CAMES, **28**(3): 243–261, 2021, doi: 10.24423/cames.345

Guide to Domain Specific Language Graphical Editor Prototyping

Anass RABII[1)*], Saliha ASSOUL[2)], Ounsa ROUDIÈS[1)]

[1)] *Ecole Mohammadia d'Ingénieurs, Siweb, E3S, Mohammed V University in Rabat*
Rabat, Morocco
*Corresponding Author e-mail: anassrabii@gmail.com

[2)] *Ecole Nationale Supérieure des Mines de Rabat,*
Siweb, E3S, Mohammed V University in Rabat
Rabat, Morocco

Model-based systems engineering (MBSE) is a methodology that supports the use of models to better analyze and understand complex systems and create quality and cost-efficient products. The analysis is facilitated by platforms that support formal graphical modeling and provide complementary modules for testing, validation, code or documentation generation. Thus in specialized fields, researchers create domain-specific languages (DSLs) for their niche purposes. In systems engineering, these DSLs can be created through the extension of the standard modeling language SysML. However, these DSLs do not possess modelers unless they are renowned. Therefore, they cannot benefit from advances in the MBSE tools. Our study aims to provide a graphical editor prototype for all DSLs to allow access to the MBSE tools. Using the analytic hierarchy process (AHP) method we establish that Eclipse Papyrus is the best plugin to use due to its extensibility, the richness of the Eclipse modeling platform and ease of use. Next, we provide a step-by-step guide to incorporate any profile in SysML Papyrus as an extension allowing to model any DSL. This guide is illustrated by an example taken from the domain of urban planning.

Keywords: DSL, SysML Profile, AHP, Graphical Editor, Eclipse Papyrus, MBSE.

1. Introduction

Model-based engineering (MBE) is a formal approach centered on models as a foundation of requirement specification, analysis, design, implementation and verification of a specific ability and/or product all along its lifecycle [1]. MBSE reiterates the same philosophy as the one applied to systems engineering. In addition, MBSE addresses system-specific aspects such as system architecture, constraints, flow or requirement traceability [2]. In fact, its central concept, modeling, is defined as simple graphical or mathematical representation of

a phenomenon or a system including its structure and relationships [3]. Models are created with the goal of simplifying the understanding and study of complex concepts. In turn, models simplify decision-making and analysis. In that respect, MBSE seeks to formalize system design through the use of models to increase quality and identify and treat risks that might arise during development by detecting defects early. This results in better in-depth design using iterative or parallel development cycles, and a better understanding of each component's structures, roles and relationships without increasing costs [4].

The formalism provided in the MBSE approach relies on standard languages with well-defined graphical notations, syntax and semantics. Some of the prominent modeling languages include UML [5] for software engineering, SysML [6] for systems engineering, MCAD and ECAD for mechanical and electronic computer-aided design [7] and other niche DSLs [3]. These languages have various graphical editors such as the CAD series and Hopex [8], Enterprise Architect [9], Sirius [10] and Eclipse Papyrus [11] for UML and SysML. The main benefit of these tools is the ability to create models that comply with the syntax and semantics of the language and the domain rules. The crucial functionality of graphical editors is providing supportive features all along the engineering life cycle such as testing, validation or artifact generation. Thus, creating DSLs to model concepts and test them thereafter would shorten the experimentation and prototyping loop used in iterative development cycles.

In fact, the UML standard [5] is a prime example in the MBE. UML was designed to support creating, efficiently and effectively, software systems that would have required exorbitant costs [12]. It provides a holistic set of concepts in the form of diagrams that describe how software systems are structured and how they behave. Thus, UML has revolutionized software engineering and has enabled to create many previously impossible projects [13]. Essentially, UML is extendable, allowing the creation of new modeling languages. It is possible to add newer concepts to the UML meta-model through associations called "stereotypes". Thus a new Domain Specific Modeling Language (DSML) could be defined through a set of stereotypes called a profile. This extension feature was a catalyst of progress within MBSE. A notable UML extension is SysML [6], which adapts UML to the systems engineering paradigm. SysML was created to deal with the increase of system complexity and heterogeneous nature of system components as well as to provide a core base of concepts to model all sub-fields of systems engineering. In 2017, SysML was published by the International Organization for Standardization (ISO), making it a standard modeling language in its own rights. As a result, it became a solid basis for creating systems engineering's new modeling languages.

