Sirius: A Rapid Development of DSM Graphical Editor

Vladimir Vujović*, Mirjana Maksimović* and Branko Perišić**
* Faculty of Electrical Engineering, East Sarajevo, Bosnia and Herzegovina
** Faculty of Technical Sciences, Novi Sad, Serbia
vladimir_vujovich@yahoo.com, mirjana@etf.unssa.rs.ba, perisic@uns.ac.rs

Abstract — **Looking through the history of software development there are two mutually agreed factors used for software process and product effectiveness measurement: the level of abstraction and the level of reusability. The essential goals, among many, that have to be achieved are: increasing developer's productivity, decreasing cost (in time and money) of software construction while preserving desired quality level and improving software reusability and maintainability. Model Driven Software Development (MDSD), which is generally based on the model-centric approach to software development, appears as a challenging paradigm. MDSD is focused on the creation of semantically rich models concerning problem and solution domains while leaving the execution domain to the model based code generators. Created models are often based on a graphical representation and supported by graphical design tools, which, in the most common case, can't be universal. Depending on problem-domain, and based on Domain-Specific Model (DSM) appropriate graphical editing tool must be created. In this paper the emphasis is put on *Sirius* framework for developing a DSM Graphical Editor, which simplifies the product specification, reduces design time and rapidly increases the overall productivity. The main advantages of *Sirius* framework usage have been illustrated by the creation of a RESTful Sensor Web Network Editor.**

I. INTRODUCTION

Software researchers and developers through years have been creating abstractions that help them program in terms of their design intent rather than the underlying computing environment and thus hiding at the same time the complexities of these environments. The history of software development is a history of raising the level of abstraction and the level of reusability [1]. A promising approach to address platform complexity and the inability of third-generation languages to alleviate this complexity and express domain concepts effectively, is to develop Model-Driven Engineering (MDE) supporting technologies [2]. MDE promotes the use of models as primary artifacts of a software development process, in early software life-cycle phases, as an attempt to handle complexity through abstractions. In that way MDE primary reduces the gap between problem, solution and execution domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations [3]. In other words, the application of an MDE process results in various models residing at various levels of abstraction. Thus, higher-level models are transformed into lower level models until the model can be made executable using either code generation or model interpretation.

The key fact for MDE software development is that system is a model consistent with its meta-model [4, 5]. That model is the link between the problem domain and solution domain, while meta-model represents an abstract model of a system that describes the most common definition of the model. Models can be used for different purposes [4]:

- descriptive (i.e., for describing the reality of a system or a context),
- prescriptive (i.e., for determining the scope and details at which to study a problem) or
- for defining how a system shall be implemented.

MDE approach relies on modeling languages which are usually divided into two large groups:

- General-Purpose Modeling Languages (GPLs) – are the languages, usually with the extensible syntax, that are designed for a wide range of domains and modeling purposes.
- Domain-Specific Languages (DSLs) – are languages usually designed with a specific purpose. They are usually developed for problems which need to be described in a specific domain. By using DSL language, the problem domain is considerably simplified and consequently its solution too. This increases productivity and enlarges the user base [6-8]. The smaller size of DSL programs compared to GPL programs also makes the use of analysis, verification, optimization, parallelization and transformation more feasible [6].

A problem with DSLs is that they are difficult to design and implement and require higher initial costs [8]. Here is where MDE comes into play: DSLs and MDE are a perfect match [7]. The MDE approach makes the specification of a DSL much easier. Creating a DSL in the MDE context is often called Domain-Specific Modeling (DSM) [9]. Unlike DSL, DSM uses models to describe the individual components of the domain system. Models are often based on a graphical representation and supported by graphical design tools. DSM tool enables user to create domain models, and usually, based on created models, generate a certain part of code. Building a graphical tool for DSM is often very complex process. It needs a lot of time for developing and it often highly depends on reliable frameworks, because building an editor from scratch is mostly impossible nowadays.

In this paper, a *Sirius* framework for developing a DSM Graphical Editor is described. A *Sirius* encapsulates

978-1-4799-4615-0/14/$31.00 ©2014 IEEE

Graphical Modeling Framework (GMF), and simplifies the product, reduces design time and rapidly increases the overall productivity. In summary, a rapid software development using a MDE approach, mainly a DSM abstraction, in cooperation with *Sirius* is illustrated by the example of creating a RESTful Sensor Web Network Editor.

