribui e apresenta os vértices adjacentes a este em uma disposição circular e os vértices vizinhos dos vizinhos em uma segunda circunferência, os demais vértices por não possuírem conexões devem ser escondidos e não excluídos uma vez que caso seja selecionado novo vértice para análise a partir do leiaute Ego novos vértices serão apresentados e outros serão escondidos.

5.1.4 SC NET DRAW - Diagrama de Classes do Modelo de Análise

Após levantamento dos requisitos e detalhamento dos casos de uso construímos o modelo de análise de acordo aos elementos propostos pela *NET-UML* para representação semântica e sintática, sendo esta a primeira abstração dos objetos do domínio das aplicações de visualização de redes sociais e complexas. Apresentamos na Figura 5.4 o modelo de análise da *SC NET DRAW*, discutiremos a seguir os elementos que compõem o modelo.

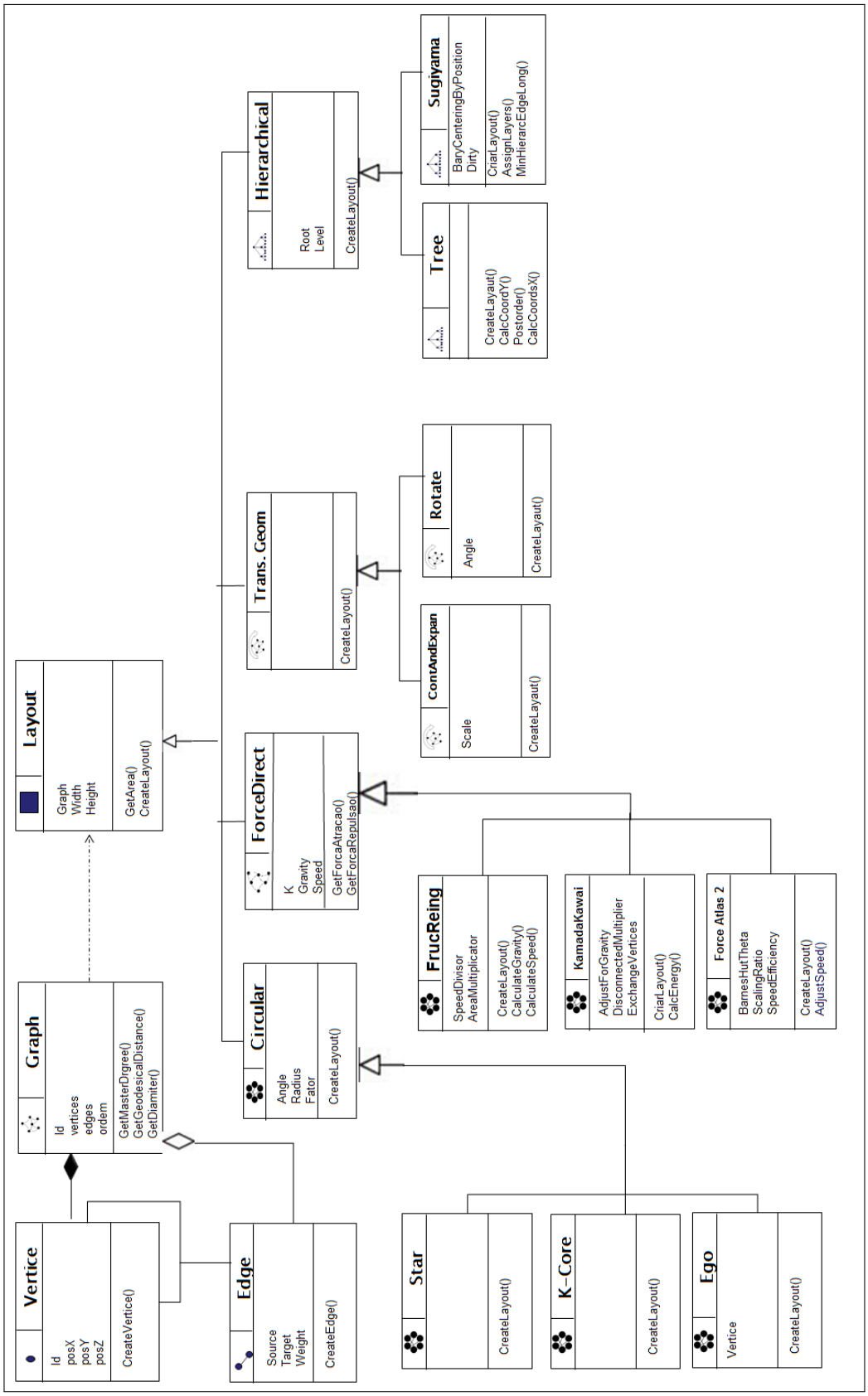


Figura 5.4: Modelo de Análise da ferramenta *SC NET DRAW*. Fonte: Próprio autor

5.1.5 *SC NET DRAW - Classes Object Graph*

Durante a tarefa de análise, obviamente, identificamos como classes *Object Graph*, que cobrem os objetos da teoria do grafo, as classes: vértice, aresta e grafo conforme destacamos na Figura 5.5. A classe grafo possui relacionamentos de agregação com a classe vértice e composição com a classe aresta de acordo com a definição do elemento matemático grafo, discutida no Capítulo 3 Seção 3.2.

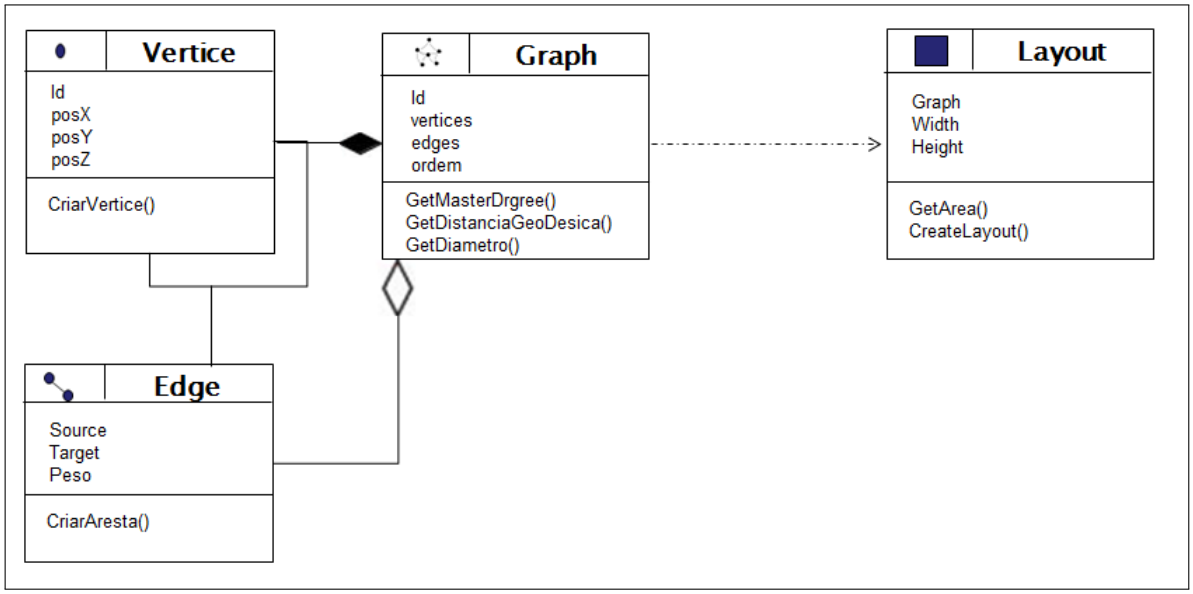


Figura 5.5: Modelo de Análise *SC NET DRAW* - Classes Object Graph. Fonte: Próprio autor

5.1.6 *SC Net Draw - Classes Object Layout*

As classes *Object Layout*, que representa os diferentes leiautes de distribuição espacial dos grafos, foram representadas no modelo de análise a partir da definição de três níveis de classes evidenciando o relacionamento de herança.

O nível 1 é composto pela classe abstrata *Layout* que possui os atributos genéricos que todo leiaute de distribuição deve ter independente da sua categoria.

O nível 2 é composto por classes abstratas que refletem a categoria dos leiautes de distribuição discutidos pela *NET-UML*. Neste nível as classes possuem atributos e métodos genéricos de acordo a categoria do leiaute. As classes que compõem este nível são: Circular, *Force Direct*, *Geometric Transformation* e *Hierarchical*.

O nível 3 é composto pelas classes concretas que implementam os leiautes de distribuição,

possui atributos e métodos específicos, de acordo as técnicas e premissas estéticas relacionadas com cada um dos leiautes. Este nível é composto pelas classes: *Basic Circular*, *Star*, *K-Core* e *Ego* relacionadas a classe Circular; *Fruchterman Reingold*, *Force Atlas 2* e *Kamada Kawai* relacionadas com a classe *Force Direct*; *Expansion* e *Rotate* relacionadas com a classe *Geometric Transformation* e; *Tree* e *Sugiyama* relacionadas com a classe *Hierarchical*.

Apresentamos na Figura 5.6, as classes de distribuição espacial de grafo da categoria Circular. Criamos a classe abstrata denominada Circular e nesta foram colocados os atributos comuns a todos os algoritmos desta categoria. O atributo *angle* representa o vértice cujo o posicionamento do leiaute se dará pelas operações trigonométricas *cosseno*, posição do vértice em relação ao eixo X, e *seno*, posição do vértice em relação ao eixo Y. O atributo *radius* representa o valor do raio da circunferência e o atributo $fator$ é utilizado para manter o valor de ajuste do raio da circunferência. Os leiautes de distribuição apresentados pelas classes *Basic Circular* e *Star* não necessitam de novos atributos para a construção dos respectivos algoritmos já o leiaute apresentado pela classe *Ego* necessita do atributo $Vertice$ para a construção do algoritmo uma vez, que a distribuição dos demais vértices levam em conta a proximidade com o vértice informado como parâmetro.

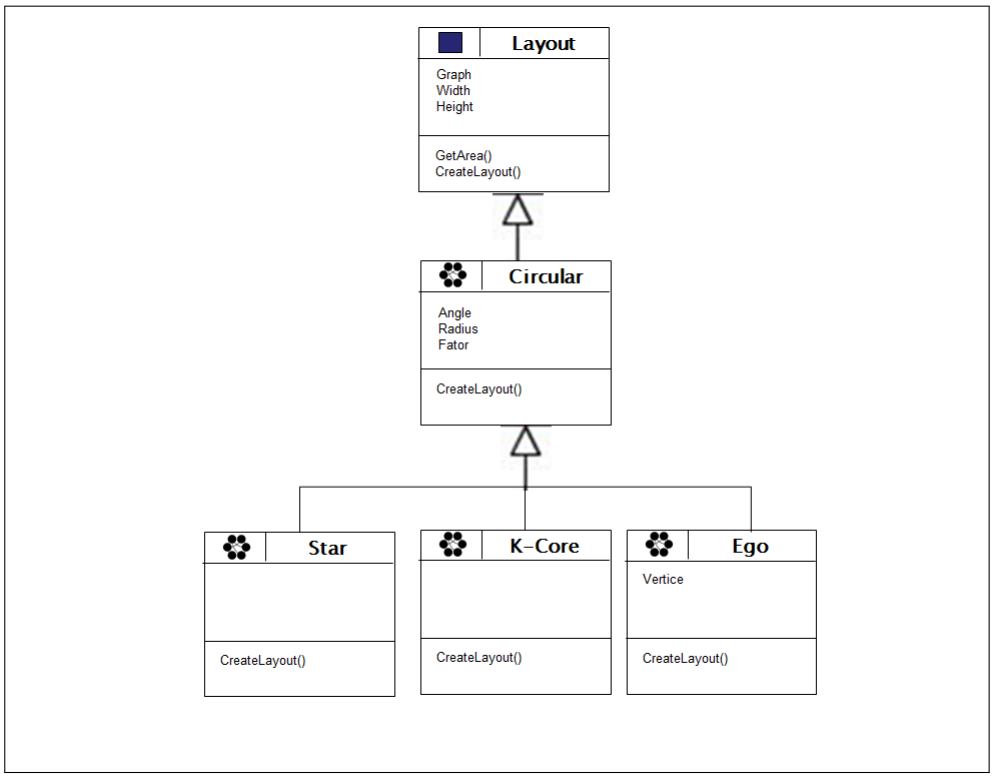


Figura 5.6: *SC NET DRAW* - Classes Object Layout - Circular. Fonte: Próprio autor

Apresentamos na Figura 5.7, as classes de distribuição espacial da categoria *Force Direct*. Criamos uma classe abstrata de mesmo nome com os atributos, K, $Grvaity$, *Speed*

e *MaxDisplace* comuns a todas as classes desta categoria e que representam respectivamente: constante utilizada nos cálculo das forças de atração e repulsão; força da gravidade; velocidade do deslocamento dos vértices e; deslocamento máximo dos vértices. Conforme discutido no Capítulo 3, os algoritmos *Force Direct* utilizam princípios físicos para distribuir os vértices no leiaute sendo assim existem métodos genéricos comuns a todas as classe desta categoria que são *CalculateAtractionForce()*, *CalculateRepulsionForce()*, *CalculateGravityForce()* e *CalculateDisplacement()*.

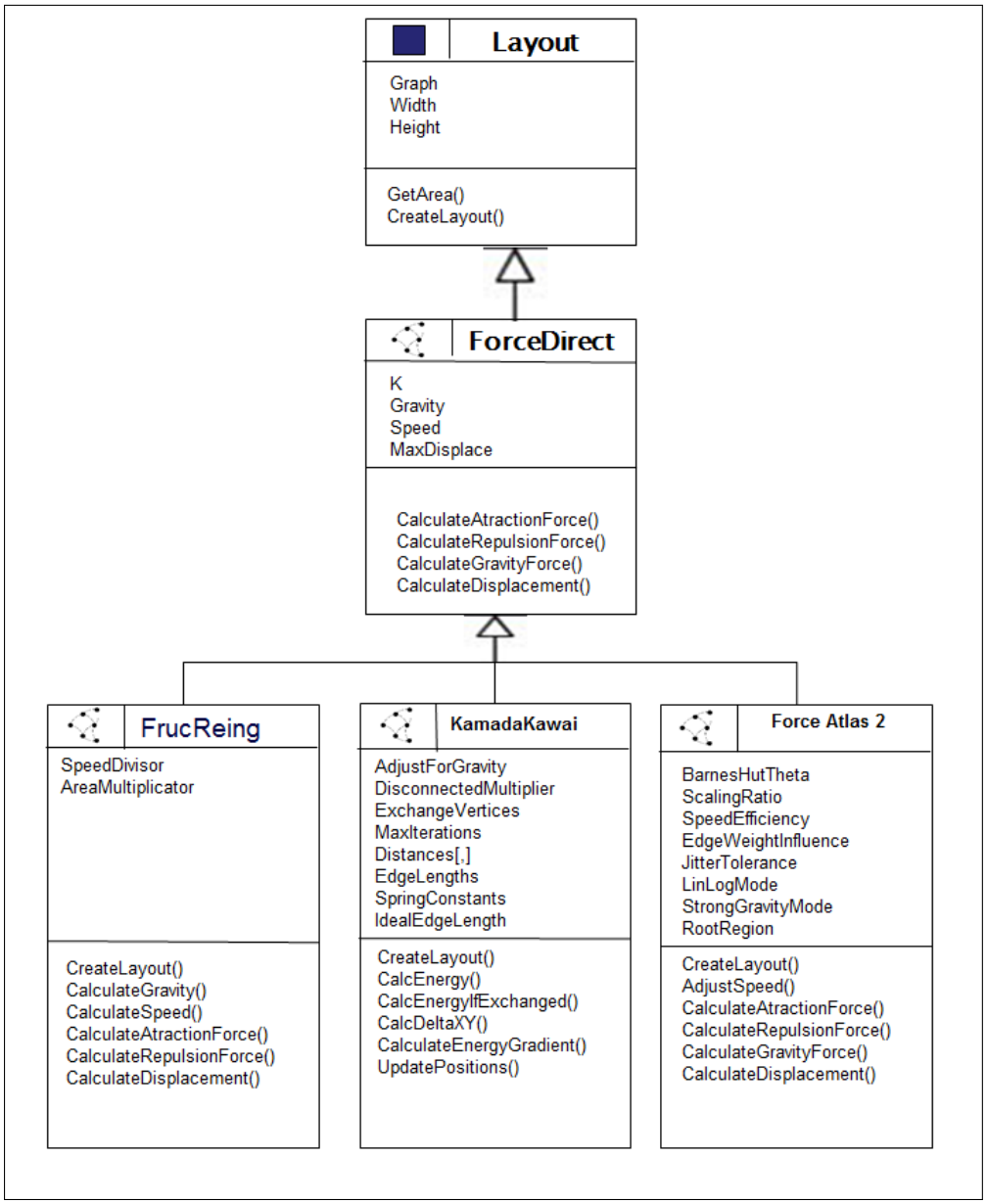


Figura 5.7: *SC NET DRAW* - Classes Object Layout - *Force Direct*. Fonte: Próprio autor

Apresentamos na Figura 5.8, as classes de distribuição espacial da categoria *Hierarchical*. Criamos uma classe abstrata também com o mesmo nome da categoria dos algoritmos. Os atributos comuns aos algoritmos de distribuição desta categoria são *root*, represen-

tando o vértice raiz da árvore e *level* que representa a quantidade de níveis existentes na hierarquia. A classe *Tree* utiliza os atributos comuns aos algoritmos hierárquicos e possui métodos específicos para posicionamento dos vértices em relação ao eixo Y, calculo da ordenação e deslocamento dos vértices bem como o posicionamento dos vértices em relação ao eixo X, respectivamente os métodos CalculateCoordY(), PostOrder() e CalculateCoordX(). A classe Sugiyama possui atributos específicos para ordenação dos vértice, adição de vértices aleatórios, verificação de *gaps* horizontais/verticais e largura máxima do leiaute, respectivamente atributos: *BaryCenteringByPosition*, *dirty*, *horizontalGap*, *verticaGap* e *maxWidth*. Os métodos específicos da classe Sugiyama são *AssignLayers()* responsável pelo calculo e atribuição das camadas do leiaute, *AdjustSugiyamaLeiaute()* responsável ajustar o posicionamento dos vértices no leiaute, *MinimizeHieraquicalEdgeLonge()*, responsável por minimizar o tamanho das arestas longas e *HorizontalPositionAssignmentOnLayer()* responsável por posicionar os vértices horizontalmente.

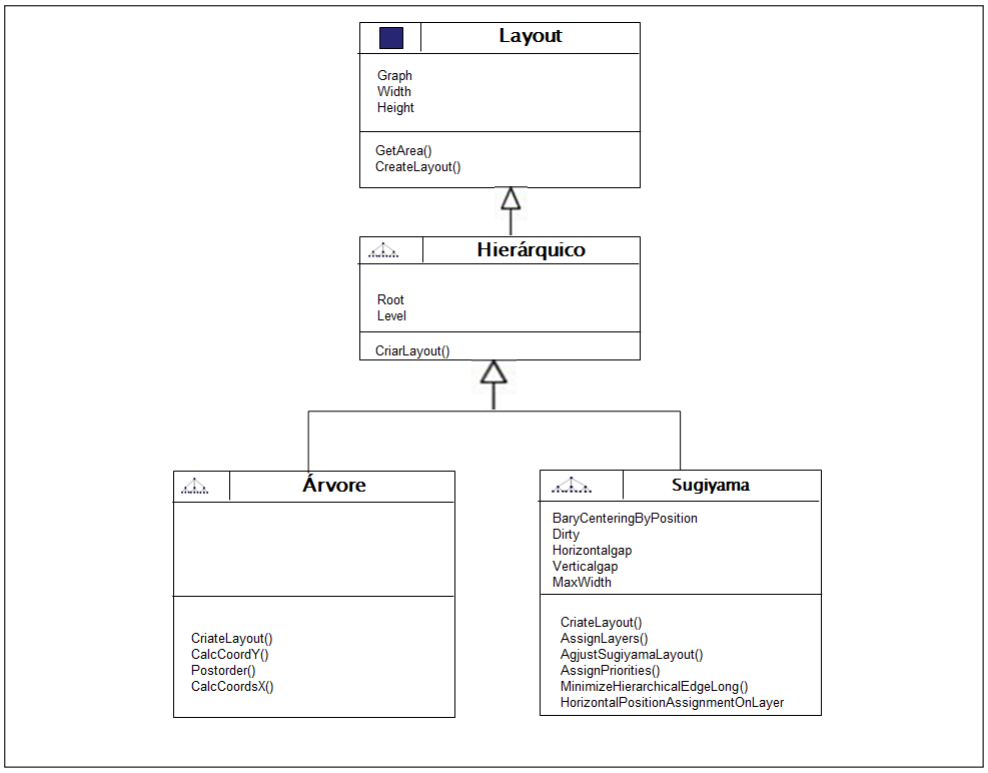


Figura 5.8: *SC NET DRAW* - Classes Object Layout - *Hierarchical*. Fonte: Próprio autor

Apresentamos na Figura 5.9, as classes dos algoritmos *Geometric Transformation*, conforme discutido anteriormente, esta classe de algoritmo se ocupa de realizar operações de transformação geométrica de expansão, contratação e rotações no sentido horário e anti-horário em leiautes de grafos cuja a distribuição espacial foi dada a partir do uso outros leiautes de distribuição.

Criamos a classe abstrata *Geometric Transformation* com a finalidade de evidenciar a

categorização dos algoritmos sendo que esta não possui atributos e métodos a serem utilizados pelas classes relacionadas.

A classe *ContractAndExpansion* implementa os algoritmos de expansão e retração do posicionamento dos vértices no leiaute da rede, possui o atributo *scale*, conforme discutido no Capitulo 3 Seção 3.4.2.4, caso o valor do atributo *scale* for menor que 1 o elementos do leiaute sofrerão contração, caso o valor do atributo *scale* for maior que 1 os elementos serão expandidos e caso o valor do atributo *scale* for igual a 1 o leiaute não sofrerá nenhum tipo de modificação.

A classe *Rotate* implementa os algoritmos de rotação do leiaute do grafo nos sentidos horário e anti-horário, possui o atributo *angle* que representa o angulo de rotação do leiaute do grafo, conforme também discutido no Capitulo 3 Seção 3.4.2.4. Caso o valor do atributo *angle* seja positivo, a rotação se dará em sentido horário, caso o valor do atributo seja negativo a rotação se dará no sentido anti-horário.

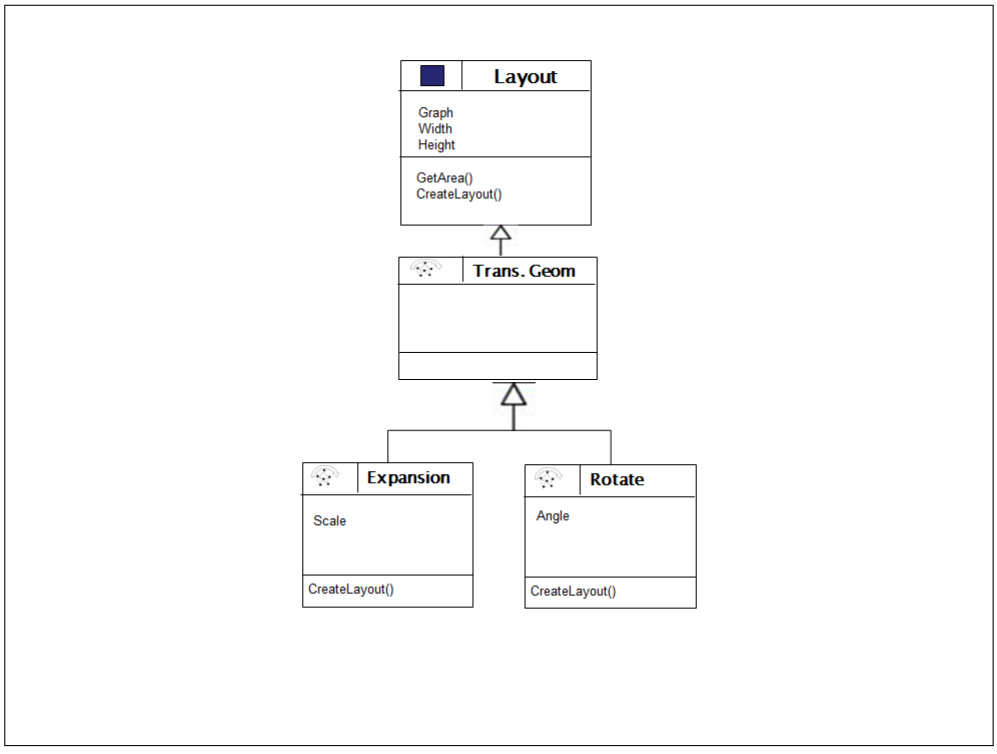


Figura 5.9: *SC NET DRAW* - Classes Object Layout - *Geometric Transformation*. Fonte: Próprio autor

5.1.7 SC NET DRAW da Análise a Codificação

Finalizada a etapa de análise com o uso da *NET-UML* e tendo como produtos: lista de requisitos, casos de uso e o diagrama de classes, de acordo a sintaxe e semântica da teoria dos grafos, a etapa seguinte seria a construção do projeto de software e codificação dos programas.

A *NET-UML* se mostrou ferramenta útil, tanto na etapa de análise, sendo este seu proposito inicial, quanto na etapa de projeto de software, uma vez que a mesma da apoia a construção das classes do projeto de software, relacionadas as redes sociais e complexas, bem como na definição dos padrões estruturais, comportamentais e de responsabilidades.

Discutiremos a seguir as funcionalidades de Criação de grafos de redes a partir da importação de arquivos de rede criados pelo software Pajek e Inspeção Visual de Grafos a partir da seleção de distribuição de grafos da categoria *Force Direct* a partir do uso dos elementos propostos pela *NET-UML*.

Apresentamos na Figura 5.10 a evolução da análise e desenvolvimento da funcionalidade de Criação de grafos a partir da importação de arquivos *Pajek*. Após a definição do macro requisito REQ001, verificamos que a *NET-UML* fornece diagrama de caso de uso mandatório que cobre os usos padrões quando se fala da criação de grafos, não importando se o grafo será criado a partir da importação de arquivo ou criação manual através de adição de vértices e arestas.

De posse do requisito e casos de uso foi possível identificar os objetos do modelo de análise que abstraem os elementos matemáticos que compõem o grafo a partir do uso das classes *Object Graph* proposta pela *NET-UML*, construir protótipo de tela da funcionalidade de criação de grafos e identificar as classes do projeto de software relacionadas com esta funcionalidade e padrão de projeto associado.