Moreover, a modeling language's maturity and popularity are also reflected in its modeling tools. In fact, UML modeling platforms gained in functionalities

such as constraint verification modules (object constraint language – OCL) [14], documentation generators (JavaDoc, Sphinx) [15, 16], version control modules (CVS, Git) [17, 18], modeling frameworks (GMF, EMF) [19], and code generators of different languages (Acceleo) [20]. This further enriches users' toolbox and facilitates the creation of higher quality products in a shorter time. On the other hand, SysML is relatively new compared to its predecessor. It possesses its own graphical editors (Papyprus 4 SysML, Cameo Systems, IBM Rhapsody, Modelio etc.) and benefits from the same frameworks and some but not all complementary modules available for UML (EMF, GMF, OCL, Xtext, etc.) [21]. One of the discrepancies is the creation of graphical editors for SysML-based DSLs. For example, the Eclipse IDE provides the necessary structure in GMF and EMF to describe how modeling languages are defined graphically and semantically. Thus, plugins such as Sirius or Obeo have enabled the creation of graphical editors for UML profiles, given UML's popularity. Although open source SysML modelers are extendable and pave the way for adapting them for DSLs, still technical knowledge is required when annotated Ecore meta-models are used in the case of Eugenia [22]. Moreover, promising projects for automatic graphical editor generation such as AMIGO [23] are discontinued. In the case of smaller-scale projects, as in most DSLs, creating graphical editors or a prototype should not be costly or time-consuming for the modeling language to develop.

This paper presents a guide to create an accessible modeler prototype for SysML-based DSLs. This modeler prototype would also be applicable in the case of alteration to the SysML profile or for single-time use. First, we present the SysML language and its extension mechanism for DSL creation. Then, we choose an adequate existing SysML modeler to extend using the analytic hierarchy process (AHP) technique. AHP allows rigorous pairwise comparisons between alternatives and criteria. Next, we detail the necessary steps required to adapt the chosen modeler to any defined SysML-based DSL applied to the urban planning domain. This field deals with the management of space to provide a good living to all citizens while reflecting their culture and adequately using and preserving resources of the land [24]. However, the different nuances of each aspect of the urban area such as ecological, economic, cultural, sustainability or functional considerations introduce complexity into the urban planning process. These nuances translate into requirements that the urban planner must satisfy, such as providing enough facilities and good mobility for citizens, imposing building height limits to have enough sunlight, and using local and recyclable resources. The challenge in the urban planning process is, beside respecting locally mandated regulation, to design in accordance with one or many theories, visions, movements or styles. In fact, the urban planning field offers a variety of approaches to deal with each nuance, such as Friedman's spatial urbanism for modular building [25], Petruccioli's Islamic and Mediterranean influence on

urbanism [26] or Le Corbusier's functional city in the Athens Charter [27] on post-war rebuilding. Each urban planning theory proposes a different vision that handles different concepts. According to Berardi [28], each study of the urban fabric can lead to the discovery of previously unconsidered concepts that are important to the urban space. Thus, the usage of a modeling tool, in which the meta-model can continuously be altered, would benefit urban planning using SysML as the basis for behavioral, structural and requirements modeling. Our paper does not aim to provide a thorough or complete meta-model for urban planning but to provide a platform where urban planners can model urban composition and behavior according to their vision and analyze their designs with respect to their requirements.

The SysML fundamentals , including diagrams and profiling, are presented in Sec. 2. Next, for the AHP method, the relevant criteria to select the adequate platform for extension are presented in Sec. 3. Lastly, the steps of our guide, illustrated with an example about urban planning modeling to showcase the role of models in solving problems such as the impact of climate change or overpopulation on urban area design, are detailed in Sec. 4.

