A Comparison of the Implementation Means for Development of Modelling Tool

Yurii Bohomaz, Oksana Nikiforova, Konstantins Gusarovs
Faculty of Computer Science and Information Technology
Riga Technical University
Riga, Latvia

***Abstract*— While Model-Driven Software Development (MDSD) is a popular trend in the software development area, it still lacks the support of the appropriate tools in many cases. This can be explained by the fact, that many proposed methods are purely theoretical and cannot be simply introduced in industry. In order to make these methods practically usable, it would be necessary to support those with the appropriate CASE tools, which in turn requires the necessity to implement such tools. Since the MDSD support tool has well-known components, its implementation shouldn't be a problem using suitable solutions for each of these components. In this paper authors study the frameworks implemented in Java programming language that can be used for the model editor development.**

Keywords— MDSD; model editor; graph editor; modelling tool

I. INTRODUCTION

In the last 10 years Model-Driven Software Development (MDSD) seems to be a popular trend. This can be proven by the amount of the research that is being done in the given area [1]. Furthermore, a statement by Object Management Group (OMG) vice president Andrew Watson [2] defines the fact that most software developers are using the modelling during the development process to minimize the costs.

Some authors [3], [4] point out that nowadays software development without any kind of modelling is possible however comparable to a manual work of 17-18 centuries because software components are produced manually and are unique. Such a uniqueness of the components is a limiting factor to their reuse possibilities despite of modern tools that support code reuse [4]. This can be explained by the fact that reusable component integration still requires manual work at the moment. Further model integration into the software development process should minimize this manual work [3] and improve the software quality and automation level, which in turn means that the modern software development should take a step forward to the MDSD.

However, to increase the role of models in the software development process tools supporting different kinds of models as well as their transformations into different models/code are necessary. Otherwise all the research and work done in this area would still be purely theoretic and not applicable to the actual software development process. Such a tool should consist of the following components [5], [6]:

- Model Editor – a component responsible for the creation and modification of the models.
- Model Repository – storage for the models that should be possible to query using different criteria.
- Model Validator – for the analysis of the developed models from the further transformation and usage point of view. This component should be able to check if the model is appropriate as well as list the errors if it's not so.
- Transformation Definition Editor – a component that allows the creation and the modification of a transformation rules for the models supported by a tool.
- Transformation Definition Repository – storage for the crate transformation rules that can be queried by the different means.
- Code File Generator – a component for the generation of the actual software system code in a defined programming language.
- Code File Parser – is necessary for the reverse engineering support during which existing software code could be transformed into the one of the supported models.
- Code File Editor – the tool for code editing, compilation, debugging etc.

By analyzing those components, it is possible to see that Code File Editor is actually the Integrated Development Environment (IDE) which can be used in a conjunction with the MDSD support tool. This means, that the necessary component count can be reduced by introducing the integration of such a tool and IDEs. Other components however must be implemented. The actual implementation of those may vary depending on the technology chosen for the tool creation, model-driven methodology being used etc. However, in this paper authors would like to compare several Java frameworks that can be used for the development of the Model Editor component.

Such a choice of the programming language and the component of a modelling tool is based on the authors' experience in such a tool development [7], [8] (one that supports so called two-hemisphere model driven approach [9] and is called BrainTool) as well as the fact that current version of the before mentioned BrainTool [8] is already written in

978-1-5090-6681-0/17/$31.00 ©2017 IEEE

Java and has the Model Repository as well as some of the other required components developed and ready to use. However, during the development of this tool authors have faced several problems in the Model Editor implementation are and had to make several decisions about the framework to support it. In this paper authors would like the present the result of such a framework case study.

The paper is structured as follows. In the second section authors define requirements for the framework that can be used to implement the model editor component of the MDSD support tool. Sections three, four, five and six provide a short description of the frameworks being compared in this paper. In the seventh section a brief comparison of the frameworks based on the support of the requirements defined in the second section is given. Section eight defines additional comparison criteria for the frameworks that can be used for the model editor implementation. Ninth section provides the results of the framework comparison using additional criteria from the section eight. Section ten introduces several examples implemented with the framework marked by the authors as the most appropriate solution for the model editor component of MDSD support tool. Finally, eleventh section contains author' conclusions as well as cover several areas of the future work.