The rest of this paper is structured as follows. Section II presents graphical model-driven engineering tools. The user's view of *Sirius* framework and the *Sirius* based graphical editor are described in Section III and Section IV, respectively. In section V a *Sirius* based editor for RESTful Sensor Web is presented. Section VI concludes the paper and states the roadmap for evaluation and the directions for future work.

II. GRAPHICAL MODEL-DRIVEN ENGINEERING TOOLS

Communication effectiveness can be measured by speed, ease, and accuracy in which the information can be understood [10]. Graphical diagrams are believed to be more effective than text in the communication between end-users and/or domain practitioners [11].

Graphical Model-Driven Engineering (MDE) 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 [12], 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 [13].

Today a leading role of MDA tool on market has the 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 [14]. The Graphical Modeling Framework (GMF), by encapsulating GEF and Draw2D, 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 to ease and speed up the development of graphical editors, it is possible to design homogeneous graphical editors that visualize an underlying Domain Model based on a tool-defined graphical notation [15].

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. A *Sirius* framework, offers a solution for rapid development of Graphical tool for DSM, without need for understanding any of backend processes [16].

III. SIRIUS FRAMEWORK

A *Sirius* framework, which is built on top of GMF (Fig. 1), is used to create, visualize and edit models using interactive editors called "modelers".

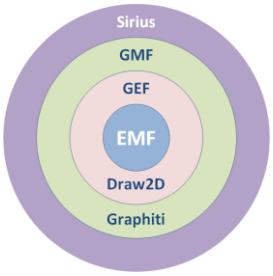

Figure 1. Hierarchy of Graphical Model-Driven Engineering tools

Depends 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 [17]. 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. Because a *Sirius* uses domain specification, which is not strictly in the scope of *Sirius*, it provides a graphical modeler for creation a DSM, which defines concepts and their relations in the abstract. After defining a 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.

In one word, a *Sirius* simplifies the product, reduces design time and rapidly increases the overall productivity of building a domain-specific graphical editor.

The basic strengths of the *Sirius* platform are [17]:

- the foundation on an open and widely used industry standard – EMF,
- adaptability to any EMF-compatible DSM,
- a strong separation between semantic and representation models,
- the support for different representations of domain models,
- easy to use and rapid development,
- the high level of extensibility.

Provided with the right configuration file, called a *Viewpoint Specification Model* (VSM), which describes the structure, appearance and behavior, *Sirius* can represent any model compatible with EMF (a meta-model description of problem-domain).

The five main concepts on which *Sirius* is based are stored under the VSM [17]:

- ***viewpoint*** – is a core element which is a logical set of representation specifications and representation extension specifications;
- ***representation*** – is a group of graphical construction which represent domain data. It also describes the structure, appearance and behavior of models. There are four representations (*dialects*) available: diagrams, tables, matrices and trees;
- ***mapping*** – strongly depends of dialect and identifies a sub-set of the semantic model's element that should appear in a representation and indicates how they should be represented;
- ***style*** – is used to configure the visual appearance of the elements;
- ***tool*** – describes behaviors mapping.

A Viewpoint elements are basicly main part of VSM model, and holds all data of building elements usually described by properties and attributes. Some of the properties are described with *Type Names* which are usually types from the represented semantic domain model. A Viewpoint also defines a *Model File Extensions*, which provides restriction of the viewpoint ability of modeling only projects which contain semantic models of the specified types.

Inside a Viewpoint element, it can be created [17]:

- ***Representation Descriptions***, for diagrams, sequence diagrams, tables and cross-tables, and trees,
- ***Representation Extensions*** (currently supported only for diagrams),
- ***Validation Rules*** (based on a meta-model rules for validation of created model, it will be applied to all representations defined inside viewpoint),
- ***Java Extensions*** (define a Java class which contains a definition of service methods).

It is possible to validate a VSM, and check correctness of specified Representations, Mappings and Tools, and find missing or erroneous element. The validation can be done for specific element or for whole model.

VSM in *Sirius* is presented with **.odesign* files. These files contains all above described elements. There is a possibility to define more than one VSM element and more than one viewpoint. Using different viewpoints, it is possible to see a model from several perspectives depending on user needs. The VSM components usually require language "description" to provide *interpreted expressions* that will, evaluate at runtime, express the domain and representations dependent behavior [17]. *Sirius* uses Acceleo [18] 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.