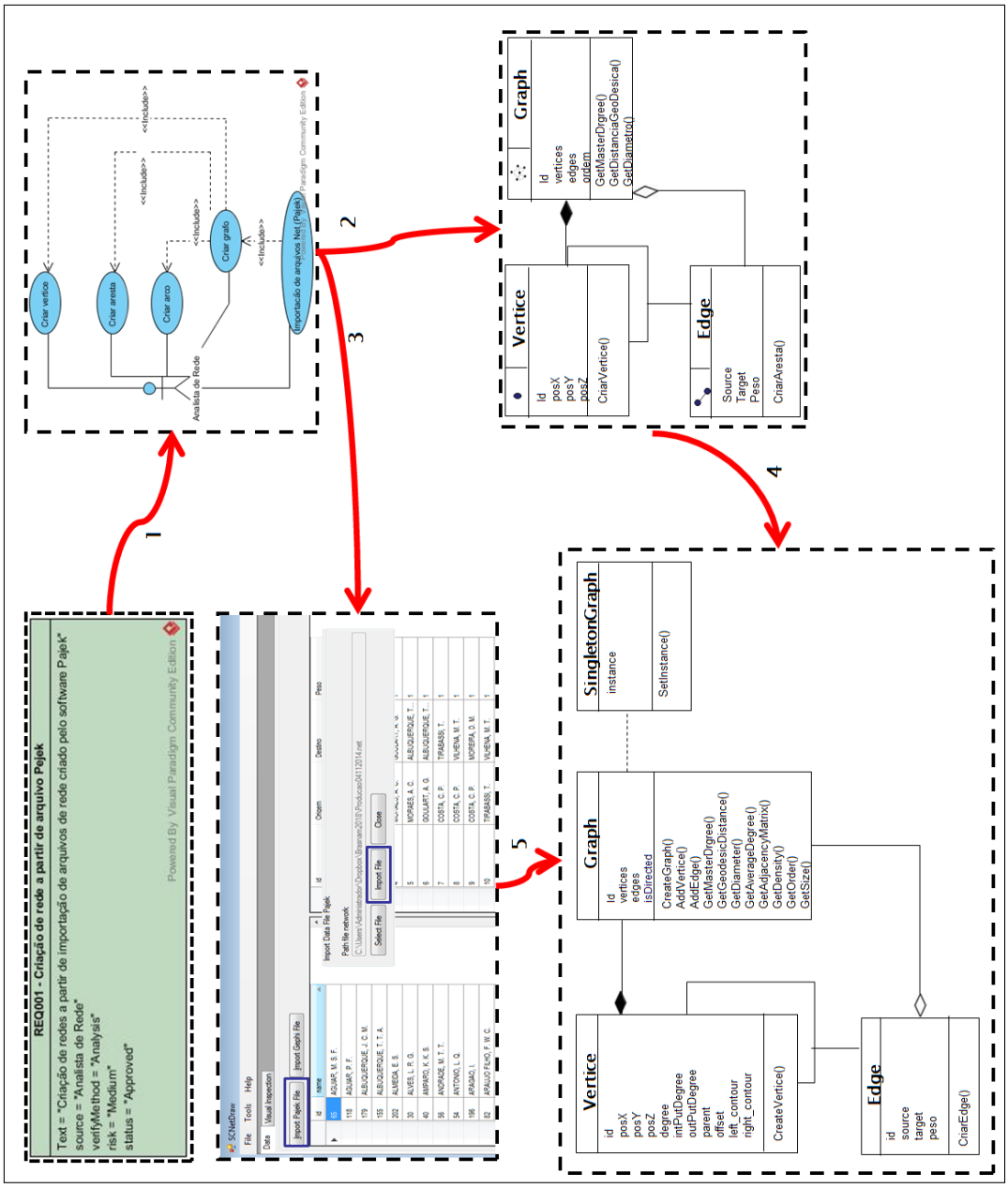


Figura 5.10: *SC NET DRAW* - Criação de Grafos a partir de arquivos Pajek. Fonte: Próprio autor

Verificamos na análise da funcionalidade de Criação de grafos a oportunidades do uso do padrão de projeto Singleton, uma vez que em uma pesquisa de análise de redes sociais e complexas o objeto grafo deve ter somente uma instancia, conforme apresentamos na Figura 5.11 (Gama et al., 2000).

```
using NetUML.Data.Business;
using NetUML.Data.ObjectGraph;

namespace SCNetDraw.Win.Controllers
{
    public class GraphController
    {
        public Graph CreateGraphByPajekFile(String pathFile)
        {
            GraphBLL graphBLL = new GraphBLL();
            Graph graph = Singleton<Graph>.SetInstance();

            graph= graphBLL.ConvertFileNetWorkPajekToGraph(pathFile);
            return graph;
        }
    }
}
```

Figura 5.11: *SC NET DRAW* - Padrão de Projeto Singleton implementado na linguagem C Sharp. Fonte: Próprio autor

Apresentamos na Figura 5.12 a evolução da análise e desenvolvimento da funcionalidade de Distribuição espacial de grafos, através de algoritmos *Force Direct* relacionado ao requisito REQ004.1.

Seguindo a mesma metodologia discutida na seção anterior, uma vez tendo o requisito definido o próximo passo é a definição dos casos de uso, por ser um modelo extensível, a *NET-UML* não define casos de usos específicos para as funcionalidade de distribuição de rede deixando a cargo do utilizador do modelo escolha da categoria e os leiautes a serem implementados.

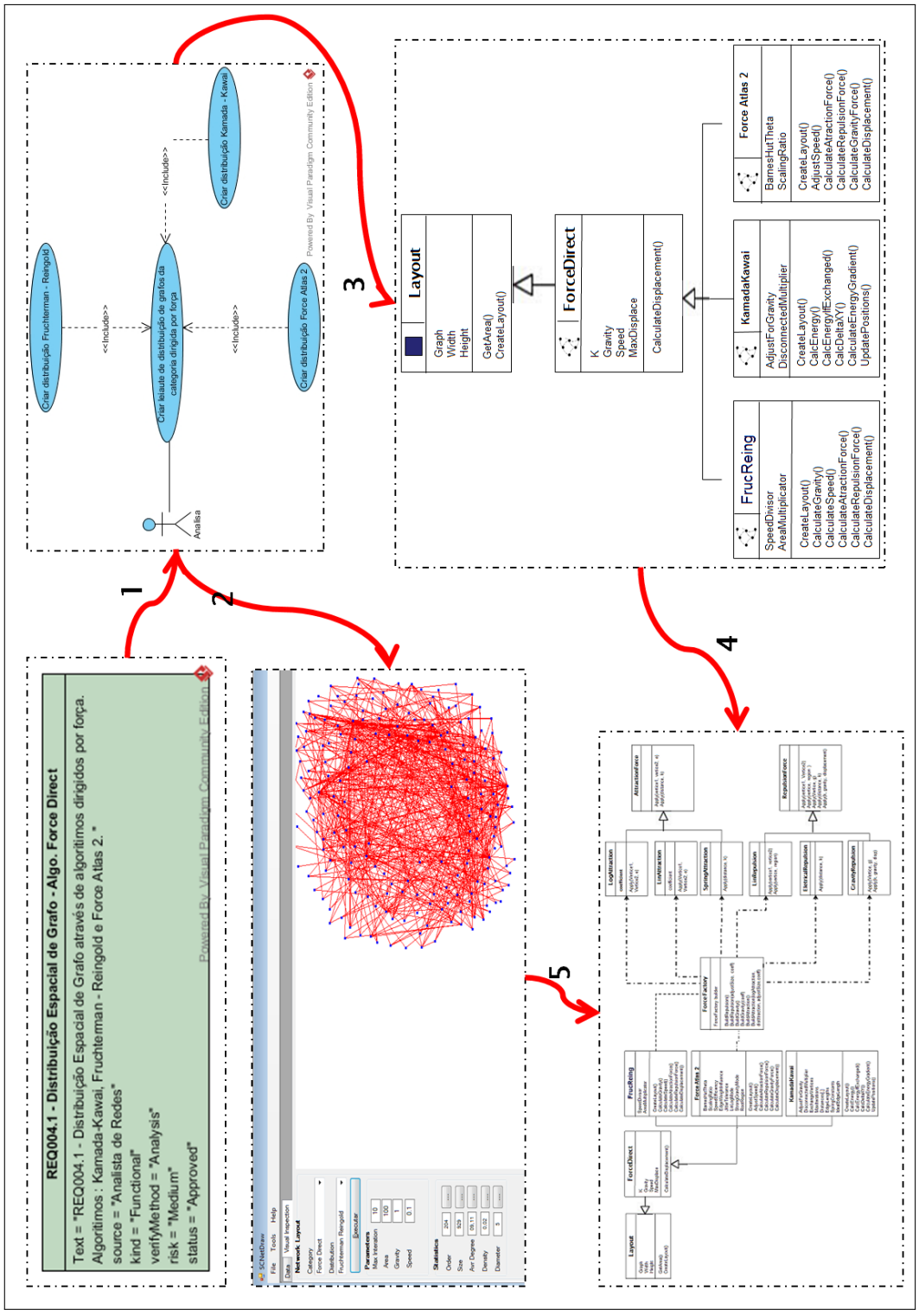


Figura 5.12: *SC NET DRAW* - Distribuição de Espacial de Grafos através de algoritmos *Force Direct*. Fonte: Próprio autor

Destacamos na Figura 5.13 as classes do projeto de software da funcionalidade de Distribuição espacial de grafos através de algoritmos *Force Direct*.

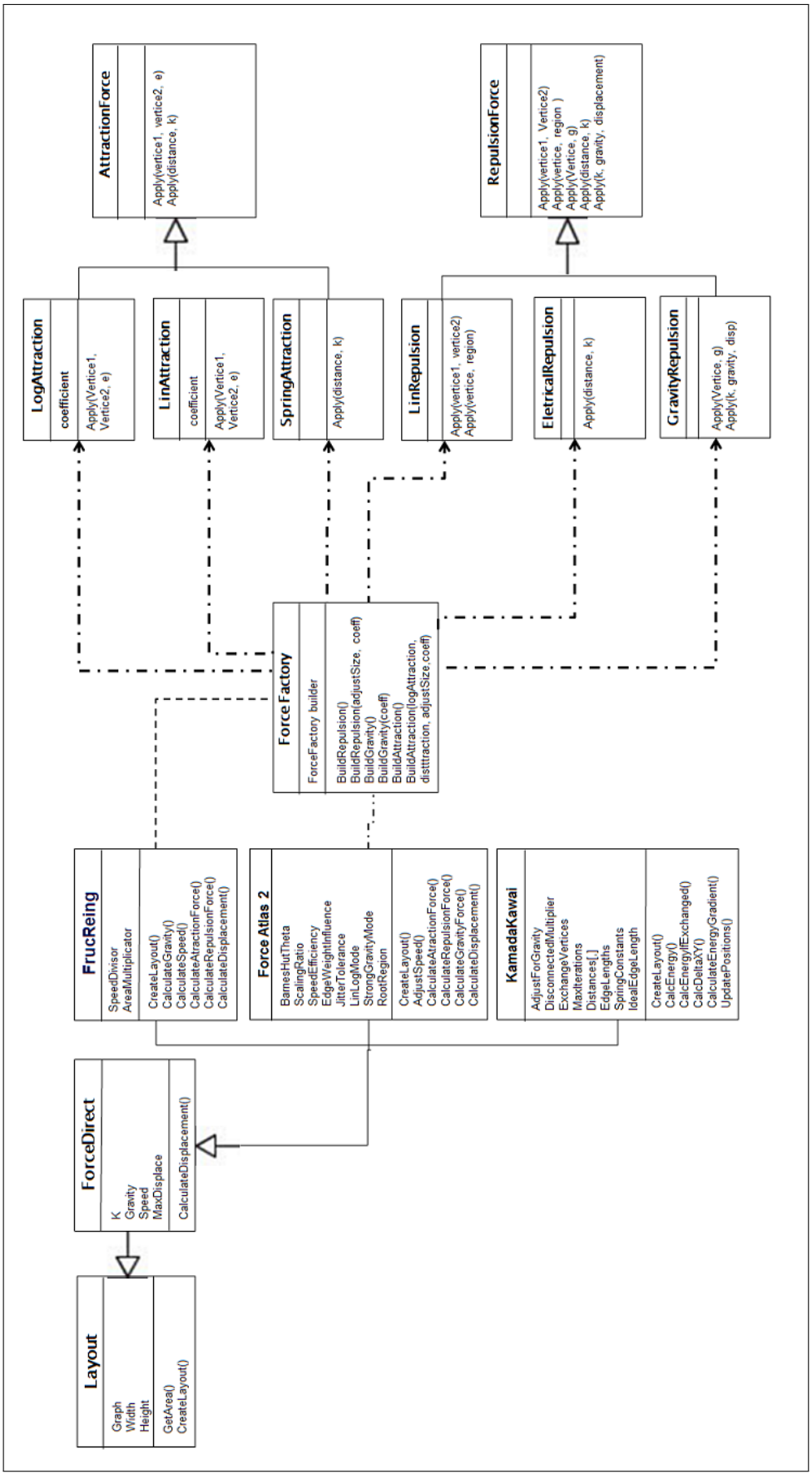


Figura 5.13: Modelo de Análise da ferramenta *SC NET DRAW*. Fonte: Próprio autor

Os algoritmos da categoria *Force Direct*, conforme discutido no Capitulo 3 Seção 3.4.2.3, utiliza princípios físicos para a distribuir espacialmente os vértices na rede porém, cada algoritmo utiliza um método especifico para realizar esta operação. Diante deste cenário identificamos a oportunidade para uso do padrão de projeto *Factory Method*, conforme apresentamos na Figura 5.13.

Foi criada a classe *Force Factory* que será a interface para criação dos objetos que possuem a implementação dos algoritmos de cálculo das forças de atração e repulsão dos vértices, as subclasses especificas, com a responsabilidade de decidir qual das classes será instanciada (Gama et al., 2000). Apresentamos na Figura 5.14, código que evidência o uso do padrão de projetos *Factory Method*.

```
using System;
namespace NetUML.Data.ConventionalClass.ForceDirectClass
{
    public class ForceFactory
    {
        public static ForceFactory builder = new ForceFactory();
        public RepulsionForce BuildRepulsion()
        {
            return new EletricalRepulsionForce();
        }
        public RepulsionForce BuildRepulsion(Boolean adjustBySize, double coefficient)
        {
            if (adjustBySize)
            {
                return new LinRepulsionAntiCollision(coefficient);
            }
            else
            {
                return new LinRepulsion(coefficient);
            }
        }
        public RepulsionForce BuildGravity()
        {
            return new GravityRepulsion();
        }
        public AttractionForce BuildAttraction()
        {
            return new SpringAttractionForce();
        }
        public AttractionForce BuildAttraction(Boolean logAttraction,
        Boolean distributedAttraction, Boolean adjustBySize, double coefficient)
        {
            if (distributedAttraction)
                {return new LogAttractionDegreeDistributedAntiCollision(coefficient);}
            else
                {return new LogAttractionAntiCollision(coefficient);}
        }
    }
```

Figura 5.14: *SC NET DRAW* - Padrão de Projeto Factory Method implementado na linguagem C Sharp. Fonte: Próprio autor

5.2 Técnicas de Modelagem de Ferramentas de Visualização de Redes Sociais e Complexas

Durante o desenvolvimento da presente pesquisa foram realizados estudos em ferramentas de visualização de redes sociais e complexas de código aberto com o objetivo de identificar os modelos e as técnicas de análise utilizadas na construção das mesmas.

Utilizamos como referencia o software de código aberto para análise e visualização de redes sociais e complexas *Gephi* versão 0.9.2, desenvolvida e mantida por estudantes da *University of Technology of Compiègne* com o uso paradigma OO e linguagem de desenvolvimento Java.

Analisamos também as biblioteca de visualização de redes: *iGraph*, com versões nas tecnologias *Python*, para uso no software estatístico *R*, e $C++$, para a presente pesquisa analisamos a versão 0.7.1 tecnologia $C++$; *jGraph*, desenvolvida na tecnologia *Java*, fornece funcionalidades de visualização e cálculos de propriedades dos redes atualmente se encontra na versão 0.9.0 e; *Graph Sharp*, biblioteca de criação e visualização de grafos, desenvolvida através da tecnologia $MIcrosoftCSharp$.

Após análise de documentações publicadas e código fonte das ferramentas e bibliotecas citadas na seção anterior, não identificamos de forma explícita os modelos de análise, uma vez que as ferramentas somente disponibilizam os códigos fontes e tutorias de utilização, não fornecendo documentos técnicos de modelagem das mesmas entretanto, os códigos dão indícios de que as ferramentas fizeram uso dos conceitos triviais OO, sugerindo o uso da UML. A exceção quanto ao fornecimento de documentos técnicos é a ferramenta $CSharp$, a mesma disponibiliza diagrama de classes do projeto de software apresentando as interfaces e algoritmos implementados conforme apresentamos nas Figuras 5.15 e 5.16.

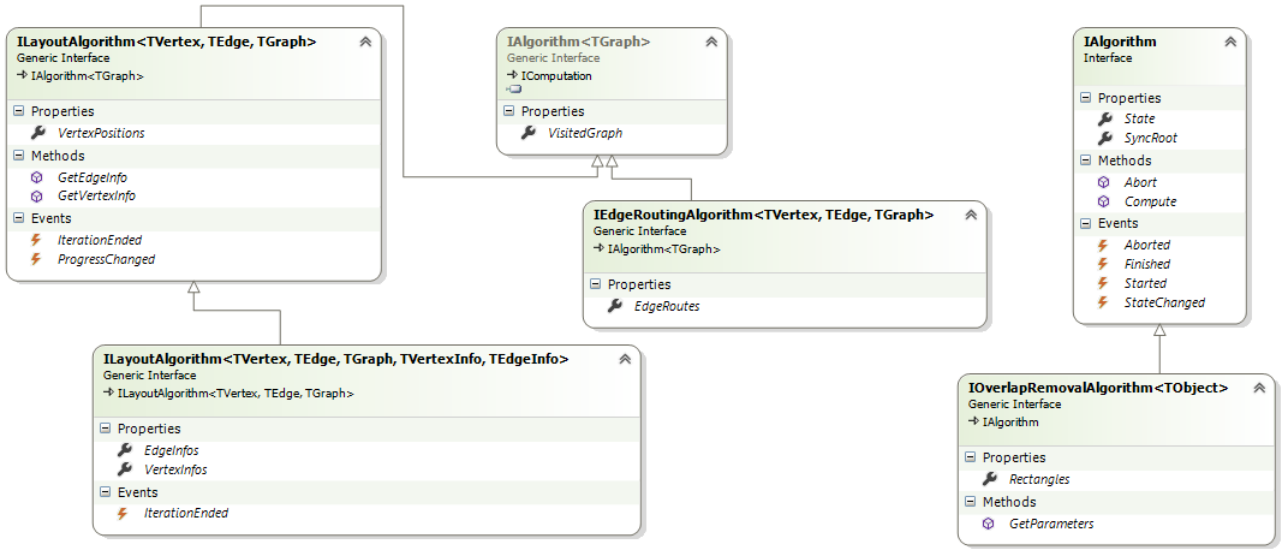


Figura 5.15: Graph Sharp Modelo de Interface. Fonte: Miesen (2018).

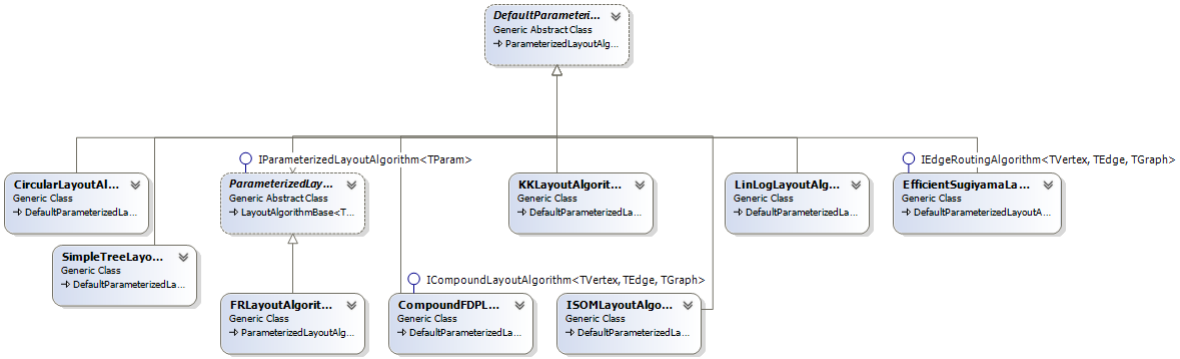


Figura 5.16: Graph Sharp Modelo de Algoritmos. Fonte: Miesen (2018).

A ferramenta *SC NET DRAW* em seu modelo de análise, conforme apresentado na Seção 5.1.4 deste capítulo, além de fazer uso dos conceitos OO tira proveito da sintaxe e semântica das redes sociais e complexas disponíveis no modelo *NET-UML*.

5.3 Representação Simbólica

Conforme discutido nesta pesquisa, o uso de imagens em contraponto ao uso de dados textuais, acelera a transformação de dados em informação e esta em conhecimento, possuir símbolos que representam conceitos pode ser útil em tarefas analíticas.

A presente pesquisa sugere representação simbólica inspirada no modelo GEO OMT e nos esteriótipos da UML. Enquanto os pictogramas do GEO OMT representam objetos georreferenciados, que possuem representação na superfície terrestre (Figura B.2), os esteriótipos, em forma de pictogramas, utilizados pela *NET-UML* se ocupam de representar elementos relacionados com a teoria e leiautes de distribuição de grafos, conforme apresentamos na Figura 5.17 e discutidos na Capitulo 4 Seção 4.1.5.

De acordo pesquisa de Revisão Sistemática, apresentada no Capítulo 2, não existem técnicas de modelagem especificas para construção de ferramentas de visualização de redes sociais e complexas, o que sugere a realização de trabalhos e contribuições nesta área.

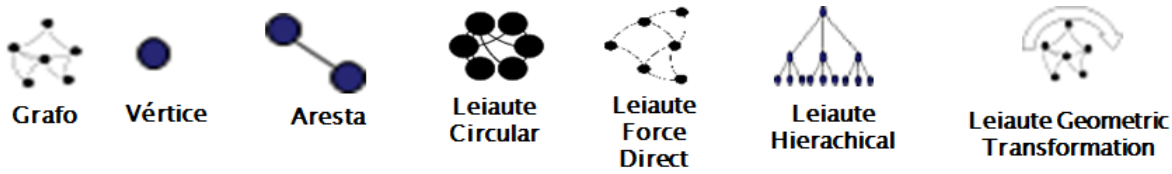


Figura 5.17: Pictogramas *NET-UML*. Fonte: Próprio autor.

5.4 Considerações do Capítulo

Apresentamos neste capítulo os resultados obtidos com uso da *NET-UML* para a construção de uma ferramenta de visualização de redes sociais e complexas bem como, a verificação das as técnicas de modelagem utilizadas para a construção das ferramentas e bibliotecas de visualização de redes sociais e complexas *Gephi*, *JGraph*, *iGraph* e *Graph Sharp*.

Discutimos com profundidade o modelo de análise da ferramenta *SC-NET DRAW* e observamos o quanto o uso dos casos de uso e diagramas de maquina de estados mandatórios sugeridos pela *NET-UML* potencializa a construção dos artefatos de análise e apoia a identificação das classes do projeto de software e uso de padrões de projetos associados.

Apresentaremos no próximo capítulo as considerações finais e contribuições desta pesquisa para á área de estudo de visualização de redes sociais complexas.

Capítulo Seis

Considerações Finais

A análise de redes sociais e complexas se apoia na verificação das propriedade das redes, por meio de métodos matemáticos e estatísticos e na observação dos eventos e comportamentos, verificados por meio da inspeção visual dos leiautes de distribuição.

Comportamentos como hierarquias não são percebidos facilmente através de cálculos matemáticos porém, podem ser rapidamente observados a partir da inspeção visual. Existe diversidade de leiautes de distribuição de rede e a escolha de qual utilizar dependerá do objetivo e contexto da pesquisa.

A visualização de informações em redes têm como objetivo apresentar os eventos e comportamentos contidos na mesma de acordo com o objeto de análise. Ao levar estas questões em consideração entendemos que existe oportunidade de criação de novos leiautes de distribuição de redes que sigam as premissas estéticas relacionadas com o domínio da visualização das redes, considerando a adição de atributos aos elementos do grafo (e.g. vértices e arestas) que gere informações importantes, apoiando o processo cognitivo de transformação de dados em informação e esta em conhecimento.

Neste contexto as ferramentas computacionais dotadas de funcionalidades de inspeção visual de redes, apoiam os pesquisadores nas tarefas analíticas, e como dito na seção anterior, descoberta de conhecimento.

A construção de tais ferramentas leva em consideração técnicas, metodologias e linguagens, genéricas aderentes aos paradigmas de desenvolvimento utilizados.

Existem sistemas de informação, que dado a complexidade do seu domínio, necessitam de recursos específicos para realização das tarefas de abstração e análise, para que a sintaxe e semântica envolvida no domino da aplicação seja corretamente considerada e implementada nas ferramentas computacionais (e.g modelo *GEO OMT*, utilizado para criação de sistemas de informações geográficas).

Não encontramos na literatura categorização explicita de algorítimos de distribuição espacial de grafos, diferentes autores sugerem organização dos algoritmos em grupos mas, não foi encontrado consenso ou referência bibliográfica que defina categoricamente como agrupar tais algoritmos.

Para o desenvolvimento desta pesquisa e proposição do modelo, foi sugerida categorização

de algoritmos a partir das premissas que os mesmos utilizam para posicionar os elementos do grafo no leiaute, está escolha se mostrou eficaz uma vez que ao comparar com as sugestões de categorização existente na literatura os resultados foram parecidos, conforme apresentamos no Capítulo 3, Seção 3.4.1.

A análise de ferramentas computacionais *Open Source*, de visualização de redes sociais e complexas e das técnicas utilizadas para a modelagem e desenvolvimento das mesmas, nos ajudou a perceber que, apesar das ferramentas analisadas não disponibilizarem documentação técnicas, disponibilizando apenas os códigos fontes, foram utilizadas técnicas e linguagens comuns ao paradigma OO (e.g. UML), conforme apresentamos no Capítulo 5, Seção 5.2.

O uso destas técnicas e linguagens não comprometem o desenvolvimento mas em alguns casos pode fazer com que questões conceituais passem desapercebidas. Ao verificar o código fonte da ferramenta *Gephi* observamos que o objeto aresta contempla atributos de um arco, sugerindo assim um erro de conceitual. Computacionalmente está questão é solucionada uma vez que foram criados atributos e métodos específicos para realização de operações em arestas e arcos.

Propomos na presente pesquisa, modelo icônico para construção de ferramentas que possuem funcionalidades de visualização de redes sociais e complexas, denominado *NET-UML*, através da proposição de elementos que apoiam a tarefa da análise e abstração dos conceitos sintáticos e semânticos envoltos na teoria dos grafos e leiautes de distribuição dos mesmos.

A *NET-UML* sugere o uso de elementos pictográficos para representação dos objetos do domino das redes bem como, oferece abstração inicial de elementos estruturais e comportamentais.