II. Requirements for the Model Editor Frameworks

By analyzing the models used during the software development starting from the earliest ones proposed such as a flowcharts [10] or an Entity-Relationship (ER) diagrams [11] and ending with the Unified Modelling Language (UML) [12] family, it is possible to notice a common trend – almost all the models used in the software development process are a graphs. Even the UML sequence diagram can be presented as a such with objects and their lifelines being vertices and the messages – edges of the graph. However, in the former example additional rules are applied – edges in the graph have a strict order to appear which will be mentioned later. So, it could be possible to say that a framework suitable for the model editor implementation should actually be a graph visualization and editing framework. But it is necessary to keep in mind several other aspects of the software modelling when searching for one.

By further inspecting several models used in the software development process it can be seen that sometimes a model itself consists of not a single diagram (graph in terms of framework) but multiple. Even more – in some cases these diagrams need to have a way of interconnecting. For example, business process and conceptual diagrams in the two-hemisphere model driven approach [9] are linked together by connecting elements of the conceptual diagrams to a data flows (or edges of a graph) in the business process model. Another example would be a set of the UML diagrams [12]. These are used to display the different aspects of a system – static features can be modelled with a help of the UML class diagram, dynamic – with the help of UML sequence and activity diagrams etc. However, all the UML diagrams in a single software system model will share the common elements – for example classes defined in the UML class diagram will also appear in the UML sequence diagram as the message senders and receivers. So, it is possible to refine the requirements for the model editor framework by at least one additional requirement – ability to interconnect multiple graphs that are being created/edited.

A deeper look into the structure of the formerly mentioned two-hemisphere model [9] could reveal another requirement for such a framework – the ability to create the graphs without edges. The conceptual diagram in the two-hemisphere model is actually a set of the concepts describing the data types used in the system. It is not the job of the analyst to define the relationships between the different concepts but rather one of the transformation goals. So, conceptual model is being created by simply describing the concepts with their attributes without making any assumptions about how these concepts should be related. The same would also apply for the other types of models – ER diagram [11] can contain isolated entities, and UML use case diagram [12] could also contain isolated use cases etc.

Also, it is necessary to be able to create vertices of the graph that will not only contain primitive data objects (like string or integer) but should be able to hold child objects. For example, classes in the UML class diagram [12] contain attributes and methods, concepts in the two-hemisphere model's conceptual diagram [9] contain attributes etc. Strictly speaking, graph editing framework that would be suitable to serve as a backbone for the model editor should be able to handle hypergraphs as well.

By taking a look at the UML sequence diagram [12], one can define that it is possible to represent it as a graph. However, such a graph should have a fixed geometry not affected by any means of automatic layout. By analyzing this requirement as a requirement for the graph editing framework, it is possible to say that such a framework should allow the definition of the fixed coordinates and sizes of its elements as well as should provide the way of restricting manual layout. Again, when developing the UML sequence diagram, user should be able to rearrange its elements only in a limited way.

Finally, given framework should be able to import and export the graphs created/edited – since MDSD support tool should have not only the model editor but the model repository integrated with the one as well. So, next additional requirement is the ability to save/load the data user defines.

To summarize all the information provided in this section, authors would like to define the following list of the requirements for the framework that can be used for the model editor component implementation:

- Such a framework should be graph visualization/editing framework.
- It should support hypergraphs and hanging vertices.
- It should allow to create multiple graphs and interconnect their elements.
- It should be possible to define fixed coordinates for the elements of the graph as well as restrict automatic/manual layout capabilities.
- It should be possible to import/export the graph information using such a framework.

As it was already mentioned, authors of this paper are focusing on the Java programming language do to the several reasons (ready infrastructure for the transformations, existing code etc.). So, in the next sections a brief description for the several Java-based frameworks supporting these requirements is given.

A. JGraphX Framework

JGraphX framework [13] is an open-source version of the commercial mxGraph JavaScript framework. It is built on the top of Java Swing UI library and provides the API that separates the visualization from the graph model.

One of the main advantages of JGraphX is ability to write the code in the transactional manner – for example several graph modifications can be combined into "one-or-nothing" atomic changeset that later can be applied to the graph model.

Out-of-the box JGraphX supports element grouping into the so-called cells as well as creating a subgraph that is fully contained in one of the graph's elements. This is achieved by the ability to define a parent for the each of the cells which can be omitted to become a root one. From the display point of view this also allows to collapse/expand the elements by displaying parent or its children. Another advantage of using such a technique is an ability to make the available element palette in the model editor also a graph which holds the templates for the elements that are being cloned into other graphs.