Fig. 2 shows an example of VSM **.odesign* files, opened in *Viewpoint Specification Editor*.

The top-level element of a VSM is always a Group element, and it has no specific semantics and serves only as a container for the other building elements. Inside the *Group*, an *User Colors Palette*, which provides a set of color for models, and Viewpoint elements can be defined.

Depending on meta-model, a *Sirius* can visualize three main building elements:

- ***node*** – object which doesn't have sub-objects, and only references or calls different objects,
- ***container*** – element which has a sub-objects,
- ***relation*** – element that visualizes a relation between two objects.

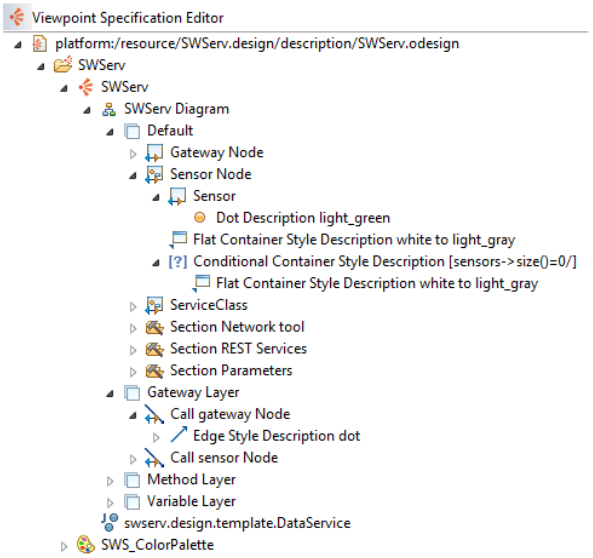

Figure 2. VSM opened in Viewpoint Specification Editor

IV. USER VIEW OF SIRIUS BASED EDITOR

A *Sirius* framework depends on Eclipse platform, therefore for users are two ways to run a created editor: as a plug-in extension for Eclipse, or as standalone editor (standalone distribution of Eclipse like editor). In both cases, a *Sirius* editor is opened in *Modeling perspective* which provides all required views, wizards and menus. A default *Modeling perspective* is shown in Figure 3 and presents a composition of several views and diagram editor (*canvas* – for presenting and editing diagram) which is the main part of *Sirius* based graphical editor and contains a drawing surface and palette with available tools.

The *Modeling perspective* provides the following views by default [17]:

- A *Model Explorer*, which is the main UI to interact with models. It shows the projects and the files they contain. It also allows viewing and manipulating of semantic models and their representations directly inside explorer.