2. SysML fundamentals

SysML is considered the standard modeling language in the systems engineering paradigm [6]. This is in part due to the richness of its models allowing formal representation of both system structure and system behavior, as shown in Fig. 1. First, system composition can be defined through the block definition

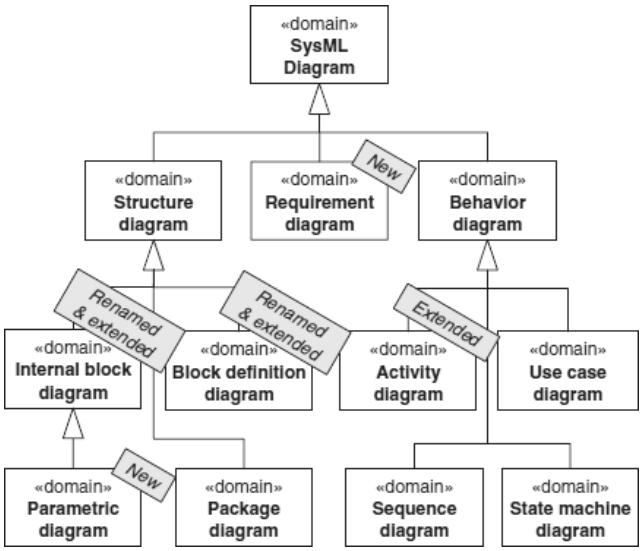

Fig. 1. SysML diagrams.

diagram, where a block is a generically defined system component. Blocks have a similar semantic to that of a class and interact through the same associations (composition, inheritance, etc.) but possess more compartments such as constraints and references. Each block can also be deconstructed and modeled by nested blocks to express block composition and relationships between properties. The nested block notation can also be used on its own in an internal block diagram. For example, a car can be represented through interacting blocks such as the engine, the wheels, the drive shaft, etc. However, breaking down the engine and detailing its parts is done with an internal block diagram.

The package diagram remains unchanged from UML, allowing the modeling of complex structures where a package can stand for the collection of design elements it contains. On the other hand, the parametric diagram models the principles and laws that bind block properties in the form of constrains such as the laws of physics (torque, friction, etc.) or performance expectations (energy consumption). Secondly, behavior diagrams model the dynamism of a system. The use case diagram describes the system's functionalities and the interactions with the prescribed actors outside of the system. State machine diagrams describe a structure's change depending on a given event. It models the necessary sequencing of conditions that must be met to reach the desired state. The activity diagram represents flows of actions and operations. It has the ability to describe actions and the interchanged information that occurs from step to step. Using conditions and timers It can even represent the continuous flow of physical elements using conditions and timers. Returning to the car example, the activity diagram can model the steps starting from ignition until the car's movement while detailing the energy flow. The sequence diagram describes the timing and nature of interactions between different system components. Lastly, the requirement diagram is an entirely new addition to model non-functional requirements, where functional requirements are modeled through use cases. The need for requirement modeling in UML arose in practice, but only the requirement diagram implemented in SysML is considered a standard. Requirements possess the same semantics as classes while they also possess inter-requirement relationships such as duplication, derivation and composition. This diagram also models what design elements aim to satisfy the requirements and test cases that verify them.

As previously mentioned, SysML is a standardized UML extension. In fact, UML 2.0 has expanded further outside its expected application field in software engineering into fields such as business process modeling. By taking a closer look on the systems engineering paradigm, it became clear that this field needs a unified modeling language of its own, later expressed as a request for proposal (RFP) called UML for systems engineering [29]. In order to create the first modeling language suitable for SE, its concepts needed to be independent of specific

