Comparative analysis of DSM Graphical Editor frameworks: Graphiti vs. Sirius

Vladimir Vujović[1], Mirjana Maksimović[1] and Branko Perišić[2]

[1]*Faculty of Electrical Engineering, University of East Sarajevo, Bosnia and Herzegovina*
[2] *Faculty of Technical Scieneces, University of Novi Sad, Serbia*
vladimir_vujovich@yahoo.com, mirjana@etf.unssa.rs.ba, perisic@uns.ac.rs

Developing languages and tools which depend on Domain-Specific Modeling (DSM) methodology, represent a hot topic nowadays, but building a set of graphical tool is often very complex process which takes a lot of time and it often highly depends on developer's knowledge and reliable frameworks. Today, a leading role of Integrated Development Environment (IDE) tools on market has an open source – Eclipse platform, which became de-facto standard and primary choice for developing a DSM environment. The choice of user interface framework, to solve the traditional problems of custom modeling in an elegant fashion, strongly influences the development process and system's lifetime costs spent on maintenance. In order to realize which of the open source frameworks based on Eclipse is better choice, allowing developers to establish a very flexible graphical environment for editing the models, a comparative analysis of Graphiti and Sirius frameworks for developing a DSM Graphical Editor is done with aim to define pros and cons of each of them.

1 Introduction

A significant factor behind the difficulty of developing complex software is the wide conceptual gap, between the problem and the implementation domains of discourse, which can be reduced using Model-Driven Engineering (MDE) [1]. The key fact for MDE software development is that system is a model consistent with its meta-model [2, 3]. That model is the link between the problem domain and solution domain, and meta-model represents an abstract model of the system that describes the most common definition of the model. The key benefits of Model-driven approaches are: increased developer productivity, decreased cost (in time and money) of software construction, improved software reusability and the higher level of software maintainability [2].

Domain-Specific Modeling (DSM) methodology uses models to describe the individual components of the domain system, which is usually based on graphical or textual description. Considering that communication effectiveness can be measured by speed, ease, and accuracy in which the information can be understood [4], it can be stated that graphical diagrams are believed to be more effective than text in the communication between end-users and/or domain practitioners [5]. Therefore, models are often based on a graphical representation and supported by graphical design tools. A set of DSM tools enables user to create models relied on metamodels, and usually, based on created models, generate a certain part of code using generators.

Graphical Model-Driven Engineering tools have become extremely popular concerning the development of applications for a large number of domains from natural language processing to computer vision in bioinformatics. Using a graphical model, which is a popular and well-studied framework for compact representation of a joint probability distribution over a large number of interdependent variables [6], facilitates better understanding of a problem-domains. It can be stated that Model-Driven visualization provides model driven engineers with the tools and technologies to integrate interactive visualizations in their systems. By separating the customization and configuration of the view from its underlying model, engineers can explicitly state how their data should be displayed [7].

Today a leading role of MDE tool on market has the open-source *Integrated Development Environment* (IDE) - Eclipse, which supports a wide range of frameworks for development and usage, depending on problem type. An *Eclipse Modeling Framework* (EMF) provides an object graph for representing models, as well as capabilities for (de)serializing models in a number of formats, checking constraints, and generating various types of tree editors for use in Eclipse. The *Graphical Editor Framework* (GEF) and *Draw2D* provide the foundations for building graphical views for EMF and other model types [8]. The *Graphical Modeling Framework* (GMF), by encapsulating GEF and *Draw2D* (Fig. 1), provides a tool for creating graphical editor with a high degree of flexibility. Creation of editor in GMF is often complex and highly depends on Java, XML and Eclipse plug-in knowledge. By using *Graphiti framework*, that hides GEF's complexities from the developer and bridges EMF and GEF (Fig. 1) to ease and speed up the development of graphical editors, it is possible to design homogeneous graphical editors that visualize an underlying model based on a tool-defined graphical notation [9]. A big disadvantage of all mention frameworks (GMF, GEF, *Graphiti*) is a high level of required knowledge in domain of Java object oriented language, EMF and Eclipse plug-in development. On the other side, *Sirius*

framework, offers a solution for rapid development of Graphical tool for DSM, without need for understanding any of backend processes [10].