Como forma de testar o modelo proposto, apresentamos no Capítulo 5, Seção 5.1, a ferramenta *SC NET DRAW* construída a partir do uso da *NET-UML*.

Verificamos que a *NET-UML* se mostra adequada para uso no desenvolvimento de ferramentas OO, que discutam visualização de redes sociais e complexas uma vez que possibilita ao desenvolvedor capturar aspectos sintáticos e semânticos do domino da aplicação além de ser um modelo que pode ser estendido, possibilitando a adição de novos elementos relacionados ao domínio das aplicações de visualização de redes sociais e complexas.

6.1 Contribuições

As contribuições que esta pesquisa traz são (1) proposição de modelo computacional para construção de ferramentas especialista em visualização de redes sociais e complexas, a partir da abstração de conceitos relacionados a teoria de redes e algoritmos de distribuição, (2) estudo e categorização de algoritmos de distribuição de redes a partir de método estatístico e categorização existente na literatura, (3) ferramenta *Open Source* de visualização de redes sociais e complexas capaz de criar redes, a partir de arquivos gerados em softwares como *Pajek* e *Gephi*, e prover inspeção visual a partir de algoritmos de distribuição de grafos largamente utilizados para análise de redes sociais e complexas e (4) biblioteca de algoritmos de visualização de redes implementados na linguagem *CSharp*.

6.2 Atividades Futuras de Pesquisa

Como atividades futuras da pesquisa podemos citar:

- Estudo para proposição de padrão de projetos para criação de algoritmos de visualização de redes sociais e complexas - acreditamos que existe oportunidade para proposição de padrão de projeto dado os problemas técnicos comuns identificados no estudo e implementação dos algoritmos discutidos neste trabalho.
- Estudo para proposição de framework para construção de ferramentas computacionais do domino das redes sociais e complexas.
- Implementação funcionalidade de impressão das imagens de leiautes de grafos gerada pela ferramenta *SC NET DRAW* - esta funcionalidade está presente em todas as ferramentas de visualização de rede open source e propicia ao analista colocar as imagens dos leiautes de distribuição nos artefatos de pesquisa.
- Implementação funcionalidade de impressão de imagem da matriz de adjacência de um grafo - possibilitar a análise da dimensão fractal das redes geradas.
- Incorporação dos leiautes de distribuição discutidos neste trabalho nas ferramenta *SCNTools*, desenvolvida na linguagem Guará Script.

Apêndice A

Linguagem Unificada de Modelagem - UML

O paradigma de desenvolvimento de software Orientado a Objetos (OO) continua sendo referência para o construção de soluções computacionais, uma vez que proporciona elevado grau de abstração, encapsulamento e reutilização de objetos entre os projetos, possibilitando assim, a criação de modelos de software próximos do mundo real.

A UML é uma linguagem originada a partir da união dos métodos, OOSE, OMT e método de Booch que a partir da padronização dos conceitos, notações, símbolos e diagramas discutidos por estes métodos apoia a criação de modelos de software OO.

O objetivo deste apêndice é apresentar os conceitos de OO e elementos da UML utilizados para construção do modelo *NET-UML* conforme apresentamos no Capítulo 4.

A.1 Orientação a objetos

Conforme dito na seção anterior a OO se destaca por abstrair elementos do mundo real como uma coleção de objetos que possuem características e comportamentos distintos e se comunicam para realizar o objetivo do sistema.

Dentre os conceitos da OO, destacamos objetos, classes, e herança pois estes, foram utilizados para a composição da *NET-UML*.

Objeto é uma abstração concreta ou abstrata de uma entidade do mundo real (e.g. vértice, aresta, arco ou um grafo) e possui três propriedades: estado, comportamento e identidade. Classe pode ser considerada como o molde de um objeto, possui a especificação das propriedades, comportamento, relacionamento e semântica que o objeto terá ao ser instanciado (Cardoso, 2015; Borges, 1997b; Larman, 2007; Wazlawick, 2011).

Herança é o mecanismo que permite o compartilhamento de propriedades e comportamento entre classes a exemplo da família de leiautes de distribuição de grafos dirigidos por força que compartilham características de distribuição dos elementos da rede através de princípios físicos conforme discutido no Capítulo 3, Seção 3.4.2.3 e Capítulo 5, Seção 5.1.4 (Cardoso, 2015; Borges, 1997b; Larman, 2007; Wazlawick, 2011).

A.2 UML - Unified Modeling Language

Com o surgimento do paradigma do desenvolvimento de software OO sugiram também técnicas, metodologias e linguagens para criação de artefatos utilizados no processo de construção de software.

A década de 1990 foi marcada pelo surgimento de várias metodologias e ao analisar cada uma destas, foi observado a existência de elementos comuns e outros que se completavam, uma vez que conseguiam suprir a deficiência um das outro.

A união das metodologias propostas por Ivar Jacobson (método OOSE – Object Oriented Software Engineering), James Rumbaugh (modelo OMT - Object Modelling Technique) e Grady Booch (método de Booch) provocou o surgimento da linguagem UML (Unified Modeling Language) (Tacla, 2007).

A UML não é uma metodologia e sim uma linguagem que objetiva a padronização dos modelos, diagramas e notações propostos pelos métodos citados anteriormente necessários para a execução de um projeto de software OO. A UML é mantida pelo OMG (Object Management Group) teve sua primeira versão rascunho em 1995 e atualmente se encontra na versão 2.5 publicada em 2015.

A UML propõem como fases para o desenvolvimento de software a análise de requisitos, análise, projeto, implementação e testes. Na análise de requisitos são mapeadas as funções do software através dos casos de usos e dos atores que utilizaram ou possuem interesse nestas funções. A fase de análise se caracteriza pela construção das primeiras abstrações semânticas e conceituais, são levantadas as classes, objetos e seus relacionamentos. Na fase de projeto serão incorporadas as questões técnicas que não foram tratadas na fase anterior e com isso novas classes e seus relacionamentos surgem agora com a responsabilidade não mais de abstrair questões relacionadas ao domínio da aplicação e sim com questões de acesso a bases de dados, interação com sistemas externos dentre outras questões técnicas. Na fase de implementação as classes mapeadas na fase de projeto são transformadas em códigos em linguagem de programação. Na fase de testes, são realizados os testes unitários, de integração, de sistema e testes de aceite (Presman, 2011; Wazlawick, 2011; Borges, 1997b; Tacla, 2007).

Por ser uma linguagem visual, a UML propõem diversos diagramas e mecanismos de extensão. Neste apêndice apresentaremos o mecanismo de extensão esteriótipo e os diagramas da UML porém, com maior profundidade o digrama estrutural de classes e os digramas comportamentais de casos de uso e máquina de estados pois a *NET UML* propõem especializações para estes diagramas (Omg-group, 2015; Presman, 2011; Wazlawick, 2011; Borges, 1997b; Tacla, 2007).

Apresentamos na Figura A.1 a estrutura de diagramas da versão atual da UML e sua divisão em diagramas de estruturas e de comportamento (Omg-group, 2015).

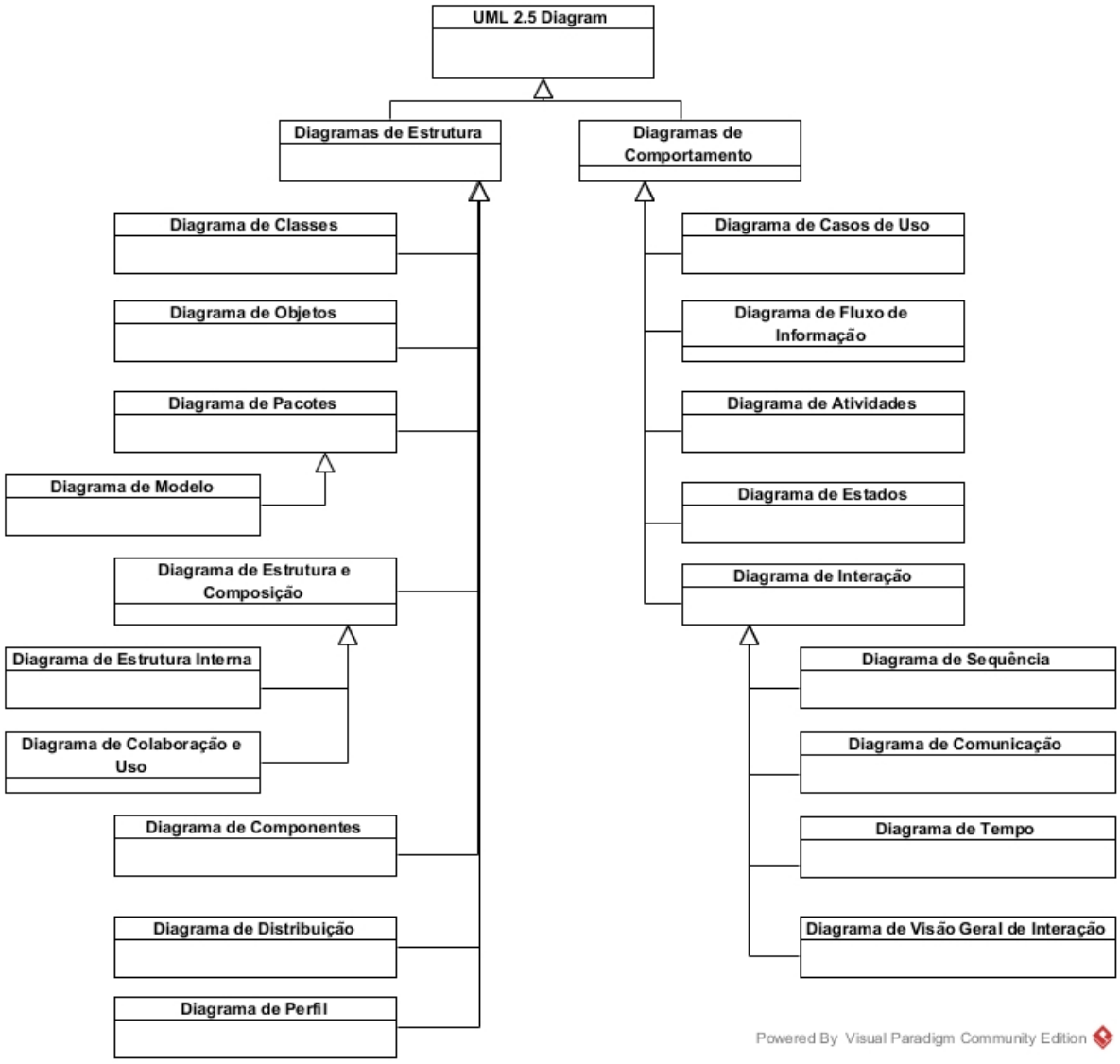


Figura A.1: Diagramas da UML versão 2.5. Fonte: (Omg-group, 2015).

Discutiremos a seguir o mecanismo de extensão estereótipos e os diagramas da UML. Para melhor entendimento utilizaremos como estudo de caso, um hipotético software de criação de redes sociais e complexas.

A.2.1 Estereótipos

Estereótipo é um mecanismo da UML que possibilita estender a linguagem através da criação de símbolos utilizados para classificar, adaptar e personalizar elementos.

Graficamente os esteriótipos são representados por um nome entre os símbolos « <Nome estereótipo>> " (e.g. <<Vértice>>, <<Aresta>>, <<Arco>> e <<Grafo>>) (Larman, 2007).

Tendo como inspiração o modelo *GEO-OMT*, a *NET-UML* propõem a criação de esteriótipos específicos a partir de representação simbólica de objetos relacionados com a teoria dos grafos e leiautes de distribuição/visualização de grafos.

A.2.2 Casos de Uso

Iniciamos o desenvolvimento de um software entendo o seu objetivo e quais as funções o mesmo deve executar para que este objetivo seja alcançado, esta tarefa é realizada na etapa de análise de requisitos e o têm como suporte da UML o diagrama de casos de uso. Os casos de uso são narrativas textuais utilizadas para descrever as funções do sistema, suas delimitações, atores responsáveis e o fluxo de operações necessárias para a execução (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.2 exemplo de diagrama de casos de uso com as macro funcionalidades de criação de grafo e leiaute de visualização.

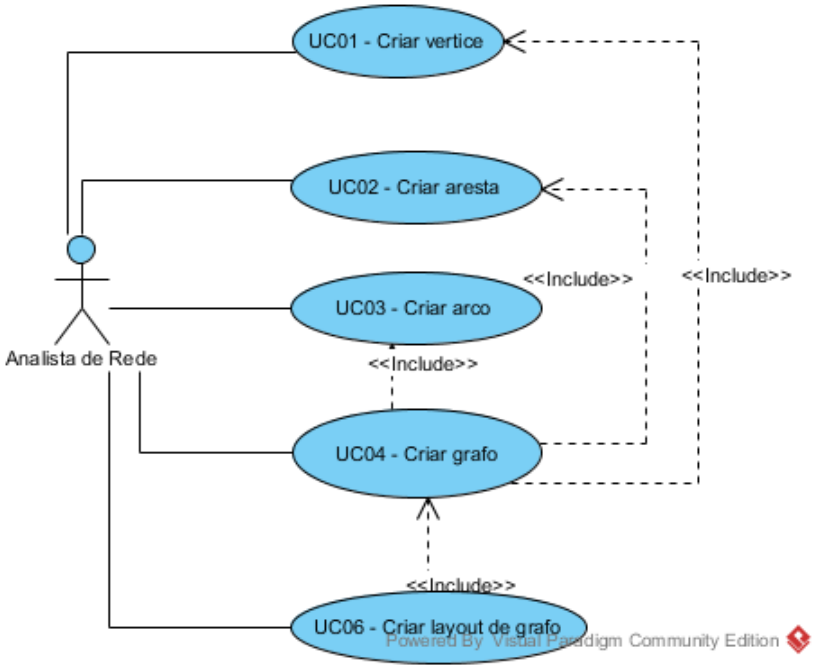


Figura A.2: Diagramas de Casos de Uso. Fonte: Próprio autor

A.2.3 Diagrama de Classes

O diagrama de classes é um artefato estático que apresenta as classes, interfaces e relacionamentos obtidos a partir de outros artefatos de software gerados na fase de análise de requisitos (e.g casos de usos e especificação de requisitos) (Larman, 2007; Tacla, 2007). A seguir, discutiremos o digrama de classes, tendo como insumo os requisitos extraídos do diagrama de casos de uso que apresentamos na Figura A.2.

Conforme dito anteriormente, a classe é um elemento que dará origem aos objetos. A sua estrutura possui três partes: nome, atributos e métodos. Os atributos e métodos da classe possuem modificadores de visibilidade que indicam se o elemento terá visibilidade publica (+), privada (-) ou protegida (#) (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.3 a estrutura de uma classe nomeada de Vértice, que possui os atributos codigo, nome, grau, posicaoX, posicaoY e posicaoZ, métodos exibirGrau, exibirNome e atribuirPosicao, cada um dos elementos possui seu respectivo modificador de visibilidade.

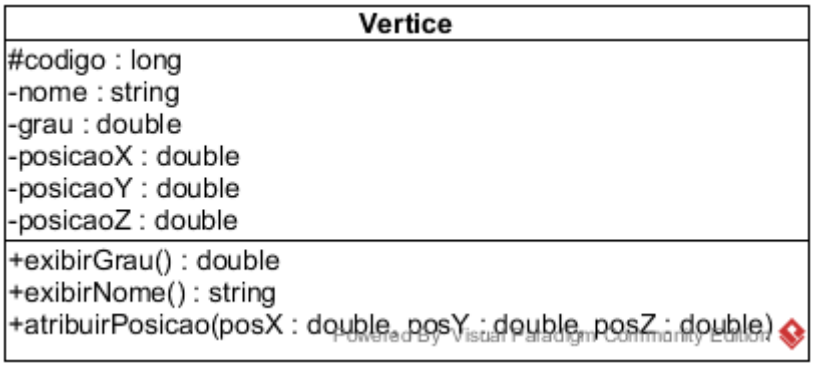


Figura A.3: Elemento classe proposto pela UML. Fonte: Próprio autor

Na UML, a associação entre as classes é definida pelos relacionamentos que são classificados como associação, agregação, composição, generalização e dependência. Associação indica que objetos de tipos diferentes estão ligados. Agregação é um tipo especial de associação entre classes cujo o objetivo é representar relacionamentos de pertinência ("parte de"), quando um objeto ou mais fazem parte de um outro objeto de tipo diferente (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

No relacionamento de composição a existência dos objetos que compõem o relacionamento é mandatória já a dependência e um tipo de relacionamento que indica que mudanças em um dado objeto podem causar mudanças no objeto relacionado enquanto a generalização é o relacionamento que explicita o mecanismo de herança na OO, "o objeto B é um tipo de

objeto A", os relacionamentos entre classes possuem nome, cardinalidade e navegabilidade (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.4 o diagrama de classes que dará origem a um software de criação de redes, a partir da abstração de conjuntos que são compostos por elementos, que representam vértices, e a relação entre os elementos dão origem as arestas. Os vértices e as arestas oriundos dos conjuntos formarão grafos que representam redes.

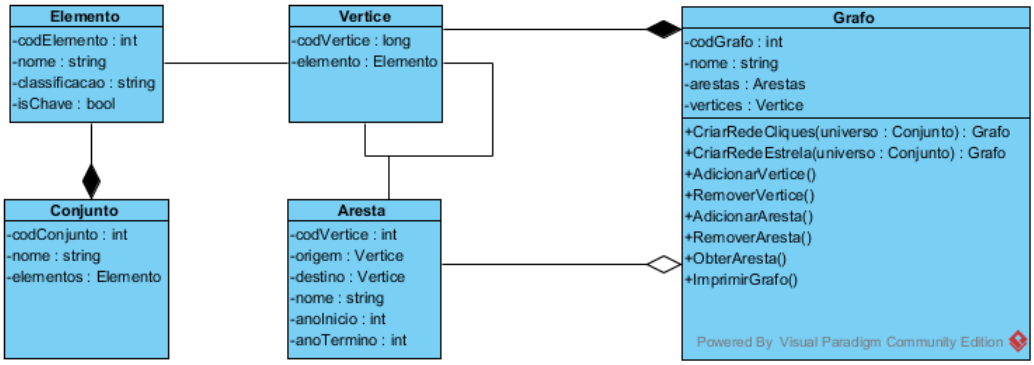


Figura A.4: Diagrama de classes UML do software de criação de redes. Fonte: Próprio autor

A.2.4 Diagrama de Objetos

Diagrama que apresenta um momento de execução do software, ou seja, uma instância hipotética das classes e seus relacionamentos dando uma ideia de como os dados serão mantidos nas entidades do sistema (Larman, 2007; Tacla, 2007).

Apresentamos na Figura A.5 o diagrama de objetos do software de criação de redes com a instância do artigo científico *Clique Approach for Networks* e sua coautoria, representado pela abstração de que um artigo cientifico é um conjunto, cujo os elementos são os autores que ,quando pensamos em redes de colaboração cientifica, são transformados em vértices cuja conexão, que dará origem as arestas, e dada a partir do momento que fazem parte do mesmo conjunto.

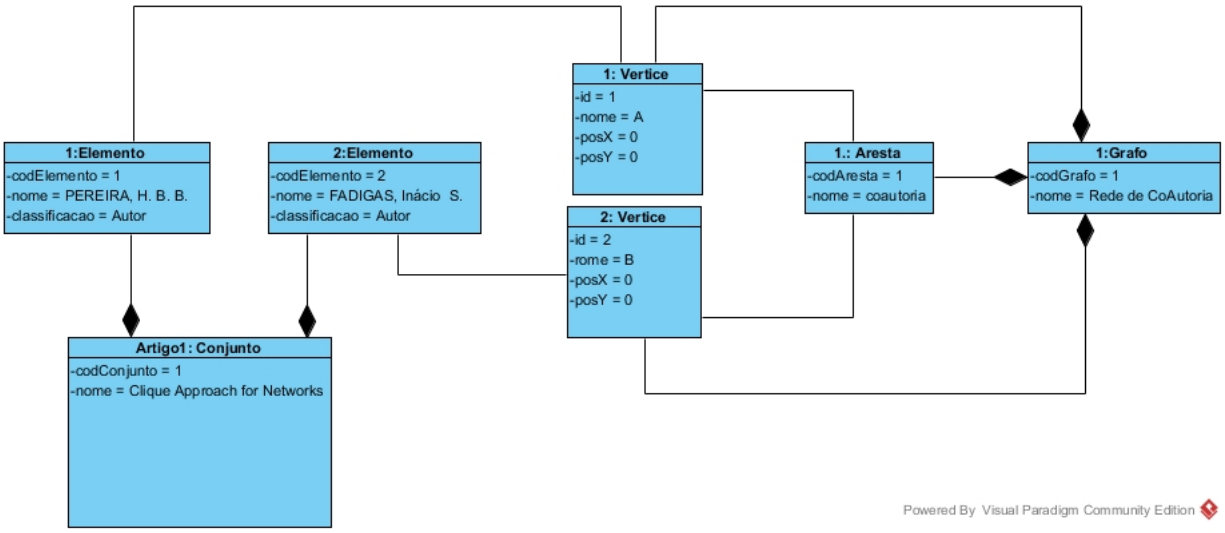


Figura A.5: Diagrama de objetos UML do software de criação de redes. Fonte: Próprio autor

A.2.5 Diagrama de Máquina de Estados

Diagrama que apresenta o comportamento de um objeto do software através da transição de estados. Seu uso e indicado para objetos complexos e possui como elementos: o estado ou situação em que o objeto se encontra em um dado momento; o evento que irá provocar a mudança do estado; ação, tarefa atômica executada pelo objeto que não pode ser interrompida até o final da sua execução; atividade, tarefa que se diferencia da ação por poder ser interrompida antes do final da sua execução e; condição de guarda, expressão lógica quem em sendo verdadeira provoca a mudança de estado do objeto (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.6 um digrama de máquina de estados do objeto grafo de um sistema de criação de redes sociais e complexas. Neste diagrama, são apresentados os estados, a sequência de mudança e os eventos que provam a transição de estados do objeto conforme descrito a seguir.

- Criado - Estado primário de um grafo cuja os vértices e arestas/arcos (caso existam) foram definidas.
- Apresentado - Estado que exibe os elementos do grafo no leiaute podendo ocorrer primariamente após o estado de Criado. Uma vez apresentado o grafo pode ser Salvo, Distribuído ou ter seu ciclo de vida encerrado.
- Distribuído - Estado que representa a aplicação dos algoritmos de distribuição espa-

cial e manipulação dos elementos do grafo. Ema vez Distribuído o grafo volta a ao estado de Apresentado.

- Salvo - Neste estado as propriedades do grafo são gravadas para continuidade da inspeção visual. Normalmente este estado sucede as operações de manipulação e distribuição de um grafo precedendo a finalização do ciclo de vida do objeto.
- Excluído - Após ser apresentado ou distribuído o grafo pode ser excluído do ambiente de visualização e com isso ter seu ciclo de vida encerrado.

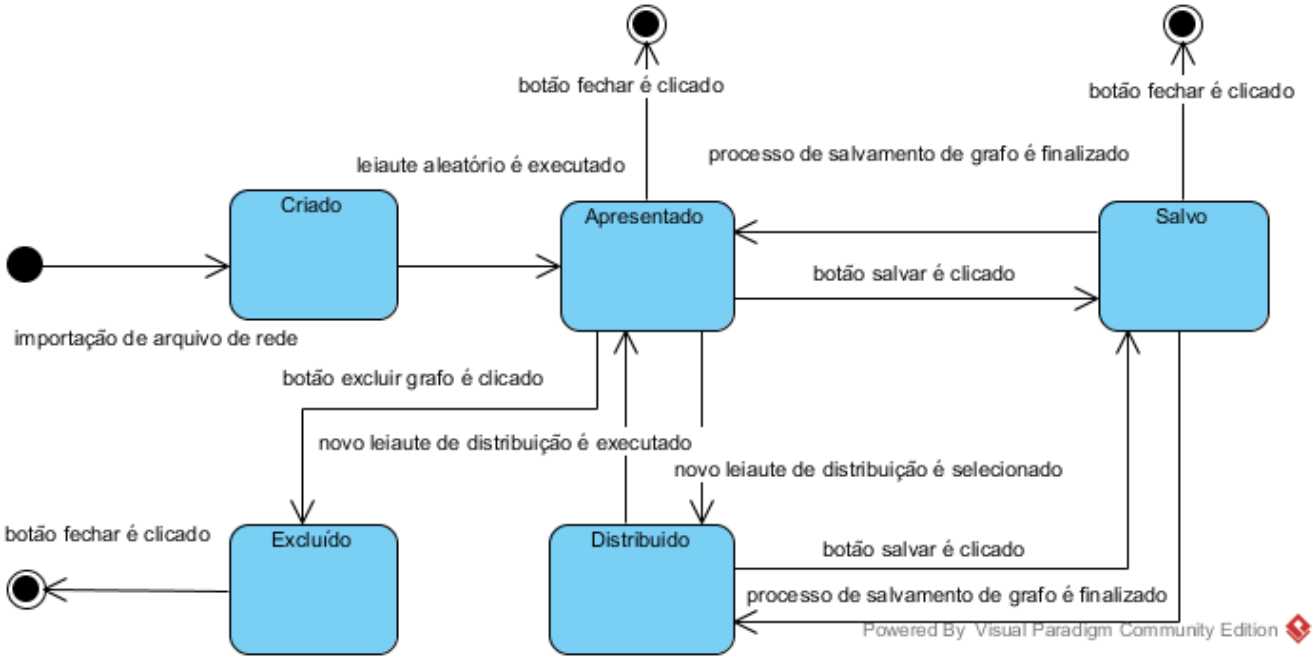


Figura A.6: Diagrama de máquina de estados da UML para o objeto grafo do software de criação de redes. Fonte: Próprio autor

A.2.6 Diagrama de Sequência

A função do diagrama comportamental de sequência é apresentar os eventos do software para um cenário especificado em um caso de uso, demonstrando a ordem com que os eventos acontecem indicando como ocorrerá as trocas de mensagem entre os objetos do sistema (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.7, digrama de sequência do software de criação de redes sociais e complexas com a sequência de eventos necessários para a execução do caso de uso UC04 - Criação de grafo.