fields, reuse UML concepts and use the profile and extension mechanisms. These extension mechanisms are contained in the profiles package allowing the UML meta-model to be tailored to different circumstances and purposes. The profile extension mechanism is consistent with the object management group's (OMG) meta-modeling architecture called meta-object-facility (MOF) [30] and guarantees that all modeling languages resulting from the UML profiles are consistent with this architecture as well. Profile diagrams provide the capability of adding elements in the MOF layers, thus defining new models through stereotyping. The definition of stereotypes adds semantics to an existing UML model element. This newly defined element is linked to the meta-class of the original model element using the extension association. For example, the new central concept within SysML is the block. It is an extension of the model element class (Fig. 2). In order to use these stereotypes, they should be grouped in a package called profile and linked to a model. These new sets of stereotypes related to blocks are what transforms the class diagram into the block definition diagram. As a result, the extension mechanisms allow to create new tailored modeling languages that reuse the existing UML and SysML concepts. The genericity of the base concepts fully enables adaptation, further facilitating the design of niche higher quality products for specific domains. For example, its semantics can be extended to cover engine parts and characteristics instead of using a generic block.

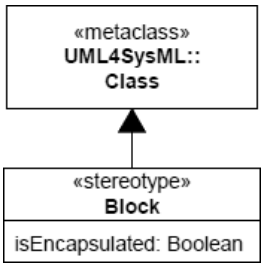

Fig. 2. Block stereotype.

3. Graphical editor comparison

As of now, we have come across a large variety of solutions that support model-based engineering. Yet only the prominent modeling languages have a diversity of choice between solutions, access to supporting tools and modules on their platform and possess sufficient documentation for support. Our objective is to guide the creation of a customizable prototype of a graphical editor that:

- adequately models SysML profiles with respect to the existing syntax and semantics of the language,
- possesses a modular platform that is rich in complementary tools,

- possesses a reasonable learning curve,
- provides an adequate documentation and support for users with limited expertise.

3.1. Modeler selection criteria

Since the functionality of creating graphical editors for SysML profiles is not provided, it has to be created as an extension of an existing SysML modeler. To that end, we use the AHP [31] method to rigorously find an adequate choice amongst different alternative modelers depending on a set of criteria. First, we define the selection criteria; the choice of an appropriate modeling tool to extend is made based on: **extensibility**, **modularity**, **support and documentation** and **learning curve**. Our objective is to spread the use of models to all systems engineering subsets that are not necessarily related to software engineering. Therefore, these criteria will help to evaluate the usability of the modeler.

3.1.1. Extensibility. In fact, we ascertain the different modeling alternatives for the convenience of their extension mechanisms by judging the requirements needed beforehand to create a working graphical editor. The appraisal reflects the degree of effort and technical skillset required to extend the existing tools. This differentiates tools that provide extension mechanisms from those that require source code modification or even the need for independent third-party projects.

3.1.2. Modularity. We also evaluate the capacity of the modeling platforms to append additional modules that support MBSE and the amount of existing complementary tools and plugins. The appraisal reflects the number of external functionalities from which the modeling tool can benefit. An open and modular platform would allow for the creation and addition of novel and diverse services. This ensures that the prototype creates benefits fully from the progress made within the MBSE paradigm.

3.1.3. The support and documentation. This criterion is crucial for the success of the prototyping guide. In fact, the lack of expertise in plugin development and software engineering, in general, might hinder the usage of models by more fringe subcategories within systems engineering. The appraisal reflects the maturity of the modeling tool and its usage by the community.

3.1.4. Learning curve. We believe the usability of the modeler also rests on the skills required to pilot it. In fact, the definition of the profile to create a new modeling language requires an understanding of what the SysML diagrams provide. We do not anticipate that the modeler requires complex computer science

skills such as coding. A modeler for any language can be created from scratch or by adapting the source code of an open-source modeler. Therefore, this appraisal reflects the amount of skills required to make and use a functional modeler to be adapted.

3.2. Preliminary step: modeler selection

First, we must decide which modelers will be taken into consideration for an extension. We opted for the open-source and free modeling tools such as Eclipse Sirius, Eclipse Papyrus and Modelio, excluding prominent commercial tools such as Magic Draw [32], Enterprise Architect and IBM Rhapsody [33]. This first choice is motivated by the availability of these tools for all users which also reflects extensibility. Our preliminary selection, Eclipse Sirius, is an open-source Eclipse project designed to create graphical modelers for generic use. Leveraging EMF and GMF, the created modelers can define models, trees or graphs from declarative configuration projects. These modelers can then be easily deployed as Eclipse plugins for reuse.