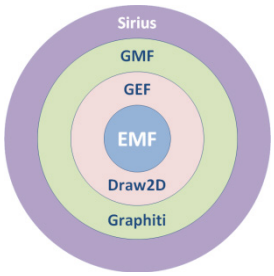

Figure 1. Hierarchy of Graphical Model-Driven Engineering tools

In this paper, the emphasis is a comparative analysis of *Graphiti* and *Sirius frameworks* for developing a DSM Graphical Editor. Thus, the rest of this paper is structured as follows. Section II describes the state of the art, namely *Graphiti* and *Sirius*. In section III a comparative analysis of chosen frameworks is performed in order to define pros and cons of each technology, while Section IV concludes the paper and states the roadmap for evaluation and other issues of future work.

2 State of the Art

Building a very flexible graphical editor for editing the models is really a labor-intensive task. In this paper, two DSM Graphical Editor frameworks, *Graphiti* and *Sirius*, are considered and compared, in order to define pros and cons of each of them.

Graphiti is framework for creating graphical diagram editors which doesn't do any code-generation and is written in plain Java. It takes a different approach than both GMF and GEF. Instead of requiring the user to use the Model-View-Controller paradigm, it introduces so-called features: creation, deletion and changing of business model elements and creation, deletion and updating of visual elements. On the other hand, *Graphiti* provides similar concepts to GEF (its internals are built on GEF), but again they are provided uniformly through features. It means that edit policies, requests and commands are invisible to the user of *Graphiti*. After entering features that are supposed to make changes to state, a transaction recorder is added to the resource tree so that one doesn't need to use commands manually. The developer generally doesn't need to deal with the state of the editor. It can be summarized that *Graphiti* provides an easy entrance to creating graphical editors through a simple and contained API [11]. It is important to mention, that for each domain object, the generator creates *Add*, *Create*, *Layout*, *UpdateFeature* etc., which altogether consist of about 400 lines of code per domain objects. This can be reduced approximately 20 times using *Spray framework* on top of *Graphiti* [12].

Sirius framework, which is built on top of GMF, is used to create, visualize and edit models using interactive editors called "*modelers*". Depending of visual representations, *Sirius* supports three different dialects (kinds of representations): *diagrams* (graphical modelers), *tables*, and *trees* (hierarchical representations), but new dialects can be added through programming [13]. Because problem-domain usually makes necessary the collaboration of people with different concerns, a *Sirius* provides possibility of analysis, roles and concerns of same data using different viewpoint on the same domain model. A *Sirius* provides tools to specify the viewpoints which are relevant for user business domain, whatever it is. Due to *Sirius* uses domain specification, which is not strictly in the scope of *Sirius*, it provides a graphical modeler for creating a DSM, which defines concepts and their relations in the abstract. After defining DSM models, *Sirius* allows easily creation of specific concrete representations of these models, and representations can be presented in more than one diagrams, tables, matrices (cross-tables) or hierarchies (trees). The representations are not static, and they complete modeling environments where user can create, modify and validate their designs. It can be logically organized in viewpoints, which can be able or disabled by end-user, with purpose to provide a different, logically consistent, view on the same model. It can be concluded that *Sirius* simplifies the product, reduces design time and rapidly increases the overall productivity of building a domain-specific graphical editor. It uses *Acceleo* [14] as recommended language for expressions' defining. By using a Java class as Java Extensions and *Acceleo* queries, defined in .mtl files, *Sirius* supports customization according to the particular user needs in form of service methods which is available inside all the representations defined in the viewpoint. Considering that *Sirius* encapsulates GMF, user can customize the program code on GMF level too. However, this is an advanced feature, because user must have a deep knowledge of GMF.

3 Which framework is better, Graphiti or Sirius?

Already performed comparative analysis of DSM solutions were discussed in several papers [11, 15, 16]. First step of these analyses is setting the criteria on which the comparison will be performed. In work [15] authors have relied on research performed by P. Mohaghegh and Ø. Haugen [16], who define two approaches for evaluation:

- qualitative approach which cover case studies, analysis of the language and the tool by experts for various characteristics, and monitoring or interviewing users, and
- quantitative evaluation based on several identified metrics (effort, understandability, usability etc.).

The author of [11], as authors of work [15], focuses only on a certain criteria, which are most relevant in the context of comparative analysis of DSM Graphical Editors. Because both frameworks, *Graphiti* and *Sirius*, become a part of *Eclipse Modeling Project* in Eclipse

Luna - 4.4 version (Graphiti v0.11 incubation faze and Sirius v1.0), a question is which of this framework is better than other and why?
An evaluation criteria is setup based on mention works [11, 15] as:

- Evaluation of applicability for supporting tree-based methods depends on manually running and testing editors, reading the official documentation and relies on experience of editors' development.
- Development time - a time spent for creating a tool.
- Maintainability - a time spent for maintaine a tool (code size, dependences, making a change to the editors).
- Customizability - customization of visualization.