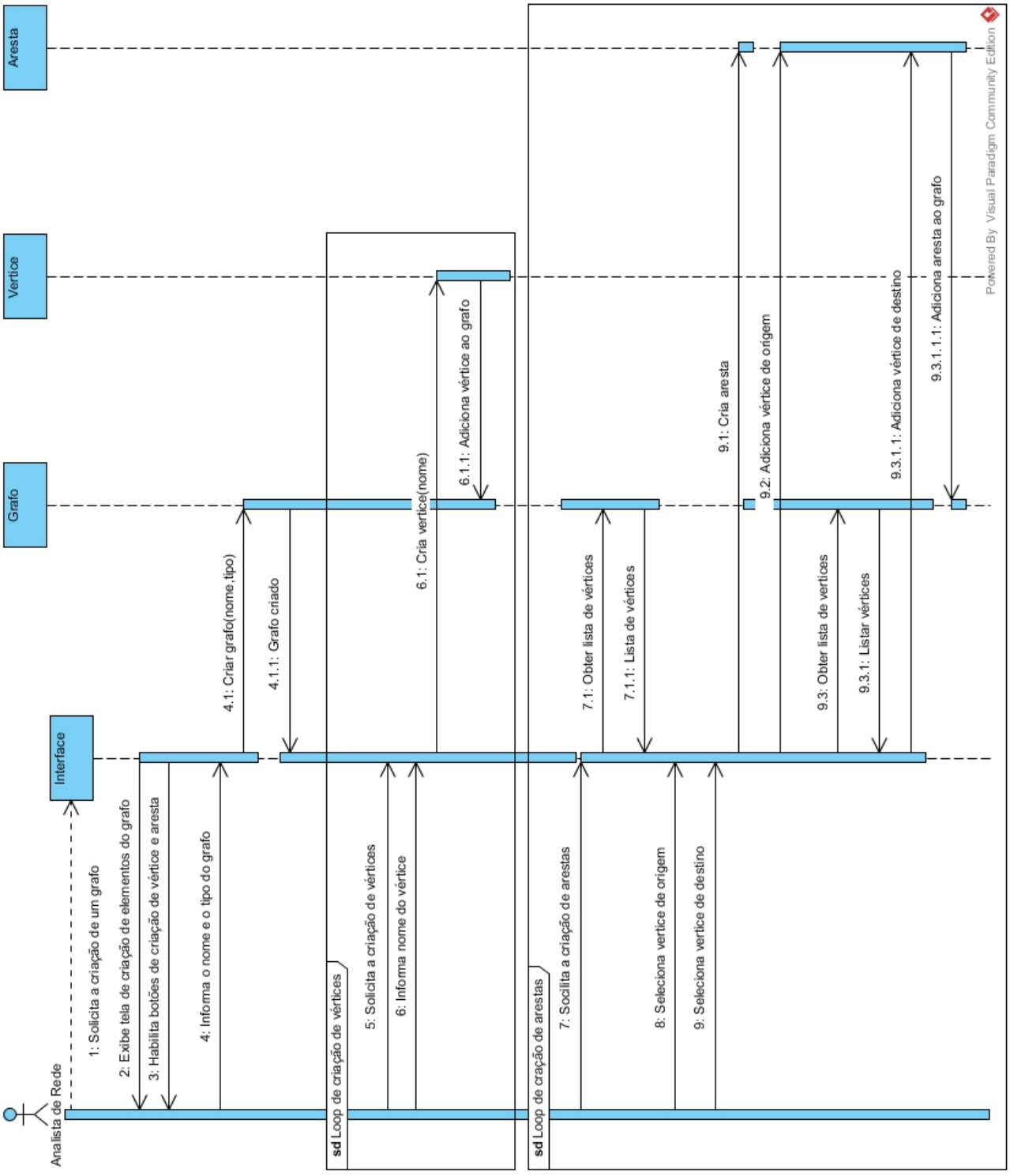


Figura A.7: Diagrama de sequência da UML para a funcionalidade de criação de grafos do software de criação de redes. Fonte: Próprio autor

A funcionalidade de criação de grafos têm como ator o analista de rede e os objetos

interface, grafo, vértice e aresta. O analista de rede acessa a interface do sistema e clica no botão nova rede, em seguida o sistema habilita os controles de registro do nome, tipo do grafo, criação de vértice e arestas. Após informar o nome e o tipo do grafo o analista de rede clica no botão salvar, em seguida o ator deve criar os vértices da rede informando o nome do mesmo e clicando em seguida no botão gravar vértice, esta operação deve ser repetir até que todos os vértices da rede sejam criados.

Uma vez criados os vértices o analista de rede cria as arestas informando os vértices de origem e de destino que formaram a conexão, clicando em seguida no botão gravar aresta, esta operação deve ser repetir até que todos os vértices da rede sejam criados. Finalizada a adição dos vértices e arestas o analista de redes conclui a criação da rede.

A.2.7 Diagrama de Atividades

Diagrama que apresenta de forma detalhada as atividades executadas por um método ou caso de uso. Conforme discutido na apresentação do diagrama de estado, somente métodos ou caso de usos complexos devem possuir diagramas de atividades (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.8 o diagrama de atividades do caso de uso criar grafo.

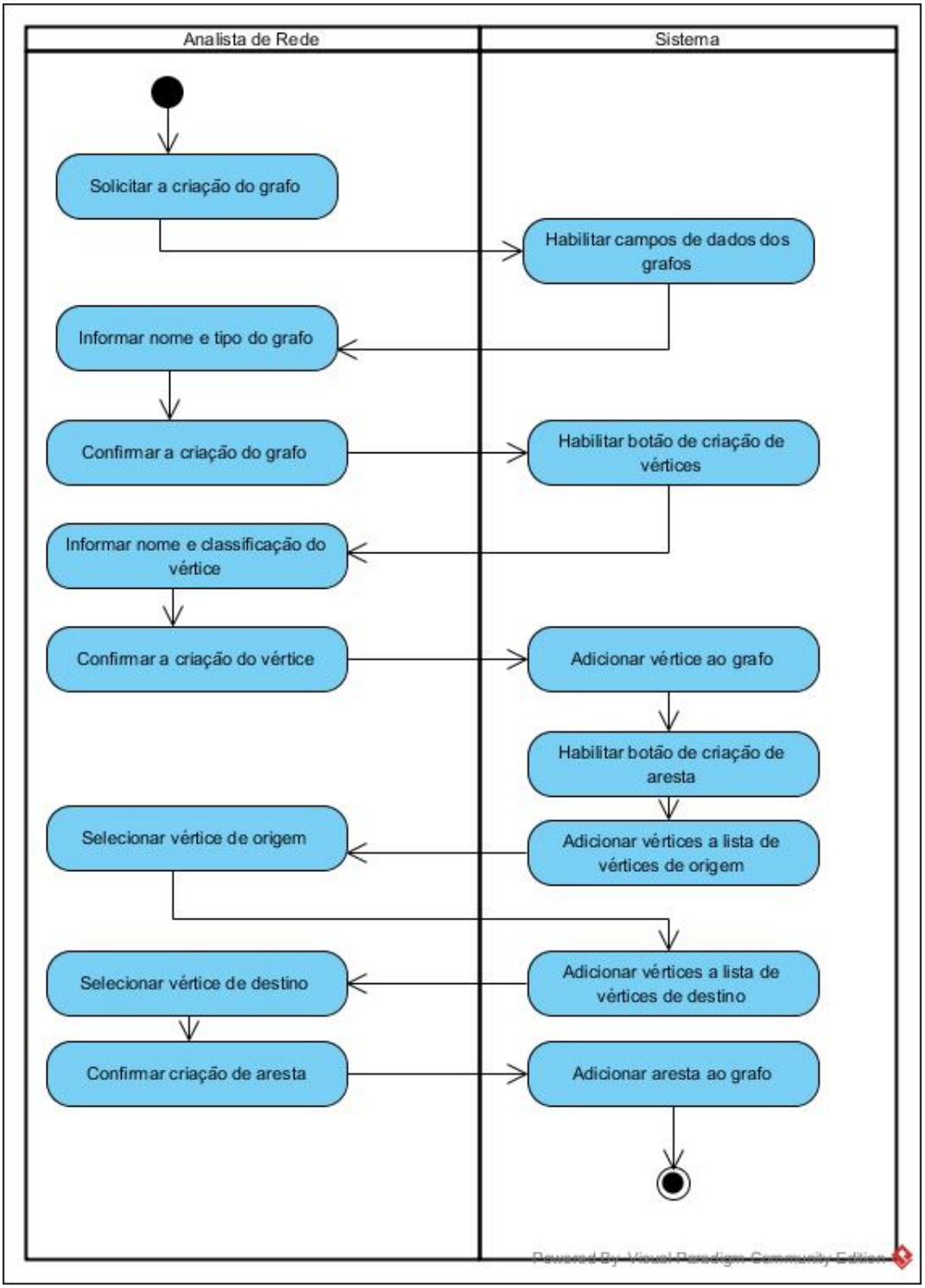


Figura A.8: Diagrama de atividade da UML da funcionalidade de criação de grafos do software de criação de redes. Fonte: Próprio autor

A.2.8 Diagrama de Pacotes

Diagrama estrutural da UML que apresenta agrupamento de elementos do projeto de software, como classes ou casos de uso, que juntos têm algum sentido. Os pacotes podem estabelecer relacionamento de dependência com um outro pacote para que seja possível executar as tarefas previstas pelos mesmo (Larman, 2007; Tacla, 2007; Wazlawick, 2011).

Apresentamos na Figura A.9 diagrama de pacotes onde as classes de interface, domínio conceitual, leiaute de distribuição de grafos e classes de funções genéricas são agrupadas em diferentes pacotes cujo os relacionamentos de dependência são evidenciados.

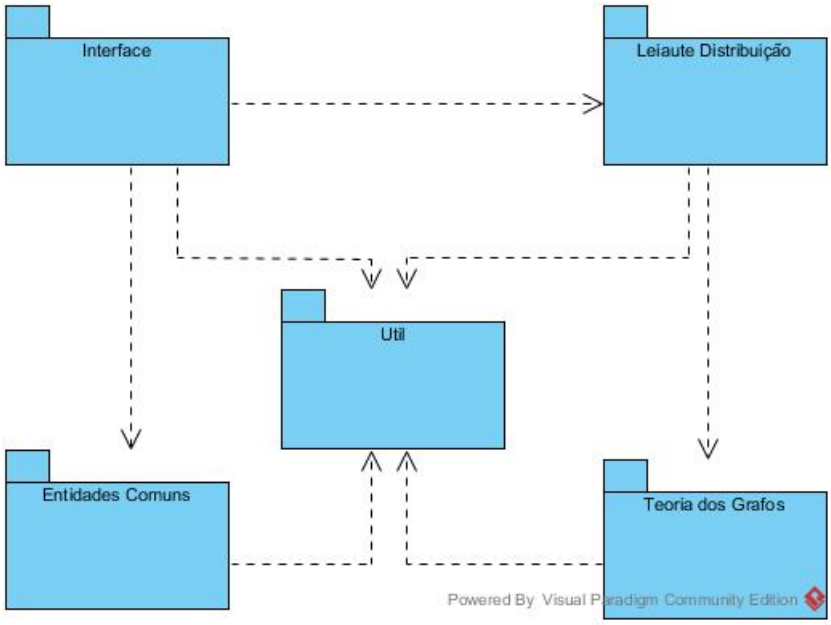


Figura A.9: Diagrama de pacotes da software de criação de redes. Fonte: Próprio autor

A.2.9 Diagrama de Componentes

Diagrama estrutural de projeto de software que apresenta visão estática do mesmo, possui forte ligação com a tecnologia utilizada para o desenvolvimento. É um artefato de grande importância pois destaca os componentes de forma individual dando a visão de reutilização (Larman, 2007; Tacla, 2007). Segundo Larman (2007), o conceito de componentes na UML possui interpretação dúbia uma vez que tanto classes quanto componentes pode ser utilizados para modelar a mesmo elemento.

Apresentamos na Figura A.10 diagrama de componentes do software de criação de redes de acordo com a função que cada componente desempenha no sistema e os relacionamentos

de associação que os mesmos possuem.

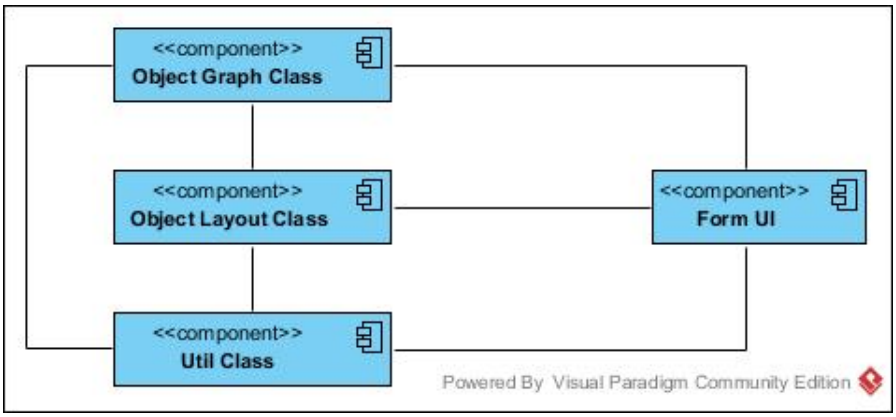


Figura A.10: Diagrama de componentes do software de criação de redes. Fonte: Próprio autor

A.2.10 Diagrama de Implantação

Larman (2007) define diagrama de implantação como artefato do projeto de software que apresenta os elementos da arquitetura do software e a comunicação entre os elementos físicos. Têm como elemento básico o nó, que pode ser um dispositivo, como um recurso computacional físico (e.g computador, telefone) ou um nó de ambiente de execução, um recurso computacional de software (e.g. sistema operacional, máquina virtual, navegador web).

Apresentamos na Figura A.11 o diagrama de implantação do software de criação de redes. O software foi concebido para ser executado am ambiente *desktop* desta maneira o diagrama de implantação possui somente um nó, representando a maquina do usuário e a indicação do sistema operacional que suporta o software. Dentro do nó destacamos os componentes da arquitetura do software ,de acordo o modelo MVC, a camada *Model* composta pelos componentes *Object graph*, *Object layout* e *Utils*, a camada *Controller* e a camada *View* que a interface do software e possui relacionamento com o componente *Open GL* através da biblioteca o *Tao Framework*.

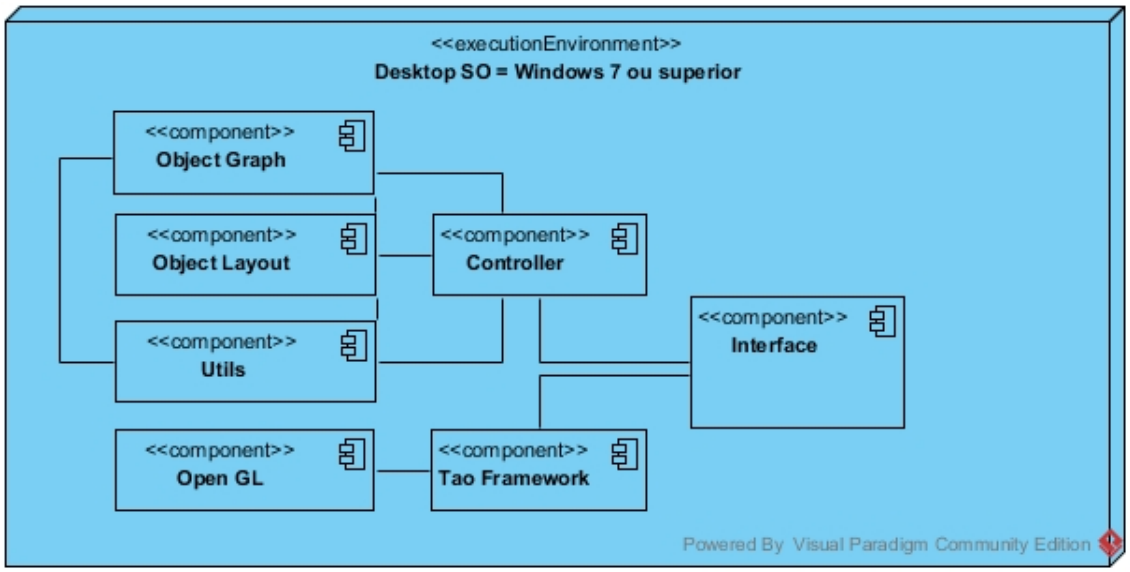


Figura A.11: Diagrama de implantação do software de criação de redes. Fonte: Próprio autor

A.3 Engenharia de Software - Projeto Arquitetural

Arquitetura de software é um modelo conceitual que da suporte a transição da fase de levantamento de requisitos para a fase de implementação do software (Presman, 2011).

Apresenta visão geral do modelo do projeto de software com base em diferentes visões arquiteturais para ilustrar os diversos pontos e aspectos da solução computacional, destaca as informações necessárias para o controle das atividades, possibilitando visão de macro dos requisitos funcionais e não funcionais, capturando e transmitindo as decisões do ponto de vista do projeto de software e têm como foco apresentar e discutir os elementos que compõem a solução computacional como os requisitos de segurança, de desempenho e de integrações, além de contemplar padrões de software, componentes, plataformas de desenvolvimento, hardware, software, além de servidores de aplicação e de banco de dados, sistemas operacionais, *frameworks* e *APIs* (Presman, 2011).

Apresentamos na Figura A.12 modelo de arquitetural de projeto de software em camadas que tem como objetivo destacar os elementos que farão parte do projeto de software.

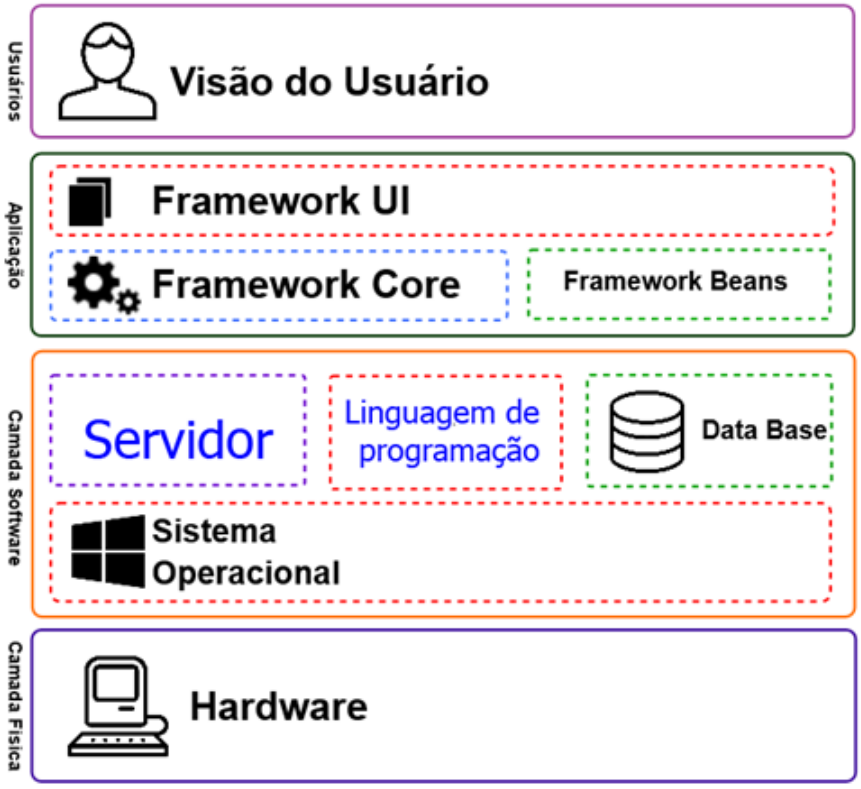


Figura A.12: Diagrama de Projeto Arquitetural de Software por Camadas. Fonte: (Software Senai Cimatec Team, 2015)

A.4 Considerações do Apêndice

Apresentamos neste apêndice os conceitos e elementos da OO e UML que guiaram a proposição da *NET-UML*.

Dentre estes conceitos destacamos: esteriótipos, utilizados para criação de representação categórica dos objetos; classes, utilizado para definição da sintaxe dos objetos relacionados com a teoria e algoritmos de distribuição de grafos e; casos de uso e diagrama de maquina de estados, relacionados com a semântica e comportamentos associados ao domínio das aplicações de redes sociais e complexas.

Discutimos também os conceitos relacionados a projeto arquitetural de software pois este foi utilizado para construção da ferramenta *SC NET DRAW* conforme apresentado no Capitulo 5, Seção 5.1.1.

Apêndice B

Geographic Object Modeling Technique - GEO OMT

Borges (1997a) propôs no trabalho intitulado Modelagem de Dados Geográficos: Uma Extensão do Modelo OMT para Aplicações Geográficas, uma extensão do modelo OMT proposto por Rumbaugh et al. (1991) para uso no desenvolvimento de modelos de sistemas de informação geográficas, se ocupando apenas de apoiar a criação do modelo de classe da etapa de análise a abstração inicial das entidades do mundo real.

A proposta do modelo *GEO-OMT* se a semelha à proposta do modelo *NET-UML* uma vez que ambos se ocupam de fornecer ferramental para apoiar a etapa de inicial de abstração dos objetos e comportamentos que cada um domínios de aplicação tratam.

Apresentamos neste apêndice os conceitos do modelo *GEO-OMT* que inspiraram a construção da *NET-UML* e no Capítulo 4, Seção 4.2 comparação entre os modelos.

B.1 Meta Modelo

O modelo GEO OMT têm como principais conceitos classe, relacionamentos e restrições de integridade espacial. Se caracteriza por propor representação simbólica que apoia a abstração do objeto geográfico modelado. Neste modelo as classes são divididas em convencionais, que modelam objetos que não possuem propriedades geométricas, e georreferenciadas, que modelam objetos com representação espacial (Borges, 1997a).

Apresentamos na Figura B.1 o meta modelo da GEO OMT destacando a separação das classes convencionais, das classes georreferenciadas.

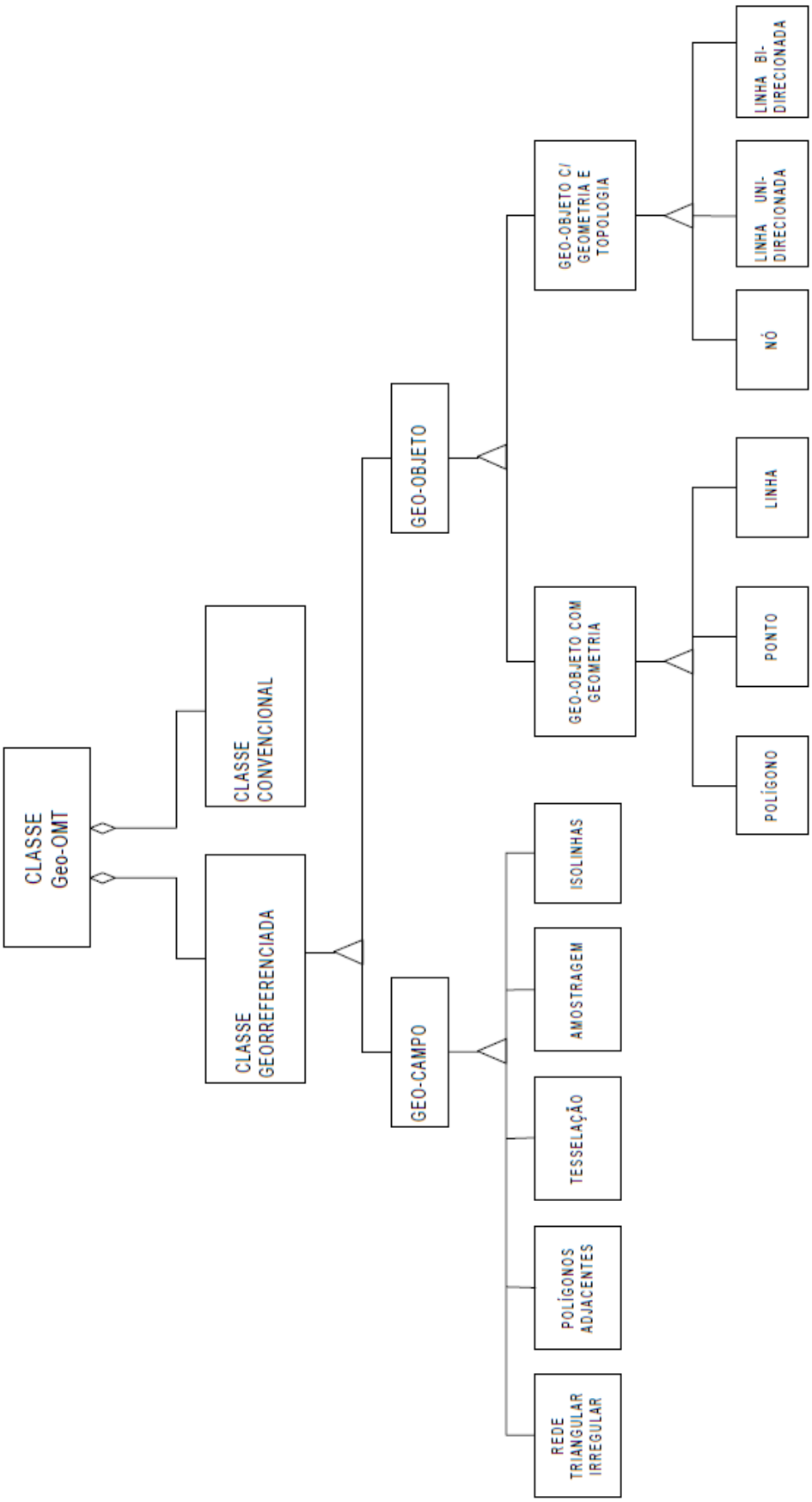


Figura B.1: Meta modelo GEO OMT. Fonte: Borges (1997a).

B.2 Representação Simbólica

O GEO OMT se destaca por apresentar representação simbólica e separação de tipos de atributos (e.g. espacial e alfanumérico) para as classes georreferenciadas evidenciando assim, a semântica dos elementos espaciais contida neste domínio de aplicação (Borges, 1997a).

Apresentamos na Figura B.2 a representação das classes convencionais e georreferenciadas com destaque para os pictogramas utilizados para representação simbólica das classes georreferenciadas.

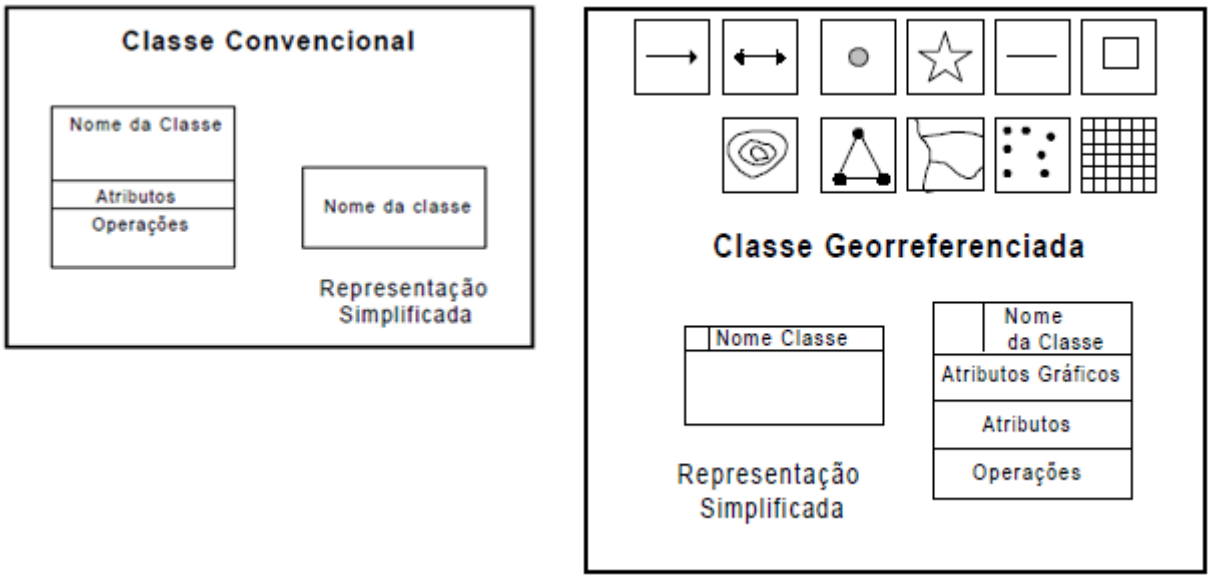


Figura B.2: Classes básicas do modelo GEO-OMT. Fonte: Borges (1997a).