On the other hand, Modelio is a modeling environment that provides a platform for a wide range of models (UML, SysML, and BPMN). It also features a plethora of existing modules that support model transformations (into Java or WSDL code) or support architectures and frameworks (SOA, TOGAF). Moreover, users can also define their own model extensions using Java API. The last choice is Eclipse Papyrus, an environment dedicated to UML and SysML modeling. It acts as the interface between diagram editors and model-driven engineering tools as well as support for profiling.

3.3. The analytical hierarchy process

The AHP method relies on pairwise comparisons between criteria, and next between each alternative relative to those criteria. These comparisons are described through judgment matrices expressed in (1):

$$\mathbf{A} = (a_{ij}) = \begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{i1} & \vdots & a_{ij} & \vdots & a_{in} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{pmatrix}, \tag{1}$$

where

$$a_{ij} > 0; \quad i, j = 1, 2, ..., n,$$

$$a_{ii} = 1; \quad i = 1, 2, ..., n,$$

$$a_{ij} = 1/a_{ij} \ (i \neq j); \quad i, j = 1, 2, ..., n.$$

Each entry in the matrix represents the priority score the element i has over the element j, taking values from 1 to 9 from the Saaty point scale. Each matrix yields the relative weights of decision elements by calculating the eigenvalue and eigenvector. Moreover, every matrix has a consistency ratio (CR) that should be under 10% that is calculated from the eigenvalue and the matrix size.

Step 1: Identifying the weights of each criterion

The first matrix, depicted in Table 1, represents the results of criteria pairwise comparisons. The last column contains the weights of each criterion. For example, the modularity criterion is much more valuable than support & documentation, which is highlighted by the value (7). This results in a bigger gap between the weight given to modularity compared to support ($0.569 \gg 0.061$), which means it will be more influential in the choices.

Table 1. Pairwise criteria modeler comparison matrix.

	Extensibility	Modularity	Support & documentation	Learning curve	Weights
Extensibility	1	1/3	5	3	0.264
Modularity	3	1	7	5	0.569
Support & documentation	1/5	1/7	1	1/2	0.061
Learning curve	1/3	1/5	2	1	0.106
Consistency ratio = 2.5%					

Step 2: Tool pairwise comparison for each criterion

Table 2 shows the modelers' ranking regarding *extensibility*. We observe that Eclipse Papyrus is clearly distinguished with its 0.731 priority. This is because has the most practical way of creating these extension through the functionalities of adding profiles to the diagrams on hand as well as configuring the palette to allow the addition of the stereotypes for common usage. On the other hand, the creation of the entire configuration file for SysML and then adding the DSL's stereotype is feasible but impractical because of the sheer size of the SysML

Table 2. Pairwise extensibility modeler comparison matrix.

	Eclipse Papyrus	Eclipse Sirius	Modelio	Priority
Eclipse Papyrus	1	5	7	0.731
Eclipse Sirius	1/5	1	3	0.188
Modelio	1/7	1/3	1	0.081
Consistency ratio = 6.8%				

profile. Lastly, Modelio remains the most constraining choice of the three in terms of requirements to add modules.

As shown in Table 3, we observe that Eclipse Papyrus and Eclipse Sirius are almost similar in terms of modularity since they are on the same platform. A slight edge was given to Eclipse Papyrus since it interfaces with non-GMF-based tools. Modelio, on the other hand, is quite limited by its own platform.

Table 3. Pairwise modularity modeler comparison matrix.

	Eclipse Papyrus	Eclipse Sirius	Modelio	Priority
Eclipse Papyrus	1	2	6	0.577
Eclipse Sirius	1/2	1	5	0.342
Modelio	1/6	1/5	1	0.081
Consistency ratio = 3%				

In terms of a learning curve, we find graphical modeling of the profile to be easier than creating a configuration file that represents SysML and the desired profile, or completely programming the desired module. Thus, as shown in Table 4, the highest priority is given to Eclipse Papyrus.