JGraphX doesn't enforce any restrictions to its visual element position in the developed program window – those can be placed inside any Java Swing container which is a noticeable advantage both from tool developer's point of view (since no restrictions are being applied on the tool's architecture) as well as from the user experience perspective – this allows each user to create layout that is most suitable.

Since JGraphX is based on the separation of concerns of graph's model and its visualization, it is easy to implement graph import/export. Actually, it is also supported out-of-the-box.

Latest release of JGraphX at the moment of this paper creation is tagged v3.7.0.0 and was done on January 16, 2017 which in turn means that the framework itself is being actively developed.

B. Sirius Framework

Sirius framework [14] is built on the top of the Eclipse IDE meaning it also inherits the access to the code editor out-of-the box as well as a set of other tools the Eclipse platform offers. Sirius framework distinguishes model structure from its representation in the similar way to JGraphX.

Sirius framework also offers import/export capabilities even using the content which structure differs from the model structure it describes.

However, due to the fact that Sirius framework uses the Eclipse IDE as its backbone, software developers that want to use it will have to face possible limitations on where the GUI elements could be placed on the screen or how they can interact one with another. While in most cases this should not be a major limiting factor, rarely this can lead to some issues.

Sirius framework allows one element model to contain other elements or even submodels which is a must have for a model editor framework as it was described earlier.

Latest release of Sirius framework at the moment of this paper creation is tagged v4.1.2 and dated December 5, 2016. So, it is being actively developed as well as the JGraphX framework.

C. JUNG Framework

JUNG [15] stays for Java Universal Network/Graph Framework and is a Java based library that is primarily meant for working with graphs. In comparison to the two previously analyzed frameworks, JUNG is heavily algorithm-focused, leaving the visualization part less mature.

JUNG supports hypergraphs and hanging vertices which is a must have feature for the model editor. It also allows for model import/export as well as separates the model from the visualization which is even highlighted by JUNG deployment structure – it is being provided as a set of different libraries where visualization is an optional one.

JUNG's heavy focus on the algorithms for the graph processing brings several advantages for developing further processing of models created with it – JUNG supports graph querying as well as creation of graph transformations using rich API.

However, the main disadvantage of the JUNG is its primitive visualization module which of course is a major limitation from the model editor perspective.

Latest release of JUNG framework at the moment of this paper creation is 2.1.1 (September 7, 2016). However, it is necessary to point out that in the last 6 years there were only a few releases (v 2.0.1 is dated January 24, 2010) which means that it wasn't developed for at least some time and could also be abandoned again.

D. yFiles FRAMEWORK

yFiles [16] is a commercial framework that provides the programmatic components that are necessary for the creation of the graph editor, graph analysis and processing. It is built on the top of the JavaFX framework which is meant to be the replacement of the Swing library in Java.

yFiles utilizes the Model-View-Controller (MVC) pattern which allows the separation of the model and its visualization keeping them tight together so the changes in either part would be propagated to another.

In a way similar to JGraphX, yFiles provides a way to group the elements of the graph by assigning the parents to those. Thus it can also support the hypergraphs which was defined as a requirement for the model editor framework.

Apart from the visualization part yFiles framework provides API to implement the graph-based algorithms, to support graph import and export as well as set of ready algorithms for the graph processing.

Latest yFiles release at the moment of this paper creation is 3.0.0.4 released in 2016. The framework currently is under active development as a commercial product.

III. Comparison of the Frameworks from the Requirement Perspective

As it was mentioned in the previous section, the framework that can be used for the model editor implementation should be able to satisfy a fixed set of the requirements. Table 1 provides the comparison summary for the analyzed frameworks based on how those are supporting the required features that are important for the development of the modelling tool. The first six requirements are focused on model editor creation. The evaluation of these requirements shows that frameworks mentioned in Section 2 support all the requirements, however in some cases some additional effort might be required to achieve the necessary results. Probably, the weakest framework at this moment is JUNG since it has very limited support for the visualization. However, it provides excellent ways of both creating and reusing the graph-based algorithms which could be an advantage.