As classes georreferenciadas são divididas em classes Geo-Campo, que abstraem objetos distribuídos continuamente pelo espaço como tipo de solo, topografia e teo de minerais, e Geo-Objeto, que abstraem objetos geográficos individualizáveis como ruas, lotes e rios. Apresentamos na figura 3.8 exemplos de classes Geo-Campos e na Figura B.3 exemplos de classes Geo-Objetos (Borges, 1997a).

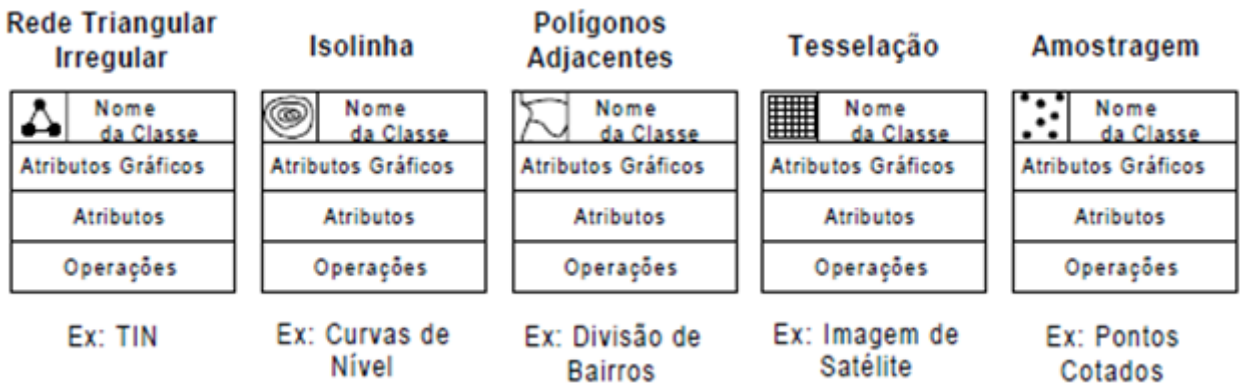


Figura B.3: Classes Geo-Campos do modelo GEO-OMT. Fonte: Borges (1997a).

Figura B.4: Classes Geo-Objetos do modelo GEO-OMT. Fonte: Borges (1997a).

B.3 Relacionamentos

Borges (1997a) propõem como relacionamentos entre o objetos no modelo GEO-OMT a associações simples, relações topológicas de rede, relações espaciais, agregação e agregação espacial conforme descrito a seguir.

- As associações simples - Relacionamentos estruturais entre diferentes tipos de classes sejam elas convencionais ou georreferenciadas.
- Relações espaciais - Objetivam explicitar o relacionamento espacial entre as classes representando relações topológicas, métricas, fuzzy e ordinais.
- Relações em rede - Obviamente, são relacionamento que formam estruturas em rede.

- Relação de agregação - Apresenta a composição de objetos formados por outros objetos, ocorrendo entre classes Convencionais, classes Georreferenciadas e entre classes Convencionais e Georreferenciadas.
- A agregação espacial é um tipo especial de agregação, representa os relacionamentos topológicos entre os objetos do modelo.

Apresentamos na Figura B.5 os tipos de relacionamentos entre objetos do modelo GEO OMT.

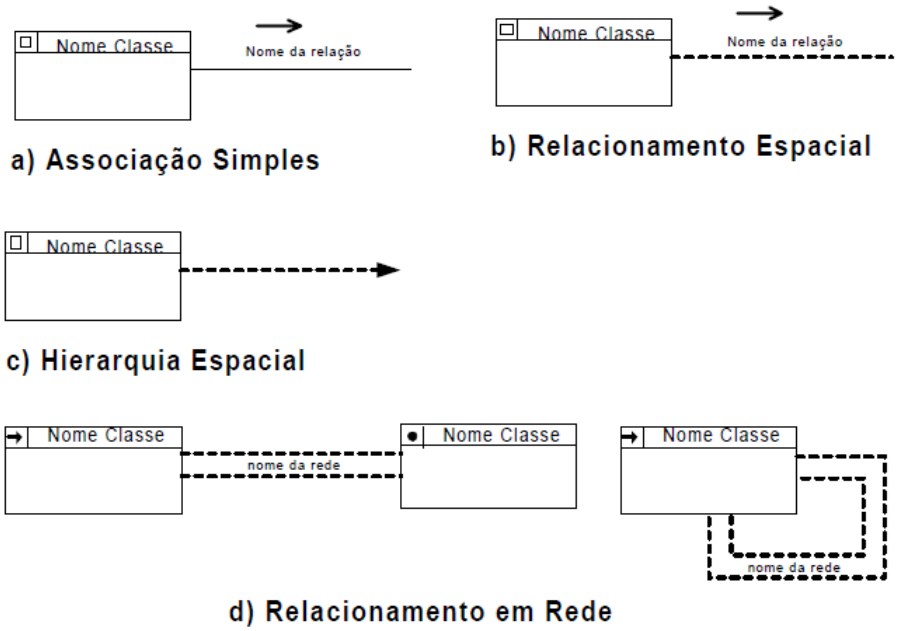


Figura B.5: Tipos de relacionamentos entre objetos do modelo GEO OMT. Fonte: Borges (1997a).

B.4 Considerações do Apêndice

O Modelo GEO OMT inspirou a proposição da *NET-UML* haja visto a contribuição deste modelo oferece para criação dos artefatos de análise dos sistemas de informação geográficas (SIG) a partir do uso representação simbólica relacionada com este domínio de aplicação, segmentação dos dados de acordo com os tipos e relacionamentos semânticos de conforme com os tipos de objetos.

Apêndice C

Leiautes de distribuição de grafos - Pseudo Códigos

Os leiautes de distribuição de grafo se constituem em importante ferramenta para visualização das informações contidas nas redes. A proposição do modelo *NET-UML* levou em consideração estudo dos aspectos conceituais e comportamentais envoltos neste algoritmos. A seguir apresentamos os pseudos códigos dos algoritmos de distribuição discutidos na presente pesquisa.

Após a discussão de cada um dos algoritmos serão apresentados leiautes de redes originadas pelos mesmos cuja a rede se refere a relação de coautoria em publicações em periódicos entre docentes, discentes e participantes externos de um programa de pós graduação da área interdisciplinar entre os anos de 2008 e 2014. A rede possui 204 vértices e 929 arestas.

C.1 Leiaute Circular Básico

O leiaute circular básico se caracteriza por organizar os vértices em um formato circular cuja o posicionamento de cada um dos vértices é dado pelo cálculo do deslocamento em radianos dos ângulos atribuídos a cada um dos vértices. Apresentamos no algoritmo 2 seu pseudo código (Csardi; Nepusz, 2006).

Algorithm 2 CIRCULAR LAYOUT

```
1: input graph;
2: og←graph.GetOrder();
3: pi←3.1415926;
4: for i ← 0 to og − 1 do
5:     av←(2*pi)/(og*i);
6:     vertices[i].posX←cosseno(av);
7:     vertices[i].posY←seno(av);
8:     i←i+1;
9: end for
```

O algoritmo têm como parâmetro de entrada um objeto grafo (linha 1) e possui com variáveis de inicialização *og*, representando a ordem do grafo, e *pi*, número trigonométrico que representa a divisão entre uma circunferência e o diâmetro correspondente, possuindo valor aproximado de 3.14 (respectivamente nas linhas 2 e 3).

Após inicializar as variáveis são realizadas iterações de acordo a ordem do grafo (linha 4). Em cada uma das iterações, é realizado cálculo para obtenção do ângulo do vértice em radiano de acordo a equação $av = (2 * pi)/(og * i)$ onde av é o ângulo do vértice a ser calculado, i representa o número identificador do vértice na iteração indo de 0 ao número total de vértices do grafo (linha 5). O posicionamento dos vértices no plano é dado pelo cosseno (eixo X) e seno (eixo Y) do ângulo atribuído ao vértice (respectivamente linhas 6 e 7).

Apresentamos na Figura C.1 distribuição da rede de coautoria gerada pelo leiaute circular. Observamos que o leiaute não se mostra adequado para este contexto uma vez que não se pode observar com clareza as relações de coautoria.

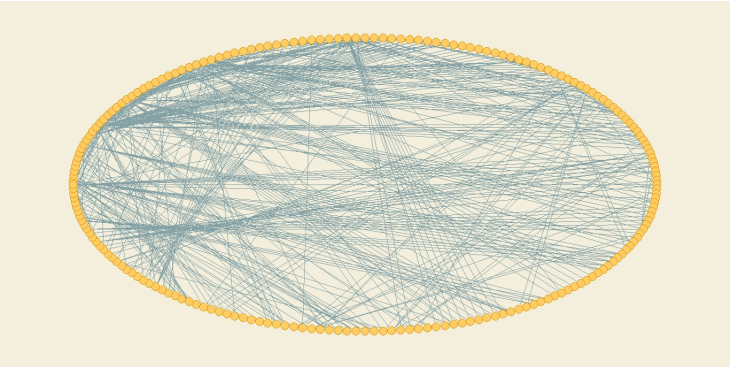


Figura C.1: Distribuição Circular para Rede de Coautoria, software Pajek. Fonte: Próprio autor

C.2 Leiaute Estrela

Neste leiaute os vértices são desenhados em formato de estrela inscrita em uma circunferência. O vértice como maior grau será posicionado ao centro da circunferência enquanto os demais serão desenhados em um formato circular.

Algorithm 3 STAR LAYOUT

```
1: input graph;
2: og←graph.GetOrder();
3: c←graph.GetMasterDegree();
4: pi←3.1415926;
5: for i ← 0 to og − 1 do
6:     av←(2*pi)/(og*i);
7:     if C == i then
8:         vertices[i].posX←0;
9:         vertices[i].posY←0;
10:    else
11:        vertices[i].posX←cosseno(av);
12:        vertices[i].posY←seno(av);
13:    end if
14:    i←i+1;
15: end for
```

Serão descritas apenas as características que diferenciam o leiaute estrela do circular. Foi necessário criarmos a variável c que recebe o identificador do vértice que possui o maior grau ou o maior número de conexões no grafo (linha 3). Durante a iteração que posiciona os vértices no grafo é realizado teste que verifica se o vértice a ser posicionado é o que possui maior grau (linha 7), sendo o vértice de maior grau, o mesmo será posicionado ao centro da circunferência (linhas 8 e 9), senão, o vértice será posicionado na borda da circunferência conforme discutido no leiaute circular (linhas 11 e 12).

Apresentamos na Figura C.2, distribuição da rede de coautoria gerado pelo leiaute estrela. Observamos que o leiaute não se mostra adequado para este contexto uma vez que somente podemos observar com clareza o autor que possui maior número conexões.

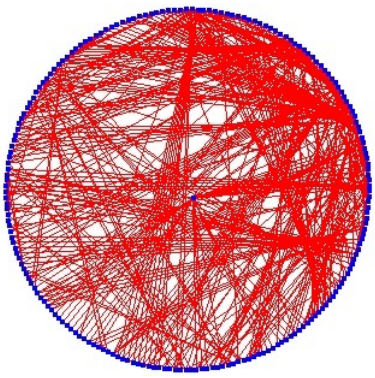


Figura C.2: Distribuição Estrela para Rede de Coautoria, software SC Net Draw. Fonte: Próprio autor

C.3 Leiaute K-Core

Conforme discutido no Capítulo 3, Seção 3.4.2, o algoritmo k-core organiza os vértices em camadas circunferenciais onde os que possuem maiores graus são posicionados nas camadas mais internas, próximas do núcleo central, enquanto os que possuem menor grau são posicionados nas camadas mais externas. Apresentamos no algoritmo 4 seu pseudo código (Vilas Bôas, 2016; Alvarez-hamelin et al., 2005).

Algorithm 4 KCore Layout

```
1: input graph;
2: radius←0;
3: og←graph.GetOrder();
4: listDegree←graph.GetListDegree();
5: fator←listDegree.Count()/100;
6: pi←3.1415926;
7: for index ← 0 to listDegree.Count() − 1 do
8:    vertices←graph.GetVerticeByDegree(listDegree[index]);
9:    for i ← 0 to vertices.Count() − 1 do
10:      av←(2*pi)/(og*i);
11:      vertices[i].posX←radius * cosseno(av);
12:      vertices[i].posY←radius * seno(av);
13:      i←i+1;
14:   end for
15:   radius←radius+fator;
16:   index←index+1;
17: end for
```

Discutiremos a seguir as características que diferenciam o leiaute k-core do circular básico. Foi necessário criarmos as variáveis de inicialização *radius*, *listDegree* e $fator$.

A variável *radius* (linha 2) será utilizada para a criação dos raios das circunferências onde os vértices serão posicionados, a variável *listDegree* (linha 4) receberá uma lista de graus dos vértices do grafo enquanto a variável $fator$ linha(5) será utilizada para ajuste do raio das circunferências a cada iteração.

Foram implementados dois laços aninhados sendo que o primeiro laço será iterado de acordo a lista de graus dos vértices (linha 7). A cada iteração é guardada na variável vértices (linha 8) uma lista com todos os vértices que possui o grau mantido na variável *listDegree*. O segundo laço (linha 9) é iterado de acordo com a quantidade de vértices mantido na variável *vertices* e a cada iteração e definido o posicionamento do vértice na circunferência, similar com o que já é feito no algoritmo de leiaute circular.

Apresentamos na Figura C.3 distribuição da rede de coautoria gerado pelo leiaute K-Core. Observamos que o leiaute não se adéqua totalmente para este contexto pois destaca apenas os vértices que possuem maior grau mas, o elevado número de cruzamento de arestas prejudica demais análises.

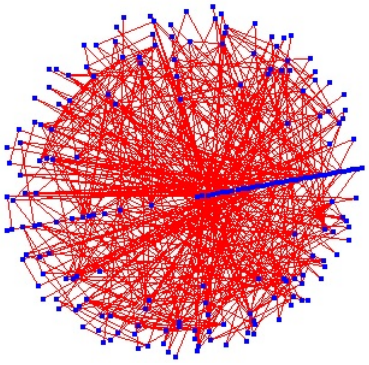


Figura C.3: Distribuição K-Core para Rede de Coautoria, software SC Net Draw. Fonte: Próprio autor

C.4 Leiaute Tree

Os leiautes de árvore se caracterizam pelo posicionamento do vértice raiz na parte superior da árvore e seus filhos são posicionados abaixo do pai como folhas, organizados de tal forma a evitar cruzamento de arestas (Wetherell; Shannon, 1979; Narkhede; Inamdar, 2014). A seguir é apresentado o pseudo código do algoritmo de leiaute de árvore proposto por Reingold e Tilford (1981) no artigo intitulado *Tidier Drawings of Trees*. O algoritmo de leiaute em árvore foi divido em quatro partes representadas respectivamente pelos algoritmos 5, 7, 7 e 8.

Algorithm 5 LAYOUT TREE - Parte 1

```
1: input←graph;
2: graph←CreateMinimalGeneratingTree(graph);
3: root←graph.GetRootVertex();
4: calculateCoordY(graph.vertices, graph.GetLevelsOfTree());
5: calculateOffset(graph.vertices, root.id);
6: calculateCoordX(graph.vertices, root.id, root.offset);
```

A primeira parte do algoritmo se encarrega de fazer a chamada ao algoritmos de definição da posição do vértice no eixo Y, cálculo do deslocamento dos vértices no eixo X e definição do posicionamento dos vértices no eixo X que representam respectivamente as partes 2, 3 e 4 do algoritmo.

Tem como parâmetro de entrada um grafo conectado dirigido (linha 1) que é transformado árvore a partir da função CreateMinimalGeneratingTree (linha 2), que implementa o algoritmo de Prim, para criação de uma árvore geradora minima (Ziviane, 2011). Foi necessário criamos a variavel de inicialização *root* que recebe o vértice raiz grafo (linha 3) e será utilizada a seguir como parâmetro nos algoritmos de deslocamento e posicionamento dos vértices no eixo X. Após inicialização das variáveis são chamados os algoritmos de cálculo do posicionamento dos vértices no eixo Y, deslocamento dos vértices no eixo X e posicionamento dos vértices no eixo X conforme descrito acima e destacado no algoritmo respectivamente nas linhas 4, 5 e 6. A seguir, é apresentado o algoritmo de posicionamento do vértice no eixo Y.

Algorithm 6 Y Coordinate Calculation Function - Parte 2

```
1: input←vertexList;
2: input←levetList;
3: posY←GetHeight()/2;
4: levelIncrement[]←double[levetList.Count];
5: yIncrment←(GetHeight() /levetList.Count) * 2;
6: for i ← 0 to levetList.Count() − 1 do
7:     posY ← posY- yIncrment;
8:     levelIncrement[i]←posY;
9:     i ← i+1;
10: end for
11: for i ← 0 to vertexList.Count() − 1 do
12:     vertexList[i].posY←levelIncrement[vertexList[i].level];
13:     i ← i+1;
14: end for
```

O algoritmo de cálculo das coordenadas Y têm como parâmetros de entrada uma lista dos vértices do grafo e uma lista de dos níveis dos vértices (linhas 1 e 2). Foi criada as variável posY (linha 3), que será utilizada para cálculo do posição dos vértices de acordo seu nível na árvore. A variável é inicializada com metade do valor definido para a altura da área disponível para a criação do leiaute, a variável *levelIncrement* (linha 4) recebe um vetor com a lista de níveis da árvore e variável *yIncrement* (linha 5) recebe o valor de deslocamento dos vértices no eixo Y dado pela divisão do tamanho do eixo pela quantidade de níveis da árvore.

O posicionamento dos vértices no eixo Y é definido pelo seu nível na hierarquia ou seja os filhos do nó raiz serão posicionados no segundo nível da hierarquia e os filhos dos filhos serão posicionados no terceiro nível da hierarquia e assim sucessivamente. O loop definido na linha 6 se encarrega de definir o posicionamento de todos os vértices do mesmo nível hierárquico enquanto o loop definido na linha 11 se encarrega de atribuir o posicionamento de cada vértice no eixo Y. A seguir apresentamos o algoritmo de cálculo do deslocamento

do vértice no eixo X.

Algorithm 7 Function of calculating the displacement of the nodes in the X axis - Parte 3

```
1: input←vertexList;
2: input←levetList;
3: vertexListBylevel←graph.GetVertexListByLevel();
4: for index ← 0 to vertexListBylevel.Count() − 1 do
5:    offset ← GetWidth()/graph.GetVertexByLevel(vertexListBylevel[index])+1;
6:    vertexListBylevel[index].offSet= offset;
7:    levelYncrement[i]←posY;
8:    index ← index+1;
9: end for
```

O cálculo do deslocamento dos vértices no eixo X é realizado por meio de função que têm como parâmetros de entrada uma lista de vértices e uma lista dos níveis dos vértices do grafo (linhas 1 e 2). Criamos a variável *vertexListByLevel* (linha 3) que guarda a lista de vértices por nível. O loop descrito na linha 4 percorre os vértices da rede agrupados por nível, e calcula o deslocamento dos vértices no eixo X a partir da divisão do comprimento do eixo X pela quantidade de vértices do mesmo nível hierárquico (linha 5), armazenando o valor na variável $offset$. Após o cálculo do deslocamento é atribuído para todos os vértices do mesmo nível hierárquico o mesmo valor de deslocamento (linha 7).

A seguir, apresentamos o algoritmo de cálculo do posicionamento dos vértices no eixo X.

Algorithm 8 X Coordinate Calculation Function - Parte 4

```
1: calculateCoordX;
2: input←vertices;
3: input←node;
4: input←xpos;
5: vertices[node].posX←xpos;
6: for index ← 0 to vertices.Count() -1 do
7:    if index = node then
8:       continue;
9:    end if
10:   if vertices[i].parent = node then
11:      calculateCoordX(vertices, i, xpos + vertices[i].offset);
12:   end if
13: end for
```

O posicionamento dos vértices no eixo X é dado por meio de função recursiva que possui

como parâmetros as variáveis *vertices*, *node* e *xpos* (linhas 2, 3 e 4) que são respectivamente uma lista de vértices, o índice do vértice e a posição do vértice em questão.

A posição do vértice no eixo X é definida pelo loop descrito na linha 6 onde de forma recursiva é definido o posicionamento dos vértices filhos a partir da soma do deslocamento do vértice filho com o deslocamento do vértice pai (linha 11) .

Apresentamos na Figura C.4 distribuição da rede de coautoria gerado pelo leiaute Tree, exibindo o relacionamento hierárquico entre os coautores. Observamos na inspeção visual que o leiautes hierárquicos se mostram inadequados para análise onde se objetiva verificar interação entre os atores, independente da existência de relações de dependência, direção do relacionamento e apresentação em níveis ou camadas.

Figura C.4: Distribuição Tree para Rede de Coautoria, software Tutorial Graph Sharp. Fonte: Próprio autor

C.5 Leiaute Sugiyama

Segundo, (Csardi; Nepusz, 2006), o algoritmo Sugiyama pode ser dividido em 4 etapas. Na etapa 1 são removidos os ciclos, e é criada uma estrutura hierárquica no grafo. Representamos esta etapa no algoritmo 9, com a execução função *CreateMinimalGeneratingTree*, que se ocupa de criar uma árvore geradora mínima que ao criar uma estrutura hierárquica elimina os ciclos existentes no grafo e atribui um valor no atributo *level* do objeto vértice, indicando o nível que o mesmo ocupa na hierarquia. Esta etapa do algoritmo se ocupa também de fazer chamada as etapas 2, 3 e 4 do SUGIYAMA, respectivamente algoritmos

10, 11 e 12.

Algorithm 9 LAYOUT SUGIYAMA - Parte 1

```
1: input←graph;
2: graph←CreateMinimalGeneratingTree(graph);
3: layerGraph←AssingLayer(graph);
4: layerGraph←ReductionOfCrossesBaricentricMethod(layerGraph);
5: graph←AssignPositions(layerGraph);
```

Na etapa 2, os dados são organizados em uma estrutura denominada de *layer* que funciona como uma coleção de subgrafos do grafo principal e possui como atributos: o índice da camada; uma lista de vértice e; uma lista de arestas. O algoritmo 10, cria uma lista de *layers*, levando em conta o nível de cada vértice na hierarquia conforme execução da função *CreateMinimalGeneratingTree*.

A etapa 3 se caracteriza pela execução do algoritmo heurístico 11, utilizado para diminuição do cruzamento de arestas (Sugiyama; Toda., 1980).

Na etapa 4 são definidas as posições horizontais dos vértices considerando a utilização de linhas retas e curtas para representar as arestas, conforme apresentamos no algoritmo 12.

Conforme apresentamos no algoritmo 10, a função *AssingLayer* se ocupa de criar estrutura de dados denominada *layer* com a adição dos vértices e arestas que compõem a camada, sugerindo a criação de subgrafos. A função têm como parâmetro um objeto grafo (linha 1) e variáveis de inicialização *levelList* que recebe uma lista de níveis de vértices de acordo com o atributo *level* do vértice (linha 2) e *layers* que receber os vértices e arestas das camadas (linha 3). Após a inicialização das variáveis é executado laço (linha 4) de acordo com o número de níveis e a cada repetição são recuperados os vértices e arestas que compõem a camada (linhas 5 e 6) e adicionados a variável *layers*.

Algorithm 10 ASSINGLAYER - Parte 2

```
1: input←graph;
2: levelList ← graph.GetLevelList();
3: layers ←null;
4: for index ← 0 to levelList.Count() -1 do
5:     layer.vertices←graph.GetVerticesByLevel(levelList[index]);
6:     layer.arestas←graph.GetArestasByLevel(levelList[index]);
7:     layers.add(layer);
8: end for
```

A função *ReductionOfCrossesBaricentricMethod*, que apresentamos no algoritmo 11 em

seu processamento busca a redução do numero de cruzamentos de arestas. Têm como parâmetro uma matriz de ordenação de vértices (linha 1) obtida a partir da estrutura de dados de camadas (estrutura de dados *layers*) e as variáveis de inicialização $numberOfInteractions$ que guarda o número de iterações a serem realizadas nas fases 1 e 2 do algoritmo e têm como valor o número de camadas da estrutura hierárquica do grafo (linha 2) e a variável Ms que representa a matriz de solução inicializada com o valor da matriz de realização parâmetro de inicialização do algoritmo.

Após inicialização das variáveis são executados dois laços representando as fases 1 e 2 do algoritmo. O laço que representa a fase 1 (linha 5), é verificado o número de cruzamento de arestas existente na matriz de realização (linha 6) em seguida a e realizado ordenamento baricêntrico na matriz de ordenação cuja o resultado é a criação de uma nova matriz de ordenação armazenada na variável $M1$ (linha 7), após a a criação da nova matriz é realizado nova verificação de cruzamento de arestas agora na matriz $M1$ cuja o valor é armazenado na variável K (linha 8), caso o número de cruzamentos da matriz de solução seja maior que o número de cruzamentos da nova matriz de ordenação (linha 9), a matriz de solução terá valor igual a nova matriz de ordenação baricêntrica $M1$. Após a criação da nova matriz de solução é criada nova matriz de ordenação a partir de $M1$ cuja o valor é armazenado em $M2$ (linha 13) sendo verificado novo número de cruzamento de arestas a partir de $M2$. Caso o número de cruzamento da matriz de solução seja maior que o número de cruzamentos de $M2$ então a matriz de solução Ms passará a ser igual a $M2$ (linha 16) e em seguida é feito nova verificação de cruzamentos de arestas (linha 17) o laço se repete enquanto $M2$ for diferente de Mr ou o número de iterações for alcançado.

O laço que representa a fase 2 do algoritmo se repete até que o número de iterações seja alcançado (linha 22), neste são realizada transformações baricêntricas onde são trocadas as ordens das linhas (linha 23), caso a ordenação baricêntrica ascendente não seja alcançada e realizada nova transformações baricêntricas onde são trocadas as ordens das colunas (linha 25), caso a ordenação baricêntrica ascendente seja alcançada o laço é encerrado caso contrario a fase 2 do algoritmo continua sua execução.