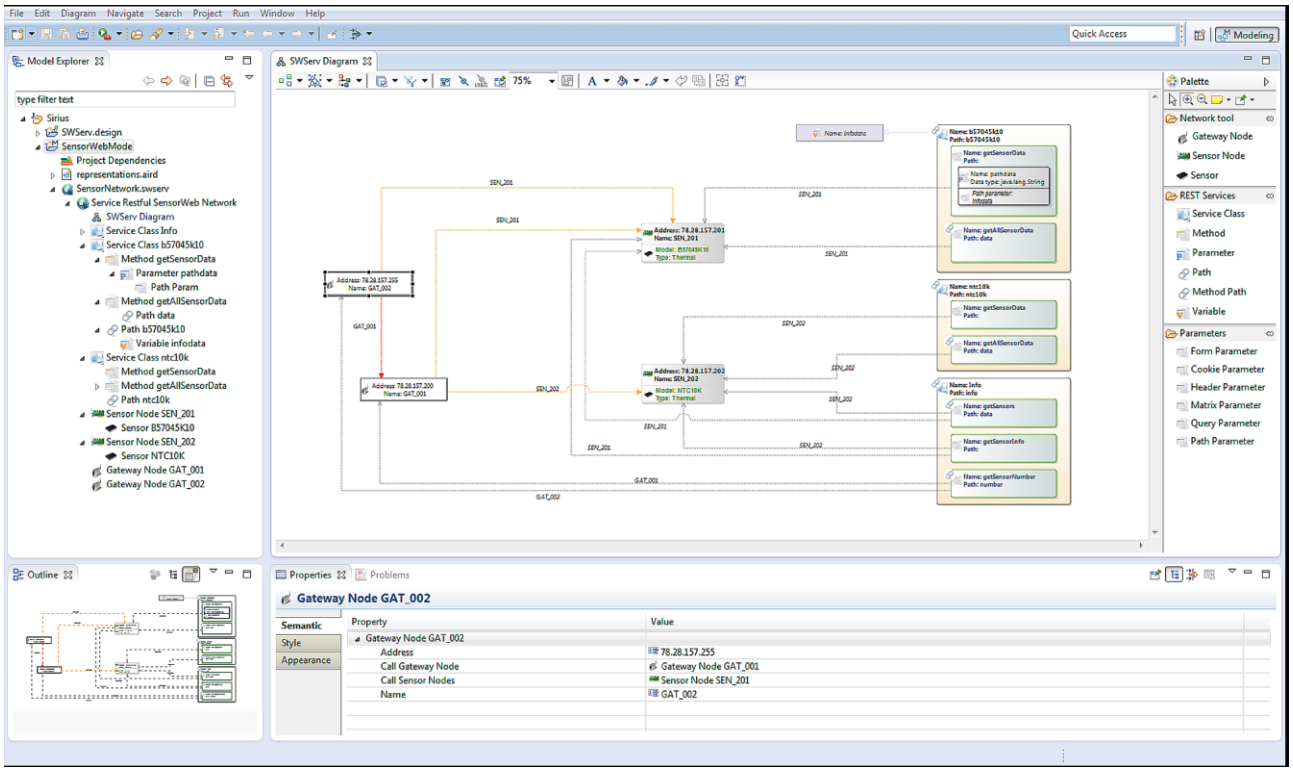

Figure 3. A *Sirius* Modeling Perspective

- An *Outline view*, which provides a structural overview of the document or model currently opened (a miniature view of the whole diagram).
- The *Properties view* gives detailed information about the currently selected element (editing depends of the selected element).
- The *Problems view* contains information markers of different severities (information only, warnings, or errors) - primarily used for validation.

In created editor which is based on Eclipse, perspective can be customized by adding, moving or removing views, shortcuts, etc. User also has a full control of editor using a well-known Eclipse function, customization, properties and keyboard shortcuts. For creating or editing a diagram it is possible to use a standard EMF editor. In this case, synchronization between Eclipse environment and *Sirius* graphical diagram is provided by default.

In *Sirius*, *Modeling projects* are responsible for storing the representation data (diagrams, tables, etc.) in special representation files with the *.aird* extension, and to organize and manage models and representations. The *.aird* file contains the data needed to display the diagrams (or other kinds of representations) but they do not contain semantic data, which are stored by the models themselves [17]. Each modeling project has a set of viewpoints which are enabled, and controls what kind of representations can be created on the semantic models inside the project. In other words, these viewpoints define the nature of project, and can be easily changed by selecting a different viewpoint in the model.

In Figure 4 is shown a dialog for selecting a viewpoints in project. Dialog is called from context menu usign a *Viewpoints Selection* action of the project itself. A *Viewpoint Selection* dialog will show all the *viewpoints* which are compatible with project (depending on installed plug-ins, or created viewpoints), and allows enabling or disabling appropriate *viewpoints*.

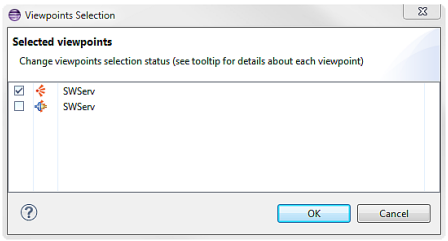

Figure 4. Viewpoint Selection Dialog

Using default *Sirius* layout manager a component layout and relation routing can be done and easily customized if needed. Sub elements can be manipulated via *Popup Bars*, shown in Figure 5, associated with a corresponding tool that can be executed in context of selected element.

Figure 5. Popup Bars

This features, speed up the creation and design process, and produce a hierarchy of objects in a diagram. An user can easily hide or show elements contained in diagrams using layer from diagram menu, what is particularly useful in case of a high density diagrams (*Arrange*, *Select*, *Align*, *Filter* and *Zoom* options).

A *Palette* has a set of standard tool like *Selection*, *Zoom In* and *Zoom Out*, and created tool splited in arbitrarily many groups and sections. A Diagram menu, with all editor specific options also has gotten.