The rest of requirements can be typed as additional requirements and are defined in order to mark the more suitable frameworks from the general point of view. One of the significant attributes of every framework is its licensing conditions. From their experience in the software development authors can say that almost every piece of the software excluding the most primitive ones will contain the programming errors (bugs). Open source license could allow the developers to fix this bug by forking the appropriate framework and later creating the pull request with the fix in order to prevent the error to happen in the later versions. Another licensing issue is not so trivial. Some open source licenses (for example GNU GPL [17]) require any software that uses libraries licensed under those to be also open-source and freely distributed. This might not be an option in many cases which means that libraries licensed under such terms should be avoided. However, this factor might and will change from the project to project so authors won't focus on the question which license is "better". Only point to be marked here is that the open-source license is a preferred one.

Another criterion defined is a project state from the active development perspective. If the project was abandoned or is not supported anymore, chances are that developers using it will face additional issues and will have to perform additional work in fixing the framework being used. Usually, it is possible to submit an issue to the actively developed framework team in order to receive a fix in a form of nightly build. Again, this criterion might be related to the fact if the framework is open-source. In the commercial framework case, it might be necessary to pay additional price for the support. Even more, if the problem will be fixed, it doesn't mean yet that the developers will be able to have the patched version of the framework as soon as possible – in case of the commercial framework it might be necessary to wait for a new release while open-sourced code can always be forked or its nightly build could be picked and later replaced by a newer version with all the necessary fixes being applied. So, in this case the framework being actively maintained is more preferred.

The next important criterion is the quality of the supporting documentation. Without the appropriate documentation, additional time would be required for the developers to get familiar with the framework and to be able to start using it. High quality documentation should include API documentation describing the interface of the frameworks available to the software developers, examples on how the framework could be used in order to achieve the specific goals as well as basic guidelines for its usage.

Also, authors would like to highlight the support of the graph processing algorithms as an addition to the criteria pool. Frameworks that are to be used in a MDSD support tool should have such an ability, since in many cases transformations performed on the models are requiring such algorithms to work. When analyzing graph processing algorithm support it is necessary to define two key features the framework might have:

- Framework itself might have a set of built-in algorithms that are well known and are widely used in the graph processing area, e.g. Dijkstra's Algorithm [18]. Such algorithms might prove useful during the transformations, or even model validation – for example it might be necessary to check that all the vertices in the model graph are interconnected.

- Framework might offer an API to extend it with new graph processing algorithm implementations. Usually model transformation algorithm is built by using the new means of data processing in conjunction with well-known algorithms.

Summarizing the comparison results, authors can conclude that JUNG framework is the weakest solution among the compared ones despite the fact it offers excellent algorithm support. Basically, it can be explained by the fact, this framework is more algorithm-centered and its visualization capabilities aren't as good as in other solutions. yFiles framework so far seems to be the best solution to implement the model editor part of the MDSD support tool. However, one of the limiting factors could be its commercial licensing model which would require the payments from the software developers for obtaining the tool as well as the necessity to wait for updates in case of found bugs. Same applies to the developers possibly not being able to perform the fixes themselves. This basically leaves the authors with two frameworks to be chosen from. Sirius framework seems to be better solution since it supports the ability to interconnect different graphs out of the box. However, this solution is tied to the Eclipse platform and in some cases this might be unacceptable. Another possible issue comes with the license. By comparing these solution licenses, it is possible to note that Eclipse Public License (EPL) is not GPL-compatible which means that some other publicly available licenses might not be usable with it [23]. Another limitation is the necessity to release the resulting software under the same license in case of using the EPL. This could be something developers don't really want to happen – they might want to release the software under less strict conditions. Another aspect to mention is the fact that JGraphX framework has a slightly better rendering performance in comparison to the Sirius framework as well as an ability to render the graphs everywhere on the screen.