Algorithm 11 ReductionOfCrossesBaricentricMethod - Parte 3. Fonte Sugiyama e Toda. (1980)

```
1: input←Mr;
2: numberOfInteractions←Mr.Count();
3: Ms←Mr;
4: contador←0;
5: while (M2 ! = Mr) or (numberOfInteractions! =contador) do
6:    Ks←GetCrossing(Mr);
7:    M1←SetBaricentricOrder(Mr,r);
8:    K←GetCrossing(M1);
9:    if Ks > K then
10:      Ms←M1;
11:      Ks←K;
12:   end if
13:   M2←SetBaricentricOrder(M1,c);
14:   K←GetCrossing(M2);
15:   if Ks > K then
16:      Ms←M2;
17:      Ks←K;
18:   end if
19:   contador←contador+1;
20: end while
21: contador←0;
22: while (numberOfInteractions! =contador) do
23:   M3←GetBaricentricTransformer(M2,r);
24:   if GetBaricentricOrder(M3,asc)=true then
25:      M4←GetBaricentricTransformer(M3,r);
26:      if GetBaricentricOrder(M4,asc)=true then
27:         exit;
28:      else
29:         Ms←M4;
30:      end if
31:   else
32:      Ms←M3;
33:   end if
34:   contador←contador+1;
35: end while
```

A atribuição das posições dos vértices e escrita do leiaute e realizada com a execução do algoritmo 12, representando a parte 4 do *Sugiyama*. O algoritmo têm como parâmetro um objeto *layers* (linha 1) após o mesmo ter seus vértices com divisão em camadas e ordenação baricêntrica e variável de inicialização *verticalPos* utilizada para cálculo das

posições dos vértices (linha 2). São executados laços aninhados para definição das posições dos vértices, o posicionamento do vértice no eixo X leva em consideração o comprimento da camada e a distância horizontal (linha 13), sendo que a posição do vértice raiz em relação ao eixo X, colocado na camada 1, é no ponto 0. Já o posicionamento dos vértices em relação ao eixo Y leva em consideração a altura da camada e o deslocamento de valor 0.5, assumindo que com este valor de deslocamento serão obtidos leiautes mais equilibrados. Para o vértice raiz o posicionamento do mesmo em Y é dado pela diferença entre a altura total da camada pela altura do vértice em questão.

Algorithm 12 AssignPositions - Sugyama Parte 4

```
1: input←layers;
2: verticalPos←0;
3: for i ← 0 to layers.Count − 1 do
4:    pos←0;
5:    layerHeight←layers[i].Height;
6:    for x ← 0 to layers[i].Count − 1 do
7:       v.RealPosition.X ← pos;
8:       if X == 0 then
9:          v.RealPosition.Y ←layerHeight - layers[i][x].Size.Height;
10:      else
11:         v.RealPosition.Y ←verticalPos + layerHeight * 0.5);
12:      end if
13:      pos ← pos + layers[i][x].Size.Width + GetHorizontalgap();
14:   end for
15:   verticalPos += layerHeight + GetVerticalGap();
16: end for
```

Apresentamos na Figura C.5 distribuição da rede de coautoria gerado pelo leiaute Sugiyama. Leiaute inadequado para este tipo de análise por questões já discutidas na seção anterior e também devido a criação de leiaute com baixa visibilidade dado ao grande número de vértices e arestas.

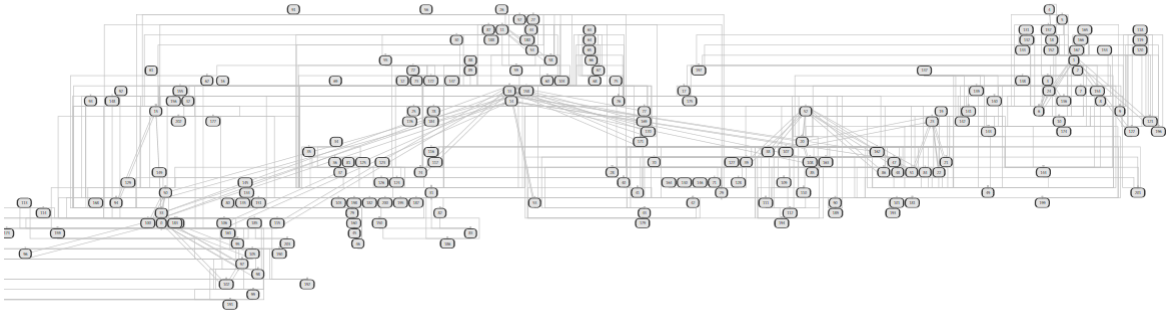


Figura C.5: [Distribuição Sugiyama para Rede de Coautoria, software Tutorial Graph Sharp. Fonte: Próprio autor

C.6 Leiaute Kamada Kawai

O algoritmo proposto por Kamada e Kawai (1989) utiliza os princípios da lei de *Hooke*, onde o posicionamento dos vértices é obtido por meio do cálculo da distância Euclidiana entre os vértices através da introdução de um sistema dinâmico virtual que define a posição final dos vértices quando a energia do sistema é considerada mínima. A seguir, é apresentado pseudo código do algoritmo proposto por Kamada e Kawai (1989).

Para melhor entendimento do algoritmo o mesmo foi dividido em três partes. A parte 1 do algoritmo é responsável por definir o posicionamento inicial dos vértices de forma aleatória (linha 1) e fazer a chamada dos algoritmos responsáveis pelo cálculo da distância entre os vértices, parte 2 (linha 4) e cálculo do cálculo do posicionamento dos vértices, parte 3 (linha 5).

Algorithm 13 KAMADA KAWAI LAYOUT - PARTE 1

```
1: input←graph;
2: graph←randomLayoutCreateLayout(graph);
3: CalculateVertexDistances(graph.GetDiameter(), graph.GetVertices());
4: CalculatePositionOfVertices(Graph.GetVertices());
```

A seguir, apresentamos o algoritmo 14 responsável pela execução do cálculo das distancias entre os vértices.

Algorithm 14 CALCULATE VERTEX DISTANCE - PARTE 2

```
1: input←diameter;
2: input←vertices;
3: height←GetHeight();
4: width←GetWidth();
5: n←vertices.Count;
6: dm←double[n, n];
7: LO←Min(height, width);
8: L←(LO/diameter) * lengthFactor;
9: for i ← 0 to vertices.Count() − 1 do
10:    for j ← 0 to vertices.Count() − 1 do
11:       dij←GetGeodeticDistance(vertices[i], vertices[j]);
12:       dji←GetGeodeticDistance(vertices[j], vertices[i]);
13:       dist←diameter * disconnectedMultiplier;
14:       if dij ! = 0 then
15:          dist←Min(dij, dist);
16:       end if
17:       if dji ! = 0 then
18:          dist←Min(dji, dist);
19:       end if
20:       dm[i, j]←dist;
21:       dm[j, i]←dist;
22:       j←j+1;
23:    end for
24:    i←i+1;
25: end for
```

O cálculo das distância entre os vértices é dado por meio do cálculo das distâncias geodésicas entre os vértices. O algoritmo têm como parâmetros o diâmetro da rede e uma lista de vértices (linhas 1 e 2). Têm como variáveis a altura e a largura da área disponível para o desenho do grafo, o número de vértices e um vetor bidimensional, utilizado para o cálculo das distâncias(da linha 3 a 6). A variável LO (linha 7) é utilizada para pegar o menor valor entre a altura e a largura, necessário para realizar o cálculo do valor da variável L (linha 8) que armazena o valor da distância tida com ideal para os vértices. Após a inicialização das variáveis são executados loops aninhados (linhas 9 e 10) para calcular e armazenar as distâncias geodésicas entre os vértices, para posterior uso na definição do posicionamento dos vértices (Kamada; Kawai, 1989; Csardi; Nepusz, 2006).

A seguir, apresentamos o algoritmo 15, responsável pela atribuição das novas posições dos vértices, parte 3 do algoritmo *Kamada Kawai*.

Algorithm 15 Calculate Position Of Vertices - Parte 3

```
1: input←vertices;
2: energy←calcEnergy();
3: for i ← 0 to vertices.Count() − 1 do
4:    dxy[]←calcDeltaXY(i);
5:    vertices[i].posX←vertices[i].posX + dxy[0];
6:    vertices[i].posY←vertices[i].posY + dxy[0];
7:    deltam←calcDeltaM(i);
8:    i←i+1;
9: end for
10: AdjustForGravity();
11: energy←calcEnergy();
12: for i ← 0 to vertices.Count() − 1 do
13:    for j ← 0 to vertices.Count() − 1 do
14:       xenergy←calcEnergyIfExchanged(i, j);
15:       if energy < xenergy then
16:          sx←vertices[i].posX;
17:          sy←vertices[i].posY;
18:          vertices[i].posX←vertices[j].posX;
19:          vertices[i].posY←vertices[j].posY;
20:          vertices[j].posX←sx;
21:          vertices[j].posY←sy;
22:          pm←i;
23:       end if
24:       j←j+1;
25:    end for
26:    i←i+1;
27: end for
```

O posicionamento dos vértices e dado por meio de minimização da energia existente entre os vértices, atribuída pelo sistema dinâmico virtual que coloca os vértices em posições estáveis, com minimização de energia. Têm como parâmetro de entrada uma lista de vértices cuja posições iniciais forma definidas de forma aleatória na parte 1 do algoritmo. A variável *energy* (linha 2) recebe a energia inicial do sistema virtual calculada pelas posições dos vértices com base no método Newton-Raphson, através da função calcEnergy(). Após o cálculo da energia inicial do sistema, é realizado o cálculo de ajuste da gravidade das partículas do sistema, neste caso os vértices, como forma de iniciar a minimização da energia e posicionar os vértices em posições mais estáveis (linha 10). Em seguida, na linha 11, é feito novo cálculo para atualização da energia no sistema após cálculo da gravidade. Os laços aninhados das linhas 12 e 13 são utilizados para realização do cálculo da troca

de energia entre as partículas com isso se obtêm a minimização da energia do sistema e as posições finais dos partículas (vértices) (Kamada; Kawai, 1989; Csardi; Nepusz, 2006).

Apresentamos na Figura C.6 distribuição da rede de coautoria gerado pelo leiaute Kamada Kawai. Leiaute se mostra adequado para este tipo de análise, podemos observar distribuição agradável, vértices que se destacam pelas propriedades de centralidade de graus, proximidade e intermediação bem como indícios da existência de comunidade.

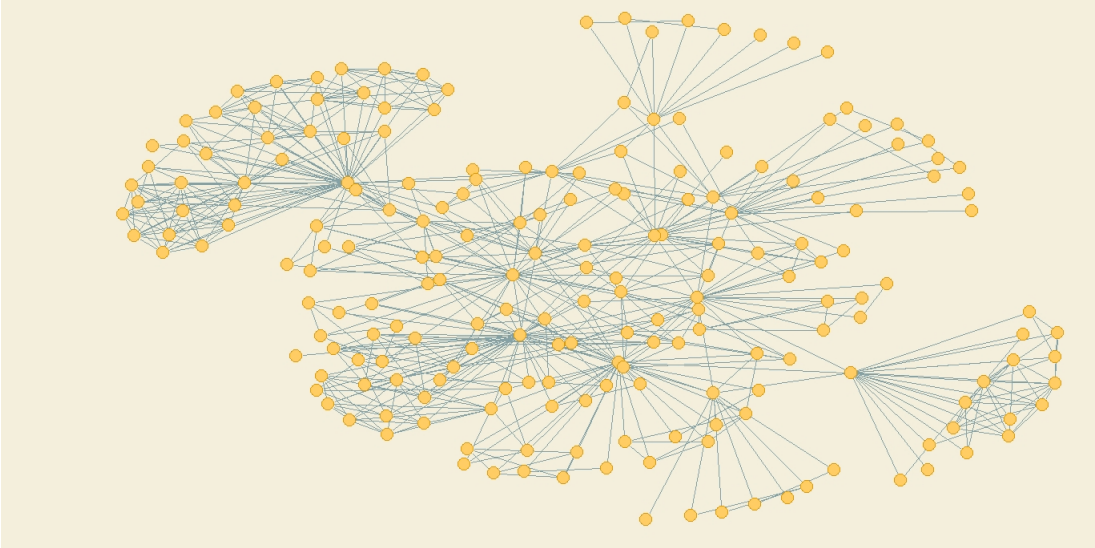


Figura C.6: [Distribuição Kamada Kawai para Rede de Coautoria, software Pajek. Fonte: Próprio autor

C.7 Leiaute Fruchterman Reingold

Conforme discutido no Capítulo 3, Seção 3.4.2.3, Fruchterman e Reingold (1991) propuseram algoritmo para grafos não dirigidos onde os vértices conectados devem ser posicionados próximos uns dos outros (e.g. força de atração) enquanto os vértices não vizinhos devem ser posicionados distantes daqueles que não possuem conexão (e.g. força de repulsão).

A seguir apresentamos o pseudo código do algoritmo *Fruchterman e Reingold* a partir da implementação realizada por Vilas Bôas (2016) no trabalho Visualização de Informação em Redes Sociais e Complexas Utilizando GuaráScript, cuja o código, foi modificado para uso em linguagens de desenvolvimento OO.

Conforme proposição de Vilas Bôas (2016) o algoritmo foi dividido em três partes. Respectivamente cálculos da força de repulsão, atração e atualização das posições dos vértices.

A posição inicial dos vértices e dada de forma aleatória, similar ao leiaute Kamada Kawai e não interfere nos cálculos de posicionamento final dos vértices (Vilas Bôas, 2016; Fruchterman; Reingold, 1991; Csardi; Nepusz, 2006).

A seguir, é apresentado o algoritmo de cálculo da força de repulsão.

Algorithm 16 CALCULATE REPULSION FORCE - FRUCHTERMAN E REINGOLD PARTE 1

```
1: input←graph;
2: for i ← 0 to graph.vertices.Count() − 1 do
3:    dxCurrentVertex←graph.vertices[i].posX;
4:    dyCurrentVertex←graph.vertices[i].posY;
5:    for j ← 0 to graph.vertices.Count() − 1 do
6:       if i ! = j then
7:          displacementDifferenceX←dxCurrentVertex - graph.vertices[j].posX;
8:          displacementDifferenceY←dyCurrentVertex - graph.vertices[j].posY;
9:          value←displacementDifferenceX * displacementDifferenceX + displacementDifferen-
            ceY * displacementDifferenceY;
10:         if value ! = 0 then
11:            d←Sqrt(value);
12:         else
13:            d←0.001;
14:         end if
15:         if d > 0 then
16:            repulseF←GetRepulsionForce(d);
17:            graph.vertices[i].forceDirectionX← graph.vertices[i].forceDirectionX + (displace-
               mentDifferenceX / d) * repulseF;
18:            graph.vertices[i].forceDirectionY← graph.vertices[i].forceDirectionY + (displace-
               mentDifferenceY / d) * repulseF;
19:         end if
20:         pm←i;
21:      end if
22:      j←j+1;
23:   end for
24:   i←i+1;
25: end for
```

A implementação do cálculo de repulsão entre os vértices não vizinhos é realizada por meio de dois laços aninhados onde o primeiro laço se encarrega de pegar a posição atual do vértice por meio das coordenadas X e Y (linhas 3 e 4) enquanto que no segundo laço é realizado o cálculo das distâncias entre os vértices. Somente após o cálculo das distâncias é realizado o cálculo da força de repulsão que é dado pela fórmula ($F_r = k/d$) e encapsulado na função GetRepulsionForce() e armazenado na variável repulseF (linha

16). Após obtermos valor da força de repulsão, é efetuado cálculo da força de direção de cada um dos vértices respectivamente, linhas 17 e 18 (Vilas Bôas, 2016; Fruchterman; Reingold, 1991).

A seguir, apresentamos o algoritmo de cálculo da força de atração.

Algorithm 17 Calculate Attraction Force - Fruchterman e Reingold Parte 2

```
1: input←graph;
2: for i ← 0 to graph.edges.Count() − 1 do
3:    displacementDifferenceX←graph.edges[i].source.posX - graph.edges[i].target.posX;
4:    displacementDifferenceY←graph.edges[i].source.posY - graph.edges[i].target.posY;
5:    value←displacementDifferenceX * displacementDifferenceX + displacementDifferenceY *
      displacementDifferenceY;
6:    if value ! = 0 then
7:       d←Sqrt(value);
8:    else
9:       d←0.001;
10:   end if
11:   attractiveF←GetAttractionForce(d);
12:   if d > 0 then
13:      graph.vertices[i].forceDirectionX←graph.vertices[i].forceDirectionX - (displacementDif-
         ferenceX / d) * attractiveF;
14:      graph.vertices[i].forceDirectionY← graph.vertices[i].forceDirectionY - (displacementDif-
         ferenceY / d) * attractiveF;
15:      graph.vertices[i + 1].forceDirectionX← graph.vertices[i + 1].forceDirectionX + (displa-
         cementDifferenceX / d) * attractiveF;
16:      graph.vertices[i + 1].forceDirectionY← graph.vertices[i + 1].forceDirectionY + (displa-
         cementDifferenceY / d) * attractiveF;
17:   end if
18:   i←i+1;
19: end for
```

A implementação do cálculo de atracão dos vértices vizinhos é realizado por meio de um laço (da linha 3 a 19) que percorre as arestas do grafo e em cada uma das iterações são realizados os cálculos das distâncias entre os vértices vizinhos, a partir do cálculo das distâncias dos vértices nos eixo X (linha 3) e Y (linha 4) e em seguida o cálculo do distanciamento ao quadrado (linhas 5 e 7), de posse do valor do distanciamento ao quadrado realizamos o cálculo para obtenção da força de atração obtida por meio da formula ($F_a = -k * d^2$) encapsulada fa função GetAttractionForce() e armazenado na variável attractiveF (Vilas Bôas, 2016; Fruchterman; Reingold, 1991). Apos obter o valor da força de atração é efetuamos o cálculo da força de direção dos vértices e seus vizinhos (linhas 13,14,15 e 16) (Vilas Bôas, 2016; Fruchterman; Reingold, 1991).

A seguir, apresentamos o algoritmo de atualização das posições dos vértices após a realização dos cálculos das forças de repulsão e atração.

Algorithm 18 Update Position - Fruchterman e Reingold Parte 3

```
1: input←graph;
2: for i ← 0 to graph.vertices.Count() − 1 do
3:     d←GetDisplacement(graph.vertices[i].forceDirectionX, graph.vertices[i].forceDirectionY);

4:     limitedDist←maxDisplace * (this.speed / this.speedDivisor);
5:     graph.vertices[i].posX ← graph.vertices[i].posX + displacementX / d * limitedDist;
6:     graph.vertices[i].posY ← graph.vertices[i].posY + displacementY / d * limitedDist;
7:     i←i+1;
8: end for
```

O novo posicionamento dos vértices é obtido a partir do cálculo das máximas distâncias entre os vértices obtida a partir da aplicação das forças de repulsão e atração. A implementação do algoritmo é dado por meio de um laço que percorre os vértices do grafo e em cada iteração é feito o cálculo do distanciamento dos vértices a partir das forças de direção do vértice nos eixos X e Y (linha 3). Após a obtenção do distanciamento dos vértices realizamos cálculo do limite da distância entre os vértices (linha x) e realizamos a atualização das posições dos vértices no leiaute (Vilas Bôas, 2016; Fruchterman; Reingold, 1991).

Apresentamos na Figura C.7 distribuição da rede de coautoria gerado pelo leiaute Fruchterman Reingold. Leiaute se mostra parcialmente adequado para este tipo de análise, podemos observar distribuição razoavelmente agradável porém não conseguimos identificar com a mesma rapidez as propriedades observadas através do leiaute Kamada Kawai, Figura C.6.

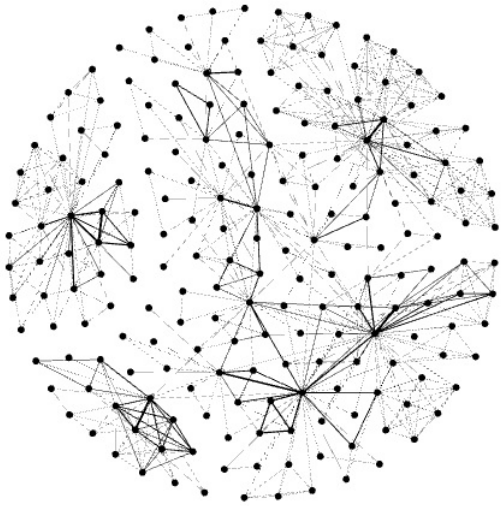


Figura C.7: Distribuição Fruchterman Reingold para Rede de Coautoria, software Gephi. Fonte: Próprio autor

C.8 Leiaute Force Atlas 2

Conforme discutido no Capítulo 3, Seção 3.4.2.3, Jacomy et al. (2014) propuseram algoritmo dirigido por força que simula sistema físico para calcular a distribuição dos vértices de um grafo. Neste algoritmo os vértices se repelem como partículas carregadas enquanto as arestas se ocupam de atrair os vértices simulando um sistema de molas.

Para melhor entendimento o algoritmo foi divido em 4 partes: Parte 1, responsável por inicializar as parâmetros do Force Atlas 2 e fazer chamada para algoritmos de repulsão, atração e atualização das posições do vértices; Parte 2, responsável por realizar os cálculos de forças de repulsão dos vértices; Parte 3, responsável pelos cálculos de força de atração dos vértices e; Parte 3, atualização das posições dos vértices a partir da aplicação das forças, respectivamente algoritmos 19, 20, 21 e 22. Criamos também algoritmos para cálculo de distância entre vértices e atribuição de novas distancias a partir da aplicação das formas de atração e repulsão, respectivamente algoritmos 23 e 24.

Conforme discutimos na seção anterior, o algoritmo 19, se encarrega de inicializar os parâmetros do Force Atlas 2 e fazer chamada as demais parte do mesmo. Os parâmetros do algoritmo são o objeto grafo (linha 1), valores numéricos de coeficiente de escala, gravidade, influencia de peso de aresta e valor de *Theta* para simulação de *Barnes Hut* (linha 2) e variáveis booleanas para definição do uso da otimização de *Barnes Hut*, aplicação de maior força de gravidade, ajuste de tamanho como forma de prevenir sobreposição de vértices e distribuição de atração resultando na colocação de *hubs* nas proximidades do grafo, (linha 2).

O cálculo da força repulsão entre os vértices é pode ser realizado de duas maneiras, levando ou não em consideração a colisão (sobreposição) entre os vértices. Caso o valor do parâmetro AdjustBySize seja verdadeiro o cálculo da força de repulsão será realizada de forma linear evitando a colisão entre vértices, caso contrário o calculo continuará sendo feito dor forma linear porém as colisões serão desprezadas conforme descrito entre as linhas 4 a 8. Após definição do tipo de cálculo da força de repulsão e realizado laço para aplicação da tipo da força nos vértices do grafo (linha 9).

A força de atração entre os vértices também leva em consideração a verificação da colisão entre os vértices, como no cálculo da força de repulsão porém esta pode ser realizada a partir do cálculo de distâncias lineares e logarítmicas simples ou de baseada em graus de distribuição, conforme estrutura de decisão compreendida entre as linhas 12 e 40. Após definição do tipo de cálculo da força de atração e realizado laço para aplicação da tipo da força nos vértices do grafo (linha 41).

Algorithm 19 ForceAtlas2 - Parte 1

```
1: input←Graph;
2: input←Scaling,Gravity,EdgeWeightInfluence,BarnesHutTheta;
3: input←BarnesHutOptimize,StrongGravity,AdjustBySize,AttractionDistribution;
4: if adjustBySize=true then
5:    typeForce ← LinRepulsionAntiCollision;
6: else
7:    typeForce ← LinRepulsion;
8: end if
9: for i ← 0 to Graph.edges.Count() − 1 do
10:   CalculateRepulsionForce (typeForce,Scaling,Graph.edges[i].source,Graph.edges[i].target);
11: end for
12: if adjustBySize=true then
13:    if logAttraction=true then
14:       if distributedAttraction=true then
15:          typeForce ← LogAttractionDegreeDistributedAntiCollision;
16:       else
17:          typeForce ← LogAttractionAntiCollision;
18:       end if
19:    else
20:       if distributedAttraction=true then
21:          typeForce ← LinAttractionDegreeDistributedAntiCollision;
22:       else
23:          typeForce ← LinAttractionAntiCollision;
24:       end if
25:    end if
26: else
27:    if logAttraction=true then
28:       if distributedAttraction=true then
29:          typeForce ← LogAttractionDegreeDistributed;
30:       else
31:          typeForce ← LogAttraction;
32:       end if
33:    else
34:       if distributedAttraction=true then
35:          typeForce ← LinAttractionMassDistributed;
36:       else
37:          typeForce ← LinAttraction;
38:       end if
39:    end if
40: end if
41: for i ← 0 to Graph.edges.Count() − 1 do
42:    CalculateAttractionForce (typeForce,Scaling,Graph.edges[i].source,Graph.edges[i].target);
43: end for
```

O algoritmo 19, se encarrega de realizar o calculo das forças de repulsão entre os vértices, o algoritmo têm como parâmetros, o tipo de calculo da força de repulsão, os coeficiente de escala e os vértices onde serão aplicadas as forças. Após a inicialização das variáveis (linhas 1 a 4), e realizado calculo da distância entre os vértices por meio da função CalculateDistancesOfTwoVertices (linha 5). Os tipos de cálculo de força de repulsão são discutidos a seguir:

- *Lin Repulsion AntiCollision A* – Cálculo linear da força de repulsão quando a distância entre os vértices é maior que zero. Onde C é a constante de escala, $MV1$ é a massa do vértice 1, $MV2$ é a massa do vértice 2 e D é a distância entre os vértices, conforme apresentamos na Equação C.1 (Jacomy et al., 2014).

$$F_R = \frac{(C * MV1 * MV2)/D}{D} \tag{C.1}$$

- *Lin Repulsion AntiCollision B* – Cálculo linear da força de repulsão quando a distância entre os vértices é menor que zero. Onde C é a constante de escala, $MV1$ é a massa do vértice 1, $MV2$ é a massa do vértice 2, conforme apresentamos na Equação C.2 (Jacomy et al., 2014).

$$F_R = 100 * C * MV1 * MV2 \tag{C.2}$$

Após a realização dos cálculos são atribuídos aos vértices as novas distâncias por meio da função SetDistancesOfTwoVertices (linhas 11).