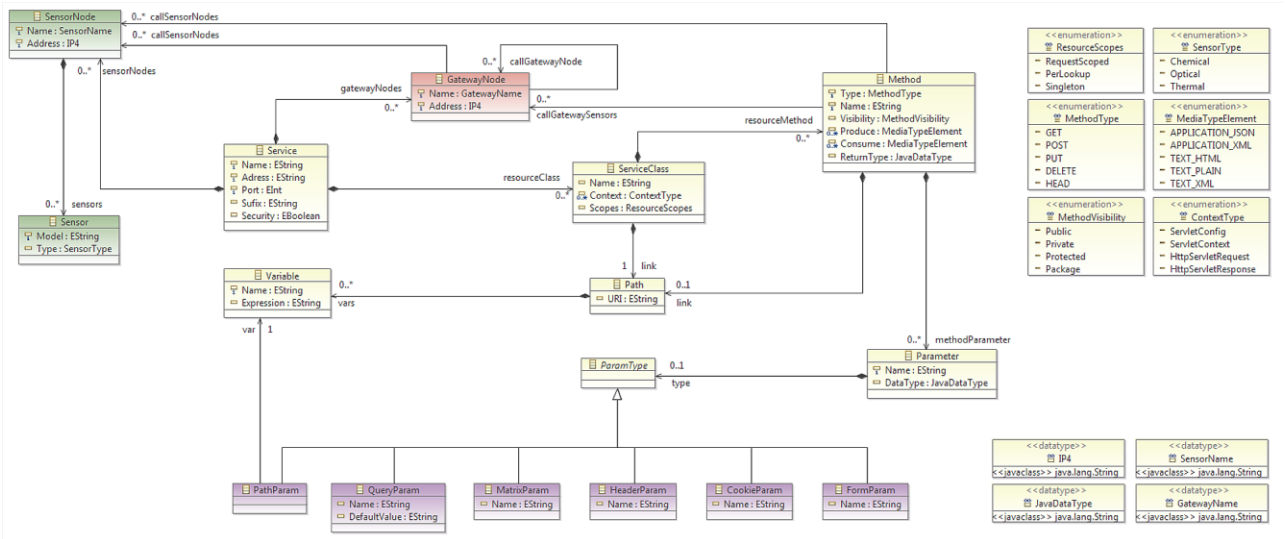

Figure 6. Ecore metamodel of RESTful Sensor Web Service

V. A SIRIUS BASED GRAPHICAL EDITOR FOR RESTFUL SENSOR WEB NETWORK

A rapid development of DSM graphical editor using *Sirius* will be shown on the example of RESTful Sensor Web Network modeling in the rest of this paper. RESTful Web services are chose because they are good option for making sensors as part of Web [19-21]. Their lighter nature is an extra advantage for resources constrained environments. The problem with sensor networks design is how to quickly and efficiently produce a complete information system that is able to: monitor, send and process data from a sensor network. Using models and tools for sensor networks design, whether they consist graphical representations, textual specifications, or both, makes the design process much easier, while the emphasis shifts to the design of networks themselves. The nature of problem-domain and its realization make covinient the usage of DSM and *Sirius* framework.

A system's structural meta-model, shown in Fig. 6, is created using EMF Ecore [14] based on the problem domain and presents one possible solution, which relies on Java object-oriented language and Jersey JAX-RS framework [22].

Figure 3 shows a solution of *Sirius* based graphical editor for RESTful Sensor Web Network. Each graphical widget in editor represents a certain element of the RESTful Sensor Web Network or service (elements like number of sensor nodes, methods, approach, etc.) and it is described by properties that describe its condition.

Main diagram is split in two elements: a RESTful service, which provides functionality of service for manipulating and communicating between sensor nodes or gateway nodes with users, and Service and Gateway nodes, which provides a concrete sensor nodes.

A Sensor Node and a Service Class, in created RESTful Sensor Web Network editor, is acting like a container for Sensors and Method, respectively, what suits a proposed DSM meta-model. Sensors can be easily added to sensor Nodes, defining a Sensor type and name, while a Method is acting as a new container for Parameter node. A Gateway Node is terminal node, and can't contain any other nodes.