TABLE I. FRAMEWORK COMPARISON BY REQUIREMENT SUPPORT

Requirement	JGraphX	Sirius Framework	JUNG	yFiles
Hypergraph and Hanging Vertex Support	Yes	Yes	Yes	Yes
Multiple Graph Creation	Yes	Yes	Yes	Yes
Ability to Interconnect Several Graphs	Has to be coded separately.	Yes	Has to be coded separately.	Yes
Ability to Define Fixed Coordinates for the Elements	Yes	Yes	Can be achieved by implementing custom layout algorithm that will use additional information stored in the graph elements.	Yes
Graph Import/Export Support	Yes	Yes	Yes	Yes
Additional Remarks	Is able to use any on-screen component as a container for its components which allows for greater flexibility when creating the GUI of the model editor	Is a part of Eclipse platform that has access to all its additional features (code editor, compiler, debugger etc.) but is also tied to Eclipse based GUI	Weak visualization support, excellent support for working with graph-based algorithms. Modular design allows to use only those parts of the framework that are necessary	Out-of-the-box provides model editor sample that can be enriched by the code necessary in the appropriate case.
Licensing Conditions	BSD License [19] – open-source	Eclipse Public License [20] – open-source	BSD License [19] – open-source	Several commercial licenses [16]
Current Development State	Actively developed	Actively developed	From 2010-2016 framework seemed abandoned, however, in 2016 two versions were released. Current state is still unclear	Actively developed
Supporting Documentation Quality	Yes, current documentation contains all the necessary information and examples	Yes, current documentation contains all the necessary information and examples	Documentation is mainly related to the API and is presented in a JavaDoc format [21].	Yes, current documentation contains all the necessary information and examples
Graph Processing Algorithm Support	At the moment of this paper creation algorithms are not implemented into JGraphX core, however, its authors are aware of such a necessity [22].	At the moment of this paper creation algorithms are not implemented into Sirius framework.	Provides the set of a ready-to-use graph processing algorithms as we as an API for implementing the new ones.	Provides the set of a ready-to-use graph processing algorithms as we as an API for implementing the new ones.
Element Querying support	By identifiers and coordinates	By identifiers, coordinates and element contents	Limited, the framework itself doesn't offer ID assigning, so even identifier-based queries are not possible	By identifiers, coordinates and element contents
Rendering Performance	1	2	3	1
Additional Remarks	Offers transactional model that allows for a batch changes as well as a possibility to observe the changes to the graphs.	Allows observing changes in the graphs and invoke callbacks when one happens.	Additional advantage is an ability to implement graph decorators, for example, one already present is able to observe changes in underneath graphs and invoke callbacks.	Allows observing changes in the graphs and invoke callbacks when one happens.

As a result of this comparison, authors would like to point out that at this moment JGraphX seems to be the best solutions to build the model editors even despite some of its limitations. Also, authors would like to point out that algorithm-wise JUNG framework is still superior solution. Since the both frameworks allow to bind any data to the graph model, it is possible to use both solutions together which would offer wider possibilities during the model editor implementation. It is also necessary to point out that these two frameworks are open-source and BSD licensed which also offers the developers of the modelling software to make the changes in their code and improve both by introducing new capabilities.

IV. CONCLUSION AND FUTURE WORK

In this paper authors describe the results of framework that are suitable for the model editor component of the MDSD support tool creation. During the research, authors have defined the criteria given framework should meet, chosen the frameworks to be compared, performed the comparison and analysed its results.

In order to compare the frameworks, it was necessary to first define the requirements for it. Main requirements defined by this paper authors during the study of models used in the software development are the following: such a framework should be graph editing framework that supports both

hypergraphs and hanging vertices. As an additional requirement, such framework should allow the ways of interconnecting several graphs via relationships between those elements, should support graph data import and export and allow to define the coordinates for graph elements.

The first round of framework comparison was done based only on former requirements. However, result analysis showed the necessity to define also additional comparison criteria to be able to find the most suitable solution. These additional criteria are also described in this paper, and later their application to further analysis of the compared frameworks. Additional criteria are type of license under which the framework is distributed, its current development state (if it is maintained), quality of the supporting documentation, degree of the graph processing algorithm support, support for performing different queries on graph elements as well as rendering performance.

After additional comparison using additional defined criteria, authors propose to use JGraphX framework for model editor component implementation. While it might have some issues (for example in relation to querying and the necessity for additional coding in order to interconnect different graphs), it still offers favourable performance and additional advantages such as an ability to develop any kind of interface, since the graph drawing is possible on any on-screen element. Also, it is worth mentioning that JGraphX is licensed under the BSD license which is the most flexible among the licenses compared frameworks use.

Another fact worth mentioning is that it might be possible to combine JGraphX and JUNG frameworks in order to fill the gap in the former graph processing algorithm support. Both are licensed under the BSD license, so from this perspective it shouldn't be a problem. Also, both allow to define free-form graph model which should ease the integration of these.

Authors would like to define main directions for the future work. Since the proposed solution still lacks some necessary functionality, it has to be improved to become a better framework for the model editor implementation. Such an improvement development is one of the directions for the future work. Another subject of the upcoming research could be research of the similar frameworks developed in a different programming language instead of Java.