3.1 Evaluation of applicability

A results for *Graphiti* framework will be taken from paper [11], updated with new version of framework and compared with *Sirius*. The main goal of this type of evaluation is to present possibility of frameworks in simple tasks.

Table 1. Evaluation of applicability for supporting tree-based methods

Requirement	Best solution
Replaces Excel-tool	-
Propagation of values	Equal
Learnability	Sirius
Easy to understand	Equal
Outline	Equal
Layout	Equal
Simple creation	Sirius
Editing	Sirius
Copy and paste	Sirius
Hiding of nodes	Sirius
Search	Sirius
Easy to read	Sirius
Zooming	Equal
Persistence	Sirius
Validation EI	Equal
Validation QCF	Equal
QCF override	Equal
Resource change tracking	Equal
Traceability	None
Major OSes support	Sirius
Free of cost	Equal
Total	*Graphiti*: 0 *Sirius*: 9

As can be seen from analysis presented in Table 1. a *Sirius* based editor provides the most of required features, which means that *Sirius* is better option for creating a DSM editor.

3.2 Development time

Both editors rely on EMF, but for simple tasks, knowledge of *Draw2D* and GEF or GMF is not required. Creating an editor in *Graphiti* framework has done by implementing a core functionalities of object in Java language (about 400 lines of code per domain object [12]), which for large DSL can be very robustness. Using a *Spray framework*, which encapsulate a *Graphiti framework*, a creation is simplified and reduced on only 20 lines of code per object [12]. A *Spray* provides three different DSLs for creating an editor: *Spray Core*, *Shape* and *Style*. Although they are simple languages, programmers must learn them, and that complicates the development process.

Unlike *Graphiti*, *Sirius* works with models which describe semantics of editors - structure, appearance and behavior of dedicated representations and associated tools. For creation of editor, no Java code is necessary. A main disadvantage is need for *interpreted expressions* which will be evaluated at runtime to provide a behavior specific to domain and representations. An expression can be written in *Acceleo* [14], *OCL* [17] or Java language.

For difficult tasks, in case of both frameworks, a deep knowledge of GEF and GMF is required (A *Sirius* encapsulates GMF instead of GEF and because of that a difficult task is simpler to realize).

3.3 Maintainability

As it mention in [11], a maintenance is measured by code size, dependencies, and making a change to the editor.

Determination of code size for *Graphiti framework* can be easily done, because every object must implements its functionality which is near 400 lines of code per domain object [12]. Using a *Spray*, lines of code can be significantly reduced. On the other hand, a *Sirius framework* works with models which describe objects. A number of code lines depend on customization and *Acceleo*, *OCL* or Java expression.

Both of frameworks rely on GEF, *Draw2D* or GMF, as well as of EMF and Eclipse platform. Because of that, both frameworks depend on core Java and Eclipse platform extension. In this case, none of analyzed frameworks has a better position than the other.

In case of customization of objects' specific features, both frameworks provide a relatively good possibility. With *Graphiti framework*, customization can be done directly on code. The main problem is to find object and part of code which will be modified. With *Sirius*, a simple modification can be done using object properties inside model. However, complex requirements typically require changes to the EMF and GMF code, what can be very complicated.

3.4 Customizability

In context of customizability of graphical editor, the emphasis is on customization of graphical objects visualization and their behavior. *Graphiti framework* customization of graphical elements is provided by using a Java code and extending an `AbstractAddShapeFeature` class. Features like *Update*, *Remove*, *Delete*, *Move*, *Resize*, *Layout*, *Connection* and *Anchors* must be implemented by extending appropriate class. A style of object is defined using predefined 2D graphical object. Using *Spray* over *Graphiti*, a definition of shape is described with *Shape DSL* which is consistently used for node elements as well as for connections. The *Shape*

DSL also defines connections and placing of objects and reduces time for objects' creation and implementation.

For diagram configuration, *Sirius framework* uses a *Diagram Description* element (inside a *Viewpoint*) and its sub-elements (which describe the layers, graphical elements and tools). The content of the *Diagram Description* is mostly made of graphical elements' mappings, organized in layers, and their associated tools. In addition, it can also contain validation rules, filters and layout configuration information [13]. There is a set of predefined styles for a node (Square, Lozenge, Ellipse, Basic Shape, Note, Gauge, Image) and Custom Style which are implemented in Java. All additional styles (borders, colors, decorations, edges, etc.) are described by properties. A *Sirius* model can be used as WYSIWYG for direct testing of created editor.