Algorithm 20 CalculateRepulsionForce-Parte 2

```
1: input←TypeForce;
2: input←Coefficient;
3: input←V1;
4: input←V2;
5: distance ← CalculateDistancesOfTwoVertices(V1,V2);
6: if distance > 0 then
7:     factor ← (Coefficient * V1.GetMass() * V2.GetMass() / distance)/distance;
8: else
9:     factor ← 100 * coefficient * V1.GetMass() * V2.GetMass();
10: end if
11: SetDistancesOfTwoVertices(v1,v2,factor);
```

O cálculo de atração entre os vértices no Force Atlas 2, como já discutido na apresentação do calculo de repulsão, pode ser realizado também de forma linear ou logaritmântica, levando em conta a distribuição em graus e colisão entre os vértices, conforme Apresentamos no algoritmo 21. O algoritmo têm como parâmetros, o tipo de calculo da força de

repulsão ($TypeForce$), o coeficiente de escala ($Coefficient$) e os vértices ($V1$ e $V2$) onde serão aplicadas as forças. Após a inicialização das variáveis (linhas 1 a 4), e realizado calculo da distância entre os vértices por meio da função CalculateDistancesOfTwoVertices (linha 5), após a obtenção da distância entre os vértices é realizado o cálculo de atração entre os mesmo de acordo o tipo de calculo selecionado pelo usuário por meio da escolha dos parâmetros iniciais do Force Atlas 2. Os tipos de cálculo de atração são discutidos a seguir:

- *LogAttractionAntiCollision* – O cálculo logarítmico da atração entre vértices anticolisão ou sobreposição é dado pelo produto do coeficiente de escala (C), peso da aresta (E) e Log da distância entre os vértices (D) entre os vértices + 1 dividido pela distância entre os vértices, conforme apresentamos na Equação C.3 (Jacomy et al., 2014).

$$F_A = \frac{-C * E * Log(1 + D)}{D} \tag{C.3}$$

- *LogAttractionDegreeDistributionAntiCollision* – O cálculo logarítmico da atração entre vértices anticolisão levando em consideração os graus de distribuição é dado pelo produto do coeficiente de escala (C), peso da aresta (E) e Log da distância entre os vértices (D) entre os vértices + 1 dividido pela massa do vértices 1, conforme apresentamos na Equação C.4 (Jacomy et al., 2014).

$$F_A = \frac{-C * E * Log(1 + D)/D}{V1.Mass} \tag{C.4}$$

- *LinAttractionDegreeDistributionAntiCollision* – A Atração linear entre os vértices anticolisão, levando em consideração os graus de distribuição é dada pelo produto do coeficiente de escala (C) pelo peso da aresta (E) dividido pela massa do vértices 1, conforme apresentamos na Equação C.5 (Jacomy et al., 2014).

$$F_A = \frac{-C * E}{V1.Mass} \tag{C.5}$$

- *LinAttractionAntiCollision* – A Atração linear entre os vértices anticolisãoé dada pelo produto do coeficiente de escala (C) pelo peso da aresta (E) conforme apresentamos na Equação C.6 (Jacomy et al., 2014).

$$F_A = -C * E \tag{C.6}$$

- *LogAttraction* – A Atração logarítmica entre os vértices é dada pelo produto do coeficiente de escala (C), peso da aresta(E), log da distância entre os vértices + 1 ($Log(1 + D)$), dividido pela distância entre os vértices (D), conforme apresentamos

na Equação C.7 (Jacomy et al., 2014).

$$F_A = \frac{-C * E * Log(1 + D)}{D} \tag{C.7}$$

- *LogAttractionDegreeDistribution* – O cálculo da atração logarítmica entre os vértices, levando em consideração os graus de distribuição, é dada pelo produto do coeficiente de escala (C), peso da aresta (E), log da distância entre os vértices + 1 ($Log(1+D)$), dividido pela massa do vértice 1, conforme apresentamos na Equação C.8 (Jacomy et al., 2014).

$$F_A = \frac{(-C * E * Log(1 + D))/D}{V1.Mass} \tag{C.8}$$

- *LinAttractionMassDistributed* – O cálculo da atração linear entre os vértices, levando em consideração a distribuição de massa dos vértices, é dada pelo produto do coeficiente de escala (C), peso da aresta (E), dividido pela massa do vértice 1, conforme apresentamos na Equação C.9 (Jacomy et al., 2014).

$$F_A = \frac{-C * E}{V1.Mass} \tag{C.9}$$

- *LinAttraction* – O cálculo da atração linear entre os vértices é dado pelo produto do coeficiente de escala (C) pelo peso da aresta (E), conforme apresentamos na Equação C.10 (Jacomy et al., 2014).

$$F_A = -C * E \tag{C.10}$$

Após a realização dos cálculos, são atribuídos aos vértices as novas distâncias por meio da função SetDistancesOfTwoVertices (linhas 23).

Algorithm 21 CalculateAttractionForce-Parte 3

```
1: input←TypeForce;
2: input←Coefficient;
3: input←V1;
4: input←V2;
5: distance ← CalculateDistancesOfTwoVertices(V1,V2);
6: if TypeForce=LogAttractionAntiCollision then
7:    factor ← (-coefficient * e * Math.Log(1 + distance)) / distance;
8: else if TypeForce=LogAttractionDegreeDistributedAntiCollision then
9:    factor ← (-coefficient * e * Math.Log(1 + distance)) / (distance / n1.GetMass());
10: else if TypeForce=LinAttractionDegreeDistributedAntiCollision then
11:    factor ← (-coefficient * EdgeWeight) / v1.GetMass();
12: else if TypeForce=LinAttractionAntiCollision then
13:    factor ← (-coefficient * EdgeWeight);
14: else if TypeForce=LogAttraction then
15:    factor ← -coefficient * EdgeWeight * Math.Log(1 + distance) / distance;
16: else if TypeForce=LogAttractionDegreeDistributed then
17:    factor ← -coefficient * e * Math.Log(1 + distance) / distance / n1.GetMass();
18: else if typeForce=LinAttractionMassDistributed then
19:    factor ← -coefficient * e / n1.GetMass();
20: else if TypeForce=LinAttraction then
21:    factor ← -coefficient * EdgeWeight;
22: end if
23: SetDistancesOfTwoVertices(v1,v2,factor);
```

Uma vez calculadas as distâncias entre os vértices, após cálculo de forças de repulsão e atração, o próximo passo é definir as novas posições dos vértices conforme apresentamos no algoritmo 22, que representa a Parte 4 do Force Atlas 2. A função de atualização das posições dos vértices têm como parâmetro um objeto grafo e a atribuição das novas posições é obtida através da execução de laço que percorre todos os vértices do grafo (linha 2), realização de cálculo da oscilação do posicionamento do vértice e fator de ajuste (linhas 3 e 4). Caso seja selecionado o parâmetro de sobreposição, conforme apresentamos na parte 1 do Force Atlas, é realizado cálculo de ajuste do fator de posicionamento (linhas 6 e 7). Após a obtenção do fator de ajuste é realizado o cálculo de posicionamento do vértice obtido pela soma das novas distâncias com a posição atual do vértice e fator de ajuste (linhas 9 e 10).

Algorithm 22 ForceAtlas2-UpdatePosition - Parte 4

```
1: input←graph;
2: for i ← 0 to graph.vertices.Count() − 1 do
3:    swinging←graph.vertices[i].GetMass()*Math.Sqrt((graph.vertices[i].GetOldDx()-
      graph.vertices[i].GetDx()*(graph.vertices[i].GetOldDx()-graph.vertices[i].GetDx())+(graph.vertices[i].
      graph.vertices[i].GetDy())*(graph.vertices[i].GetOldDy()-graph.vertices[i].GetDy()));
4:    factor ←0.1*GetSpeed()/(1+Math.Sqrt(GetSpeed()*swinging));
5:    if isAdjustSizes() then
6:       df ←Math.Sqrt(Math.Pow(graph.vertices[i].GetDx(), 2)+Math.Pow(graph.vertices[i].GetDy(),
         2));
7:       factor ←Math.Min(factor*d,10.0)/df;
8:    end if
9:    x ←graph.vertices[i].GetPosX()+graph.vertices[i].GetDx()*factor;
10:   y ← graph.vertices[i].GetPosY()+graph.vertices[i].GetDy()*factor;
11:   graph.vertices[i].SetPosX(x);
12:   graph.vertices[i].SetPosY(y);
13: end for
```

Apresentamos a seguir os algoritmos de cálculo das distâncias entre vértices e atribuição das novas distâncias respectivamente algoritmos 23 e 24.

A função de cálculo da distância entre dois vértices, algoritmo 23, têm como parâmetros os vértices que se deseja calcular a distância (linhas 1 e 2), sendo que está é obtida através da verificação da distância dos vértices em relação ao eixo X (linha 3) e em relação ao eixo Y (linha 4), após a obtenção das distância é realizado o cálculo da distância entre os vértices através do cálculo da raiz da soma do quadrado das distâncias entre os eixos X e Y (linha 5).

Algorithm 23 CalculateDistancesOfTwoVertices

```
1: input←V1;
2: input←V2;
3: xDist ← V1.GetPosX() - V2.GetPosX();
4: yDist ← V1.GetPosX() - V2.GetPosY();
5: distance ← Math.Sqrt(xDist * xDist + yDist * yDist);
6: ouput←distance;
```

A função de atribuição das novas distâncias entre os vértices, algoritmo 24, têm como parâmetros os vértices que se deseja calcular a distância (linhas 1 e 2) e a força de repulsão e atração previamente calculada (linha 3). As novas distâncias dos vértices nos em relação aos eixos X e Y são conhecidas a partir do cálculo das distancia entre as posições atuais dos vértices (linha 4 e 5), multiplicação destas distâncias pelo força

informada como parâmetro da função somada a distância atual do vértice nos eixos X e Y (linhas 6 a 9).

Algorithm 24 SetDistancesOfTwoVertices

```
1: input←v1;
2: input←v2;
3: input←factor;
4: xDist←v1.GetPosX() - v2.GetPosX();
5: yDist←v1.GetPosY() - v2.GetPosY();
6: v1.SetDx(n1.GetDx() + (xDist * factor));
7: v1.SetDy(n1.GetDy() + (yDist * factor));
8: v2.SetDx(n2.GetDx() - (xDist * factor));
9: v2.SetDy(n2.GetDy() - (yDist * factor));
```

Apresentamos na Figura C.8 distribuição da rede de coautoria gerado pelo leiaute Force Atlas 2. O leiautes se mostra adequado para este tipo de análise uma vez que leiautes desta categoria apresenta na inspeção visual estrutura de comunidades provocando assim investigação de medidas como modularidade.

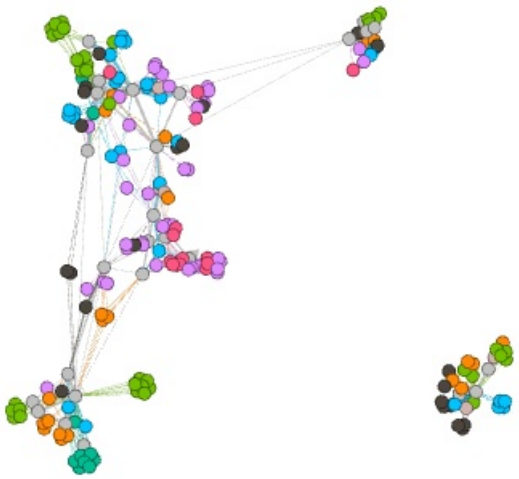


Figura C.8: Distribuição Force Atlas 2 para Rede de Coautoria, software Gephi. Fonte: Próprio autor

C.9 Leiaute Clockwise Rotate e Center Clockwise Rotate

Os algoritmos Clockwise Rotate e Center Clockwise Rotate aplicam transformação geométrica no posicionamento dos vértices a partir de um ângulo informado realizando respectivamente, operação de rotação no sentido horário e sentido anti-horário, conforme apresen-

tamos no Capítulo 3, Seção 3.4.2.4,

Apresentamos a seguir o algoritmo Rotation Layout que realiza as operações de rotação nos elementos do grafo.

Algorithm 25 ROTATION LAYOUT

```
1: input graph;
2: input angle;
3: pi←3.1415926;
4: sin←seno((-angle * pi)/180);
5: cos←cosseno((-angle * pi)/180);
6: for index ← 0 to graph.GetVertices().Count() − 1 do
7:     dx←graph.vertices[index].posX;
8:     dy←graph.vertices[index].posY;
9:     vertices[index].posX←(dx * cos -dy * sin);
10:    vertices[index].posY←(dx * cos -dy * sin);
11:    index←index+1;
12: end for
```

O algoritmo têm como parâmetros um objeto grafo, cujo posicionamento dos vértices sofre modificação, e o ângulo de rotação (linhas 1 e 2). O sentido de rotação dos elementos do grafo se dá pelo valor do parâmetro ângulo, caso o valor informado seja positivo a rotação será em sentido horário, caso o valor informado seja negativo a rotação se dará em sentido anti-horário (Khokhar, 2015). Esta tipo de algoritmo possui complexidade computacional $\theta(n)$ sendo n o número de vértices.

Apresentamos na Figura C.9 transformação geométrica de rotação no leiaute do grafo, após a aplicação da distribuição Force Atlas 2. Observamos que uso deste leiaute é útil quando se deseja observar o grafo com maiores detalhes e perspectivas diferentes

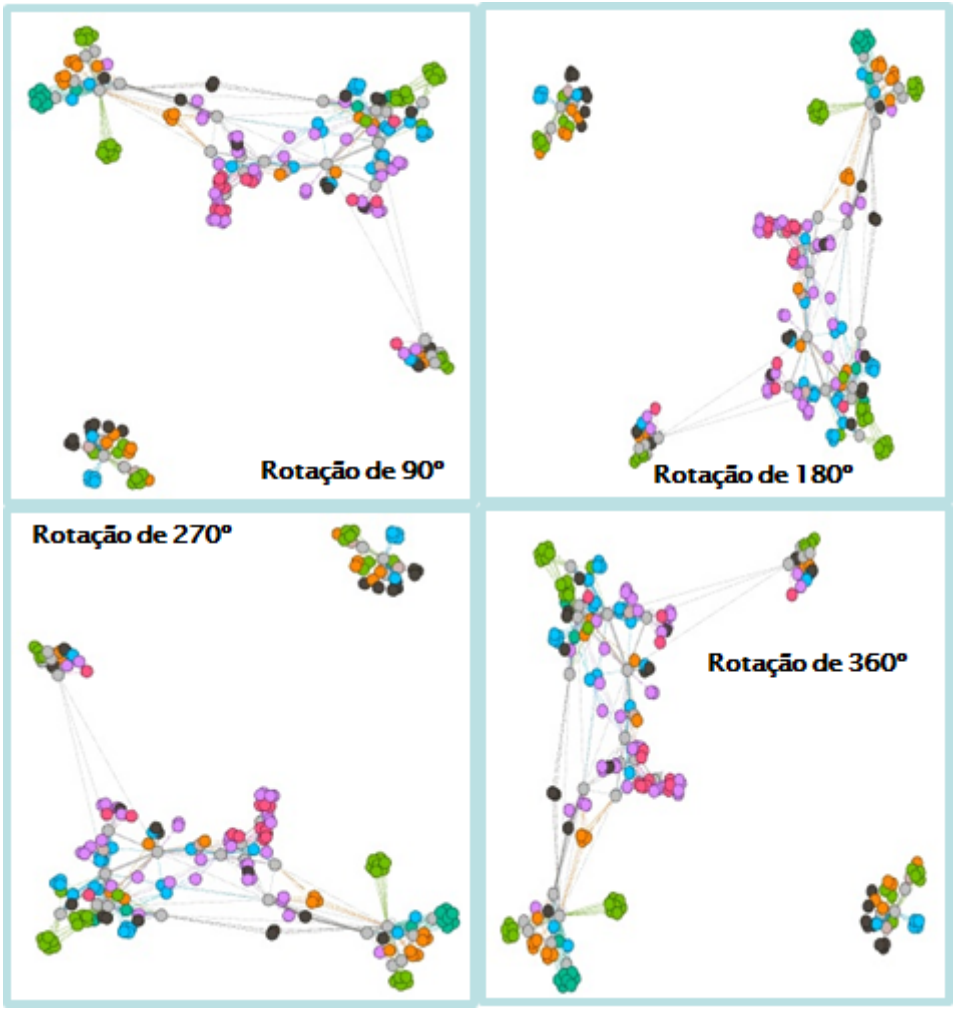


Figura C.9: Distribuição Clockwise Rotate para Rede de Coautoria, software Gephi. Fonte: Próprio autor

C.10 Leiaute Contraction e Expansion

Conforme apresentamos no Capítulo 3, Seção 3.4.2.4,os algoritmos de contração e expansão aplicam transformação geométrica nos posicionamentos dos vértices do grafo aproximando ou distanciando o vértice do centro do plano onde os mesmo foram posicionados a partir de uma escala informada, realizando movimento diagonal dando a impressão de expansão ou retração da rede (Vilas Bôas, 2016), (Khokhar, 2015).

A seguir, apresentamos o pseudocódigo do algoritmo de contração e expansão.

Algorithm 26 CONTRACTION AND EXPANSION

```
1: input graph, scale;
2: xMean←0;
3: yMean←0;
4: for index ← 0 to graph.GetVertices().Count() − 1 do
5:     xMean←xMean + graph.vertices[index].posX;
6:     yMean←xMean + graph.vertices[index].posX;
7:     index←index+1;
8: end for
9: for index ← 0 to graph.GetVertices().Count() − 1 do
10:    dx←(graph.vertices[index].posX-xMean) * scale;
11:    dy←(graph.vertices[index].posY-yMean) * scale;
12:    vertices[index].posX←xMean+dx;
13:    vertices[index].posY←yMean+dy;
14:    i←index+1;
15: end for
```

O algoritmo têm como parâmetros o objeto grafo que sofrerá mudança no posicionamento dos vértices e a escala de mudança do posicionamento do vértice (respectivamente linhas 2 e 3). Se a escala for menor que 1, indica que o posicionamento dos vértices do grafo sofrerá contração, se a escala for maior que 1, indica que o posicionamento dos vértices sofrerá expansão. Caso seja informado o valor da escala seja 1, o elementos do grafo não sofreram alterações em seus posicionamentos (Khokhar, 2015).

O algoritmo possui dois laços, o primeiro se encarrega de criar somatório das posições dos vértices nos eixos X e Y e guardar nas variáveis $xMean$ e $yMean$ (linhas 6 e 7). Após acumular os valores das posições dos vértices é feito cálculo das médias das posições conforme descrito nas linhas 10 e 11. O segundo laço se encarrega de atribuir as novas posições dos vértices, a partir da soma do fator de escala de cada posição dos vértices, armazenadas nas variáveis dx e dy (linhas 13 e 14) com o valor médio das posições (linhas 15 e 16) (Khokhar, 2015).

Apresentamos na Figura C.10 transformação geométrica de contração e expansão de um grafo, após a aplicação da distribuição Circular. Observamos a utilidade destes leiautes quando nos deparamos com grafos densos e precisamos inspecionar com detalhes os elementos do mesmo, o inverso acontece quando nos deparamos com grafos esparsos e precisamos colocar os elementos mais próximos como forma de apoiar a inspeção visual.

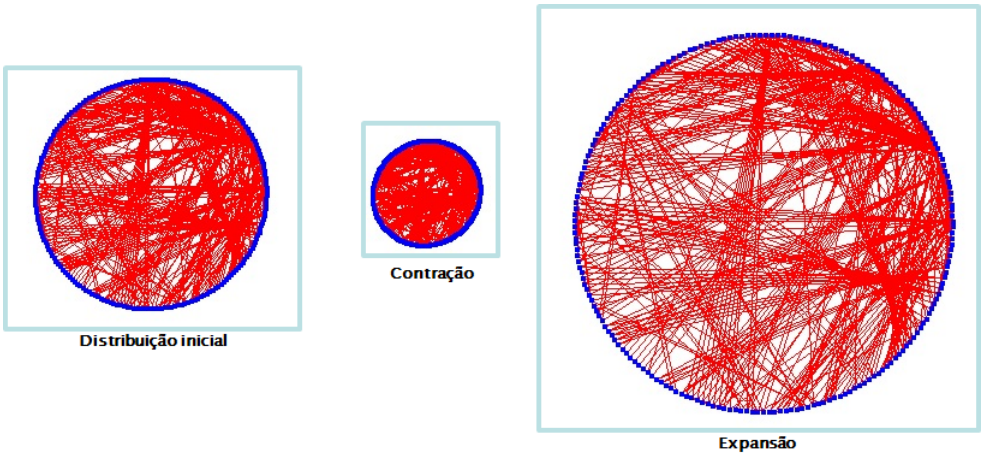


Figura C.10: Distribuição Contraction And Expansion para Rede de Coautoria, software SC Net Draw. Fonte: Próprio autor

C.11 Considerações do Apêndice

Apresentamos neste apêndice os pseudos códigos dos algoritmos de distribuição de grafos que inspiraram a construção da *NET-UML* a partir da observação das propriedades e comportamentos de cada um dos algoritmos e das categorias que os mesmos pertencem.

Apêndice D

Comparação de Leiautes de Distribuição de Grafos Implementados na *SC NET DRAW* e Ferramentas *Open Source*

Para o desenvolvimento desta pesquisa foram implementados na ferramenta *SC NET DRAW* apenas algoritmos de distribuição de grafos utilizados para visualização de redes sociais e complexas, algoritmos que estão fora deste domínio foram desconsiderados mas, podem ser revistos em pesquisas futuras.

Apresentamos a seguir resultados dos leiautes de distribuição de grafos implementados na ferramenta *SC NET DRAW* e comparação com o resultado dos mesmos algoritmos implementados nas ferramentas *Gephi* versão 0.9.2, *iGraph*, *Pajek* versão 5.0.2a e *Graph Sharp* versão 0.9.2.

Utilizamos como estudo de caso a rede real de coautoria que se refere a publicações em periódicos entre docentes, discentes e participantes externos de um programa de pós graduação da área interdisciplinar entre os anos de 2008 e 2014. Esta rede possui 204 vértices e 929 arestas.

Para teste dos algoritmos *Tree* e *Sugiyama* utilizamos uma rede social das áreas de suporte de uma instituição de pesquisa composta por 34 vértices e 34 arestas.

A comparação discutida neste apêndice tem como objetivo validar se as visualizações proporcionadas pela *SC NET DRAW* estão de acordo com as premissas estéticas dos leiautes de visualização de redes discutidos na presente pesquisa.

D.1 Algoritmos Circular

Os algoritmos de leiaute circular distribui espacialmente os vértices em *clusters*, posicionando os vértices na borda de uma circunferência, conforme discutido no Capítulo 3, Seção 3.4.2.1. Apresentamos nas Figuras D.1 e D.2 as distribuições espaciais Circular e Ego geradas pelos software *SC NET DRAW* e *Gephi*.

A distribuição circular apresenta comportamento similar entre os software, enquanto a distribuição Ego apresenta comportamentos diferentes. A *SC NET DRAW* distribui os

vértices em um formato circular, colocando o vértice analisado no centro da circunferência já o *Gephi*, utiliza o leiaute Ego como um filtro, após a seleção de um vértice de análise somente os vértices vizinhos e os vizinhos dos vizinhos são apresentados, os demais vértices são escondidos.

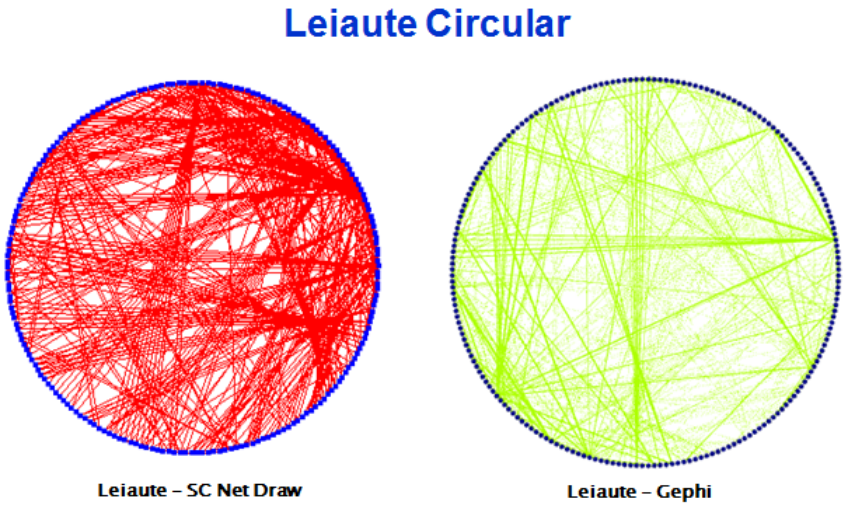


Figura D.1: Figura comparando o leiaute Circular gerado pela *SC NET DRAW* e pelo *Gephi*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

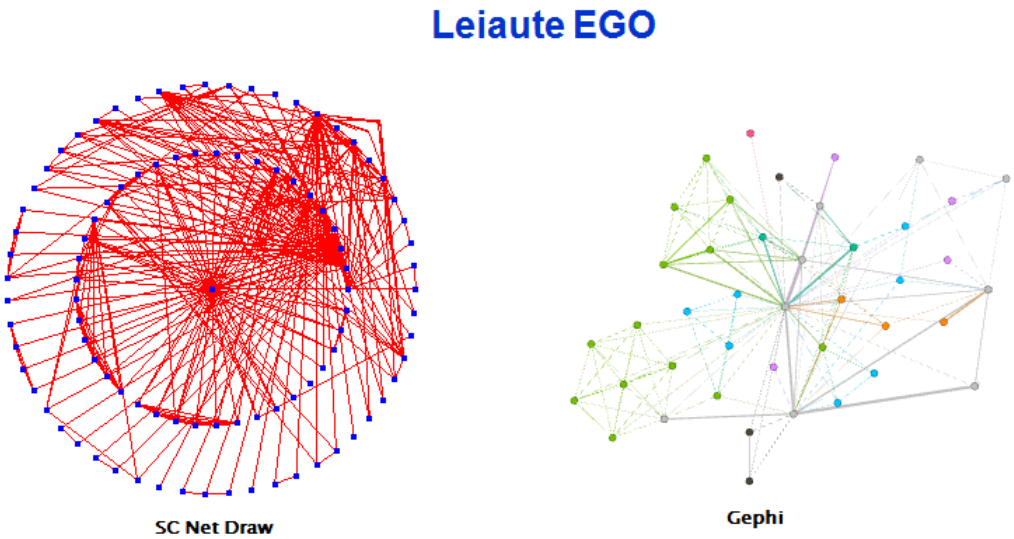


Figura D.2: Figura comparando o leiaute Ego gerado pela *SC NET DRAW* e pelo *Gephi*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

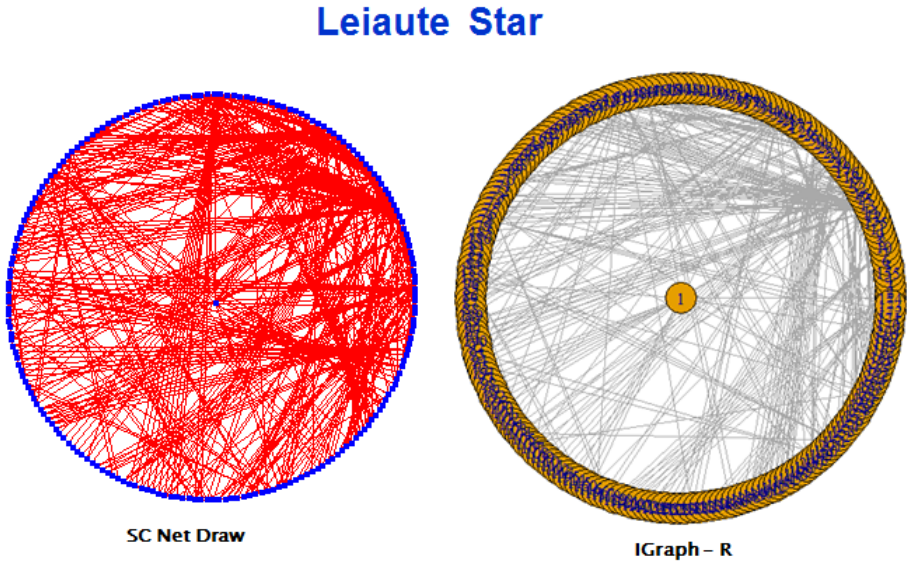


Figura D.3: Figura comparando o leiaute Fruchterman Reingold gerado pela *SC NET DRAW* e pelo *iGraph*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

D.2 Algoritmos Force Direct

Os algoritmos dirigidos por força utilizam princípios físicos para determinar o posicionamento dos vértices a partir do uso de forças de atração e repulsão, esta categoria de leiaute prioriza a criação de leiautes agradáveis em detrimento a performance de execução, conforme discutido no Capítulo 3, Seção 3.4.2.3. Apresentamos nas Figuras nas Figuras D.4, D.5 e D.7 resultados obtidos com a execução dos algoritmos *Force Atlas 2*, *Kamada Kawai* e *Fruchterman Reingold*.