ACKNOWLEDGMENT

The research presented in the paper is partly supported by Grant of Latvian Council of Science No. 09.1269 "Methods and Models Based on Distributed Artificial Intelligence and Web Technologies for Development of Intelligent Applied

REFERENCES

[1] Loniewski, G., Insfran E.,Abrah S. "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development" // Scientific Proceedings of 13th International Conference, MODELS 2010, pp.213-227.

[2] Watson A. Visual Modeling: past, present and future [Online]. Available: http://www.uml.org/Visual_Modeling.pdf [Accessed: Jan. 2, 2017].

[3] Brambilla M., Cabot J., Wimmer M., "Model-Driven Software Engineering in Practice." 1edition. USA: Morgan & Claypool Publishers, 2012.

[4] Guttman M., Parodi J., "Real-Life MDA. " 1 edition. USA: Morgan Kaufmann, 2007 – 224 pp.

[5] Kleppe A., Warmer J., Bast W., "MDA Explained: The Model Driven Architecture – Practise and Promise" // Addison-Wesley, 2003 – 170 pp.

[6] Mouheb D., Debbabi M., Pourzandi M., Wang L., Nouh M., Ziarati R., Alhadidi D., Talhi C., Lima V., "Aspect-Oriented Security Hardening of UML Design Models". Springer, 2015 – 237 pp.

[7] Nikiforova O., Gusarovs K., Gorbiks O., Pavlova N., "BrainTool. A Tool for Generation of the UML Class Diagrams" Proceedings of the Seventh International Conference on Software Engineering Advances, Mannaert H. et al. (Eds), IARIA ©, Lisbon, Portugal, November 18-23, 2012, pp. 60-69 (Scopus)

[8] Nikiforova O., Kozacenko L., Ungurs D., Ahilcenoka D., Bajovs A., Skindere N., Gusarovs K., Jukss M., "BrainTool v2.0 for Software Modeling in UML", Scientific Journal of Riga Technical University: Applied Computer Systems, Grundspenkis J. et al. (Eds), Vol.16, 2014, pp. 33-42

[9] Nikiforova O., Kirikova M., "Two-hemisphere model Driven Approach: Engineering Based Software Development" // Scientific Proceedings of CAiSE 2004 (the 16th International Conference on Advanced Information Systems Engineering), pp. 219-233, 2004.

[10] Chapin N., "Flowcharts", USA: Petrocelli Books, 1971

[11] Chen P.P., "The Entity-Relationship Model: Toward a Unified View of Data." // A CM Transactions on Database Systems, 1976, pp. 9-36

[12] Unified Modeling Language (UML) [Online]. Available: http://www.uml.org/ [Accessed: Jan. 2, 2017].

[13] JavaScript Diagramming [Online]. Available: https://www.jgraph.com/ [Accessed: Jan. 18, 2017].

[14] Sirius - The easiest way to get your own Modeling Tool [Online]. Available: https://eclipse.org/sirius/ [Accessed: Jan. 18, 2017].

[15] Java Universal Network/Graph Framework [Online]. Available: http://jung.sourceforge.net/ [Accessed: Jan. 18, 2017].

[16] yFiles for Java - Java Graph Layout and Visualization Library [Online]. Available: https://www.yworks.com/products/yfiles-for-java-2.x [Accessed: Jan. 18, 2017].

[17] The GNU General Public License v3.0 - GNU Project - Free Software Foundation [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html [Accessed: Jan. 18, 2017].

[18] Cormen T.H., Leiserson C.E., Rivest, R.L., Stein C., "Introduction to Algorithms (Second ed.).", USA: MIT Press Cambridge, 2001, 1179 pp.

[19] The 3-Clause BSD License | Open Source Initiative [Online]. Available: https://opensource.org/licenses/BSD-3-Clause [Accessed: Jan. 20, 2017].

[20] Eclipse Public License - Version 1.0 [Online]. Available: https://www.eclipse.org/legal/epl-v10.html [Accessed: Jan. 20, 2017].

[21] javadoc-The Java API Documentation Generator [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html [Accessed: Jan. 20, 2017].

[22] JGraphX User Manual [Online]. Available: https://jgraph.github.io/mxgraph/docs/manual_javavis.html [Accessed: Jan. 20, 2017].

[23] Eclipse Public License vs. BSD-License comparison | vsChart.com [Online]. Available: http://vschart.com/compare/eclipse-public-license/vs/bsd-license [Accessed: Jan. 20, 2017].