4 Conclusion

Applying DSM methodology, which uses models to describe the individual components of the domain system, is one of the main topics nowadays. A model presents and describes element of meta-model, which depends of problem domain. Building a set of tools for creating and editing these models is not usually a simple task. A lot of graphical frameworks can be found on market, and all of them have their pros and cons.

In this paper, two of open source graphical frameworks are presented and compared. Both of them belongs to EMF, and can be found in Eclipse Luna distribution.

After comparative analysis based on selected criteria, next conclusion can be stated:

- both frameworks can be used for building an user friendly graphical editor based on proposed DSL,
- both editors provide a lot of required features, but *Sirius* based editor provides the most of or the better features,
- *Sirius* uses a model for describing elements of editor instead of Java code, thus developing is faster and less error prone,
- for difficult tasks both frameworks must use GEF or GMF,
- both frameworks depend on GEF or GMF,
- a relatively good support for customization,
- *Sirius* is more customizable than *Graphiti*,
- *Sirius* is WYSIWYG.

In summary, a *Sirius* framework is a better choice for building a DSM Graphical Editors because it provides the most of or the better features than *Graphiti*. In addition, it is more customizable, allows the user to view something very similar to the end result while the editor is being created. Lastly, by using a model for describing elements of editor instead of Java code, it makes DSM Graphical Editors' development process faster and less error prone. Future work will be focused to frameworks' evaluations based on the rest of criteria proposed in [16], and testing created editors by users, in order to define as much as possible frameworks' strengths and weaknesses and making certain recommendations.

References

[1] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap", Future of Software Engineering, pp. 37-54, 2007

[2] S. W. Liddle, "Model-Driven Software Development", June 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.5995&rep=rep1&type=pdf

[3] M. Brambilla, J. Cabot and M. Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool publishers, 2012

[4] L.-O. Johansson, M. Wärja, H. Kjellin and S. Carlsson, "Graphical modeling techniques and usefulness in the Model Driven Architecture: Which are the criteria for a "good" Computer independent model? ", Proceedings of 31th Information Systems Research Seminar in Scandinavia: public systems in the future: possibilities, challenges and pitfalls, Sundsvall, 2008

[5] D. Moody, "What makes a good diagram? Improving the cognitive effectiveness of diagrams in IS development," 15th international conference of Information Systems Development, Budapest, Hungary, Springer 2006

[6] A. Deshpande, L. Getoor and P. Sen, "Managing and Mining Uncertain Data: Chapter 1- Graphical models for uncertain data," Springer 2009

[7] R.I. Bull, "Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization," PhD Thesis, University of Victoria, 2008

[8] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose, Eclipse Modeling Framework: A Developer's Guide, Addison Wesley, 2003

[9] C. Brand, M. Gorning, T. Kaiser, J. Pasch and M. Wenz, "Graphiti - Development of High-Quality Graphical Model Editors," Eclipse Magazine, [Online]. Available: http://www.eclipse.org/graphiti/documentation/files/EclipseMagazineGraphiti.pdf

[10] E. Juliot and J. Benois, "Viewpoints creation using Obeo Designer or how to build Eclipse DSM without being an expert developer?," Obeo Designer Whitepaper, 2010, [Online]. Available: http://www.obeo.fr

[11] I.Refsdal, " Comparison of GMF and Graphiti based on experiences from the development of the PREDIQT tool," Master thesis, University of Oslo, 2011

[12] F. Filippelli,S. Kollosche, M. Bauer, M. Gerhart, M. Boger, K. Thoms and J Warmer, "Concepts for the model-driven generation of graphical editors in Eclipse by using the Graphiti framework", [Online]. Available: http://spray.eclipselabs.org.codespot.com/files/SprayPaper.pdf

[13] Sirius Documentation, [Online]. Available: http://www.eclipse.org/sirius/doc/

[14] Acceleo, [Online]. Available: http://www.eclipse.org/acceleo/

[15] A. E. Kouhen,C. Dumoulin, S. Gérard, P. Boulet, "Evaluation of Modeling Tools Adaptation", hal-00706701, version 1 - 11 Jun 2012

[16] P. Mohagheghi and Ø. Haugen, "Evaluating Domain-Specific Modelling Solutions", Advances in Conceptual Modeling – Applications and Challenges, Lecture Notes in Computer Science, 2010, Volume 6413, Pages 212-221

[17] Object Constraint Language (OCL), [Online]. Available: http://www.omg.org/spec/OCL/ISO/19507/PDF