A implementação do algoritmo *Force Atlas 2* na *SC NET DRAW*, levou em consideração o trabalho proposto por Jacomy et al. (2014) e implementação na ferramenta de código aberto *Gephi*, os resultados obtidos em ambas as ferramentas são semelhantes, haja visto a distribuição dos vértices no leiaute, que sugere a correta implementação.

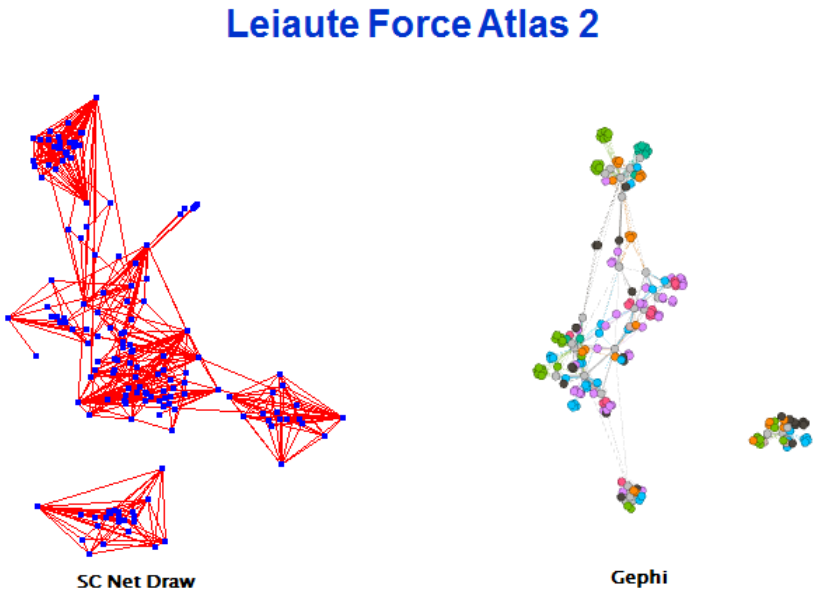


Figura D.4: Figura comparando o leiaute Force Atlas 2 gerado pela *SC NET DRAW* e pelo *Gephi*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

O algoritmo *Kamada Kawai*, que apresentamos na Figura D.5, foi implementado na *SC NET DRAW* de acordo com o trabalho proposto por Kamada e Kawai (1989) e implementação na ferramenta de código aberto *Graph Sharp*, já a distribuição proposta pela biblioteca *iGraph* e disponível na ferramenta *R Statistical Computation*, leva também em consideração o trabalho proposto por Kamada e Kawai (1989) porém, apresenta código diferente haja visto, utilizarem diferentes tecnologias para desenvolvimento, enquanto a *SC NET DRAW* utiliza o paradigma OO, a biblioteca *iGraph* utiliza o paradigma de programação estruturada.

As ferramentas apresentam diferentes resultados, observamos que o leiaute proposto pela *SC NET DRAW* apresenta em sua distribuição vértices mais espaçados em quanto na distribuição proposta pelo *R Statistical* os vértices se encontram mais próximos uns dos outros, este comportamento não garante a correta implementação do algoritmo na ferramenta *SC NET DRAW*. Como forma de validar a implementação do algoritmo *Kamada Kawai*, foi necessário analisar os resultados do mesmo na ferramenta *Pajek*, conforme apresentamos na Figura D.5, observamos que nesta implementação os vértices são apresentados também de forma espaçada, de acordo o estabelecimento das distancias teóricas estabelecidas pelas conexões dos vértices, conforme discutido no Capítulo 3, Seção 3.4.2.3, apesar de apresentar imagens diferentes, e levarmos em conta o posicionamento dos vértices, podemos sugerir correta implementação do algoritmo.

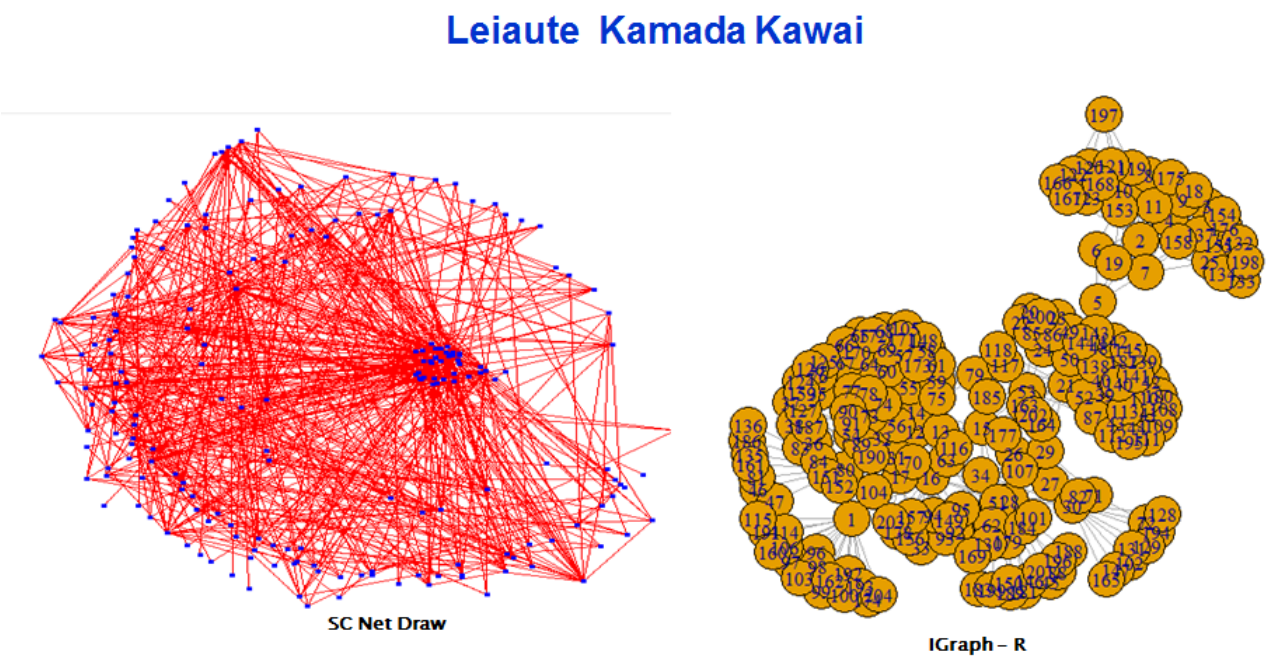


Figura D.5: Figura comparando o leiaute *Kamada Kawai* gerado pela *SC NET DRAW* e pelo *iGraph*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

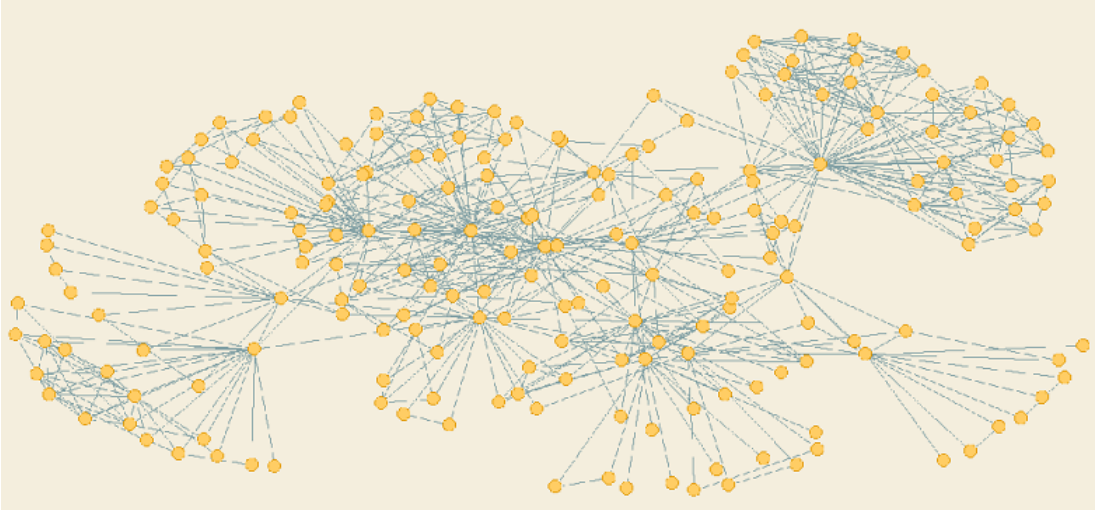


Figura D.6: Distribuição *Kamada Kawai* gerado pelo *Pajek*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

Apresentamos na Figura D.7, os resultados obtidos na execução do algoritmo *Fruchterman Reingold* nas ferramentas *SC NET DRAW* e *Gephi*. Ambas implementações tiveram como inspiração o trabalho proposto por Fruchterman e Reingold (1991) porém apresentam distribuições diferentes, enquanto o *Gephi*, apresenta distribuição mais agradável com

diminuição do cruzamento de arestas o *SC NET DRAW* apesar de distribuir os vértices de acordo com as forças de atração e repulsão propostas por Fruchterman e Reingold (1991), apresentam demasiado cruzamento de arestas fazendo que alguns vértices fiquem invisíveis. Este comportamento sugere revisão da implementação do algoritmo *Fruchterman Reingold* na ferramenta *SC NET DRAW*.

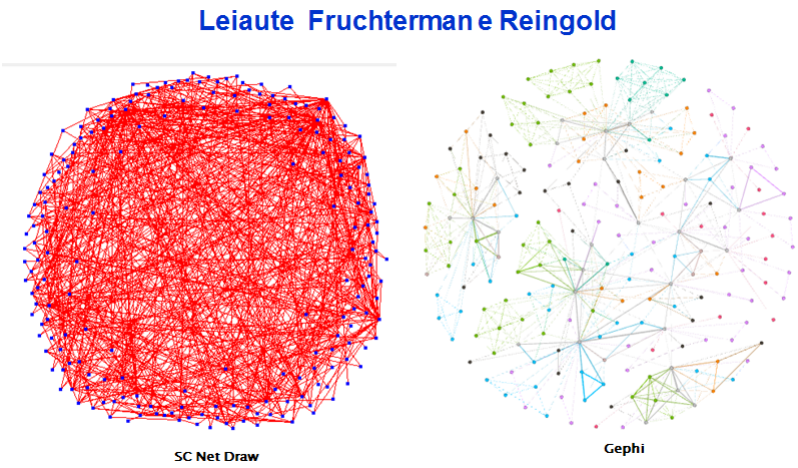


Figura D.7: Figura comparando o leiaute Fruchterman Reingold gerado pela *SC NET DRAW* e pelo *Gephi*. Rede com 204 vértices e 929 arestas. Fonte: Próprio autor

D.3 Algoritmos Hierárquicos

Os algoritmos hierárquicos são indicados para uso em grafos direcionados e conforme discutido no Capítulo 3 Seção 3.4.2.2, revela a existência de hierarquias através do desenho de grafos por camadas ou níveis. Apresentamos nas Figuras D.8 e D.9 distribuição espacial hierárquica respectivamente dos algoritmos *Tree*, proposto por Reingold e Tilford (1981) e *Sugiyama*, proposto por Sugiyama e Toda. (1980). Para teste dos algoritmos utilizamos uma rede social das áreas de suporte de uma instituição de pesquisa composta por 34 vértices, representando individualmente os colaboradores, e 34 arestas, representando as conexões entre os mesmos.

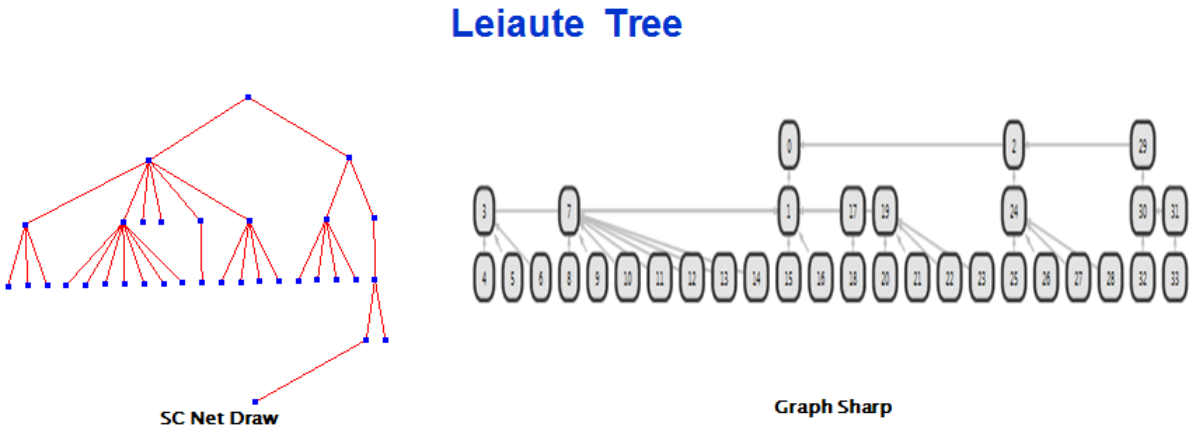


Figura D.8: Figura comparando o leiaute Tree gerado pela *SC NET DRAW* e pelo *Graph Sharp*. Rede com 34 vértices e 34 arestas. Fonte: Próprio autor

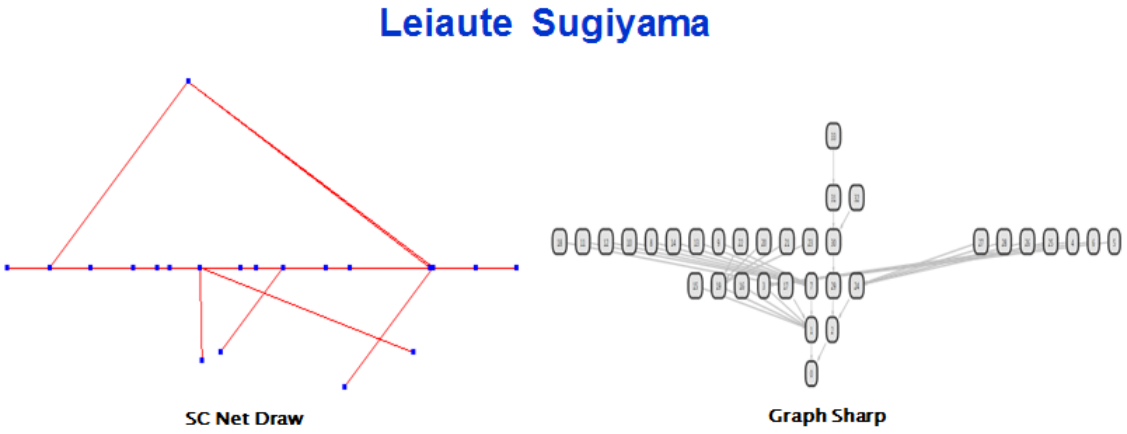


Figura D.9: Figura comparando o leiaute Sugiyama gerado pela *SC NET DRAW* e pelo *Graph Sharp*. Rede com 34 vértices e 34 arestas. Fonte: Próprio autor

Os resultados dos algoritmos apresentam distribuições espaciais hierárquicas diferentes, tanto para o leiaute *Tree* quanto para o leiaute *Sugiyama*.

Para o leiaute *Tree*, a *SC NET DRAW* apresenta como resultado uma árvore com 6 níveis a implementação na ferramenta *Graph Sharp* apresenta apenas 3 níveis, ao verificar a estrutura dos dados da rede através de análise de arquivo de dados verificamos que existem sim, 6 níveis hierárquicos na rede, indo do diretor de tecnologia, passando pelos gerentes de áreas de suporte, seus líderes imediatos, analistas e técnicos. Destacamos que a implementação do algoritmo *Tree* na *SC NET DRAW* levou em consideração o

trabalho proposto por Reingold e Tilford (1981) e implementação na biblioteca *iGraph*. Não temos como precisar a origem da implementação do algoritmo Tree na ferramenta *Graph Sharp*, pois não existe documentação técnica com esta informação porém por se tratar de ferramenta de código de aberto, pode-se ter como atividade futura comparação dos códigos com o objetivo de verificar os motivos de geração de leiautes diferentes.

A implementação do leiaute *Sugiyama* levou em consideração, em ambas as ferramentas, o trabalho proposto por Sugiyama e Toda. (1980) e implementação a partir de código aberto da ferramenta *Graph Sharp*, porém foram apresentadas diferentes distribuições espaciais, enquanto a distribuição proposta pela ferramenta *SC NET DRAW* apresenta 3 níveis, a distribuição proposta pela *Graph Sharp* apresenta 6 níveis, estándo de acordo com os dados originais da rede, conforme discutido no item anterior. A diferença nas distribuições apresentadas, sugere necessidade de trabalho futuro de revisão da implementação do algoritmo na ferramenta *SC NET DRAW*, haja visto que ambas implementações tiveram as mesmas fontes para implementação devendo desta maneira apresentar resultados semelhantes.

D.4 Considerações do Apêndice

Apresentamos neste apêndice comparação entre o resultado da implementação de algoritmos de distribuição de grafos implementados na ferramenta *SC NET DRAW* e ferramentas *Open Source* de visualização de redes sociais e complexas.

A comparação teve como objetivo validar se as visualizações proporcionadas pela *SC NET DRAW* estão em conformidade com as implementações realizadas nas ferramentas tidas como referência e respeitam as premissas estéticas de cada um dos algoritmos.

Observamos que apesar do resultado das implementações realizadas na ferramenta *SC NET DRAW* estarem em consonância com os princípios estéticos dos algoritmos de distribuição, diferem dos resultados verificados nas ferramentas de visualização de redes tidas como referência o que faz com a implementação destes algoritmos sejam revistas.

Referências Bibliográficas

Albert, R.; Barabasi, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, American Physical Society, v. 74, p. 47–97, 2002.

Alvarez-hamelin, J. I.; Asta, L. D.; Barrat, A.; Vespignani, A. K-core decomposition: A tool for the visualization of large scale networks. *CoRR*, abs/cs/0504107, 2005.

Angelis, A. F. d. *Tutorial Redes Complexas*. [S.l.]: FAPESP, 2005.

Barabási, A.-L. *Network Science*. [S.l.]: Cambridge University Press, 2010.

Boaventura, P. O. N. *Grafos Teoria, Modelos, Algoritmos*. [S.l.]: Blucher, 2012.

Borges, G. d. E. S. *Relatório de Pesquisa Uma avaliação de Métodos Orientados a Objetos e Modelos de Processo*. [S.l.], 1997b.

Borges, K. A. V. *Modelagem de Dados Geográficos: Uma Extensão do Modelo OMT para Aplicações Geográficas*. Dissertação (Mestrado) — UFMG, 1997a.

Brandes, L. C. F. U.; Wagner, D. Social networks. In: *In: Tamasia, Roberto (eds) Handbook of Graph Drawing and Visualization*. [S.l.]: CRC Press, 2013. p. 805–839.

Cardoso, C. *Orientação a Objetos na Prática Aprendendo Orientação a Objetos com Java*. [S.l.]: Editora Ciência Moderna, 2015.

Csardi, G.; Nepusz, T. The igraph software package for complex network research. *InterJournal Complex Systems 1695*, abs/cs/0504107, 2006.

Eades, P.; Gutwenger, C.; Hong, S.-H.; Mutzel, P. Graph drawing algorithms. In: *In: Algorithms and theory of computation handbook*. [S.l.]: Chapman And & Hall/CRC2010, 2010. ISBN 978-1-58488-820-8.

Eades, P.; Lin, X.; Tamassia, R. An algorithm for drawing a hierarchical graph. *Conference on Computational Geometry*, IEEE Trans. Software Eng., 1990.

Ercies, K. *Complex Networks – An Algorithmic Perspective*. [S.l.]: CRC Press, 2015.

Euler, L. Solutio problematis ad geometriam situs pertinentis. *Graph Theory 1736-1936*, Oxford University Press, USA, 1736.

Figueiredo, D. Introdução a redes complexas. In: *In: de Souza AF Jr, Meira W (eds) Atualizações em Informática., PUC-Rio*. [S.l.]: PUC-Rio, 2011. p. 303–358.

Freeman, F. L. C. Centrality in social networks conceptual clarification. *Social Networks*, Elsevier Sequoia S.A., 1978–9.

Fruchterman, T. M. J.; Reingold, E. M. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, nov. 1991.

Gabardo, A. C. *Análise de Redes Sociais Uma Visão Computacional*. [S.l.]: Novatec, 2015.

Gama, E.; Helm, R.; Johnson, R.; Vlissides, J. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. [S.l.: s.n.], 2000.

Healy, P.; Nikolov, N. S. Hierarchical drawing algorithms. In: *In: Tamasia, Roberto (eds) Handbook of Graph Drawing and Visualization.* [S.l.: s.n.], 2013. p. 409–453.

Jacomy, M.; Venturini, T.; S, H.; M, B. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. PLoS ONE, 2014.

Kamada, T.; Kawai, S. An algorithm for drawing general undirected graphs. Information Processing Letters 31 (1989) 7-15, p. 7–15, 1989.

Khokhar, D. *Gephi Cookbook*. [S.l.]: Packt Publishing, 2015.

Kobourov, S. G. Force-directed drawing algorithms. In: *In: Tamasia, Roberto (eds) Handbook of Graph Drawing and Visualization.* [S.l.]: CRC Press, 2013. p. 303–408.

Larman, C. *Utilizando UML e Padrões – Uma Introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*. [S.l.]: Bookman, 2007.

Linden, R. Técnicas de agrupamento. Revista de Sistemas de Informação da FSMA, 2009.

Maguire, R. B. Automated display techniques for linear graphs. The 4th Annual Conference on Computer Graphics and Interactive Techniques, 1977. ISSN 0097-8930.

Mancini, M. C.; Sampaio, R. F. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. Rev bras fisioter, 11(1), 83-9, 2007.

Mazza, R. *Introduction to information visualization*. [S.l.]: Springer Science & Business Media, 2009.

Mcguffin, M. J. Simple algorithms for network visualization: A tutorial. *TSINGHUA SCIENCE AND TECHNOLOGY*, 2012.

Miesen, R. Graph sharp. 2018.

Narkhede, D. S. T. P. M.; Inamdar, V. Comparative study of various graph layout algorithms. International Journal of Emerging Trends And Technology in Computer Science (IJETTCS), v. 3, 2014. ISSN 2278-6856.

Newman, M. *Networks An Introduction*. [S.l.]: Oxford, 2010.

Noack, A. Energy models for graph clustering. *Journal of Graph Algorithms and Applications*, 2007.

Omg-group, O. M. G. *OMG Unified Modeling Language TM (OMG UML)*. [S.l.], 2015.

Pereira, H. B. B. Redes sociais e complexas: Aplicações em difusão do conhecimento. In: *In: Academia Baiana de Ciências (eds) Memória III.* [S.l.: s.n.], 2013. p. 39–47.

Presman, R. S. *Engenharia de Software Uma Abordagem Profissional*. [S.l.]: Bookman, 2011.

Purchase, H. Which aesthetic has the greatest effect on human understanding? *International Symposium on Graph Drawing (Springer)*, 1997.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2016. Disponível em: ⟨https://www.R-project.org/⟩.

Reingold, E. M.; Tilford, J. S. Tidier drawings of trees. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, SE-7, NO. 2, 1981.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy f., L. W. *bject-Oriented Modeling and Design*. [S.l.]: Prentice-Hall, 1991.

Rusu, A. Tree drawing algorithms. In: *In: Tamasia, Roberto (eds) Handbook of Graph Drawing and Visualization.* [S.l.: s.n.], 2013. p. 155–188.

Six, J. M.; Tollis, I. G. Circular drawing algorithms. In: *In: Tamasia, Roberto (eds) Handbook of Graph Drawing and Visualization.* [S.l.: s.n.], 2013. p. 285–315.

Software Senai Cimatec Team. *Metodologia de Desenvolvimento de Software Senai Cimatec*. [S.l.: s.n.], 2015.

Sugiyama, S. T. K.; Toda., M. Methods for visual understanding of hierarchical system structures. *International Conference on Cybernetics and Society held in Denver*, 1980.

Tacla, C. *Análise e Projetos OO & UML 2.0*. [S.l.]: CEFET PR Departamento de Informática, 2007.

Tamassia, R. *Handbook of Graph Drawing and Visualization*. [S.l.]: CRC Press, 2013.

Tomaél, M. I.; Marteleto, R. M. Redes sociais de dois modos: aspectos conceituais. TransInformação PUC Campinas, v. 25, p. 245–253, 2013.

Vilas Bôas, R. P. *Visualização de Informação em Redes Sociais e Complexas Utilizando GuaráScript*. Dissertação (Mestrado) — SENAI Cimatec, 2016.

Wazlawick, R. S. *Análise e projeto de sistemas de informação orientados a objetos*. [S.l.]: Elsevier Editora Ltda, 2011.

Wetherell, C.; Shannon, A. Tidy drawings of trees. IEEE Trans. Software Eng., SE-5, p. 514–520, 1979.

Ziviane, N. *Projeto de Algoritimos*. [S.l.]: Cengage, 2011.

Modelagem de Algoritmos de Distribuição Espacial de Grafos: Uma extensão da UML para Aplicações de Visualização de Redes Sociais e Complexas

Claudinei Carlos dos Santos Costa

Salvador, Maio de 2018.