An important element of diagram, are connections between two elements. A Connection represents a call of Sensor Nodes or Gateway Nodes from Method, or Sensor Nodes from Gateway Node, and calls between Gateway Nodes. In created version of editor there is no tool for creating a connection, while connection is set through element property.

Through two representations, a hierarchy representation from Model Explorer, and Graphical representation from Editor, user has an insight of problem and solution. Both representations have an access to one model, from where a documentation or code for RESTful Sensor Web Network can be easily generated.

Using the Editor, it is possible to encapsulate implementation information and, at the same time, retain the ability to easily control and create the RESTful Sensor Web Network.

VI. CONCLUSION

Model Driven Software Development (MDSD) is generally based on the model-centric, as an opposite to the code-centric, approach to the desired software development. In order to utilize this paradigm the designers have to create semantically rich models concerning problem and solution domains, leaving the execution domain to the model based code generators. In other words, from these models, the majority or even the complete code of designed software, can be automatically generated by platform independent (PIM) to platform dependent models (PDM) transformation. Semantic models are usually based on UML or some other well-known modeling language, but they can also be realized through DSL or DSM developed for the specific problem-domain cases. For the stakeholders without a detailed knowledge or experience with the underlining programming language, the ease of understanding their impacts to the development process is a must.

Building a graphical tool for DSM is often very complex process. It takes a lot of time and it often highly depends on reliable frameworks. In this paper, a *Sirius* framework is described and used for a RESTful Sensor Web Network Editor development.

Results have shown that *Sirius*, while encapsulating GMF, simplifies the end product, reduces design time and rapidly increases the overall productivity. Thus, the basic strengths of the *Sirius* platform can be summarized as follows: the foundation on an open and widely used industry standard - EMF; adaptability to any EMF-compatible DSM; a strong separation between semantic and representation models; the support for different representations of domain models; easy to use and rapid development and the high level of extensibility.

In other words, as a framework built on top of GMF, *Sirius* is highly flexible and customizable and offers a solution for rapid development of Graphical tools without any understanding of the backend processes.

The example of design and implementation of *Sirius* based Graphical Editor for RESTful Sensor Web Network have shown that designers, with no expert knowledge in specific solution-domain, can be equipped with a tool that would enable them to easily define tasks, specify network architecture and implement RESTful services.

Future work will be focused on the meta-model extension with Jersey 2.x specification, improvements of created *Sirius* based Graphical Editor and the development of an automatic code generator (necessary to adopt some additional standards) for creating partial or full RESTful Sensor Web network.

REFERENCES

[1] B. Perisic, "Model driven Software Development – State of the Art and Perspectives", Invited Paper, 2014 INFOTEH International Conference, Jahorina 19-23 March 2014 (in print)

[2] D.C. Schmidt, "Model-Driven Engineering," Guest Editor's Introduction, Vanderbuilt University, 2006, pp. 25-31

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

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

[5] T. Camilo, J.S. Silva, A. Rodrigues, F. Boavida, "GENSEN: A Topology Generator for Real Wireless Sensor Networks Deployment," SEUS, volume 4761 of Lecture Notes in Computer Science, Springer, 2007, pp. 436-445

[6] M. Mernik, J. Heering and A.M. Sloane, "When and how to develop domain specific languages," *CSURV: Computing Surveys*, 37, 2005.

[7] T.J.L. Wolterink, "Operational Semantics Applied to Model Driven Engineering, Faculty of Electrical Engineering," Mathematics and Computer Science, Enschede, 2009

[8] 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

[9] P. Hudak, "Modular domain specific languages and tools," Proceedings: Fifth International Conference on Software Reuse, pages 134–142. IEEE Computer Society Press, 1998.

[10] 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

[11] 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

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

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

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

[15] 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

[16] 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

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

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

[19] R. Raj, SOAP and REST, [Online]. Available: http://www.ics.uci.edu/~cs237/

[20] A. Ludovici and A. Calveras, "Integration of Wireless Sensor Networks in IP-based networks trough Web Services," Proceedings of 4th Symposium of Ubiquitous Computing and Ambient Intelligence, Valencia, Spain, 2010

[21] M. Rouached, S. Baccar and M. Abid, "RESTful Sensor Web Enablement Services for Wireless Sensor Networks", 2012 IEEE Eighth World Congress on Services, pp. 65-72

[22] Jersey JAX-RS framework, [Online]. Available: https://jersey.java.net/