Table 4. Pairwise learning curve modeler comparison matrix.

	Eclipse Papyrus	Eclipse Sirius	Modelio	Priority
Eclipse Papyrus	1	4	6	0.691
Eclipse Sirius	1/4	1	3	0.218
Modelio	1/6	1/3	1	0.091
Consistency ratio = 5.6%				

Lastly, the comparison shown in Table 5, between available documentation and online support shows that both Eclipse plugins are heavily used and that help via online forums is abundant, coming from users and developers. In comparison, the creation of modules in Modelio is only described by the "Hello World" tutorial.

The final result of this comparison yields that Eclipse Papyrus is inherently the adequate choice depending on our criteria. The following section will detail

Table 5. Pairwise support and documentation modeler comparison matrix.

	Eclipse Papyrus	Eclipse Sirius	Modelio	Priority
Eclipse Papyrus	1	1	4	0.444
Eclipse Sirius	1	1	4	0.444
Modelio	1/4	1/4	1	0.111
Consistency ratio = 0%				

our guide through a practical example showing how a prototype for a SysML profile modeler can be achieved.

4. DSL prototyping guide

In order to have access to the extension feature through the use of graphical models, we use Eclipse Mars [34] for modeling as it is the last version with the "configure palette" option. The aim is to create a graphical editor to model the diagrams of DSLs. In our case, the domain is urban planning. The targeted diagrams are the requirements' diagram and the block diagram. So instead of using regular blocks, we introduce concepts such as housing, equipment and roadways.

4.1. Phase one: environment and profile setup

First, to set up the environment, download Papyrus SysML 1.6 from the Eclipse Market.

Fig. 3. Papyrus SysML link.

Next, create a Papyrus modeling project to create the profile diagram that defines DSL. The palette has the option "Import Metaclass" that opens a window containing all the UML meta-model elements. In order to add an element from the SysML meta-model, the user can right-click the root element in the model explorer and "Import the registered profile", as shown in Fig. 4.

The SysML stereotype needed to be extended can be dragged and dropped from the model explorer, for example, the stereotype Requirement. For this

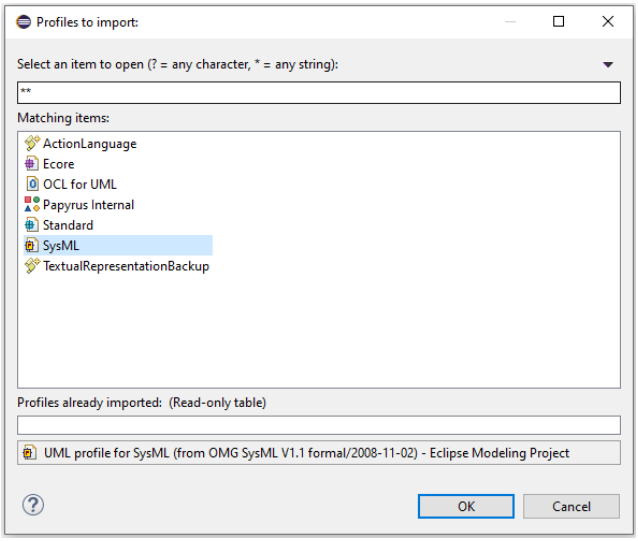

Fig. 4. Registered profiles interface.

guide, we define a simple profile (Figs 5 and 6) to represent some concepts used in the field of urban planning. This profile diagram is a simplified representation of urban area composition used solely to highlight the capabilities of the language to represent both abstract and concrete concepts. On the one hand, this profile models the basic components of an area that make up a country in terms of buildings (housing, hospitals, theatres, transportation, solar panels, etc.), void (parks, places, etc.) as well as sanitation networks and roadways. For each territory, we can capture information such as population, elevation and location in the form of properties added to an area to evaluate the possible developments paths.

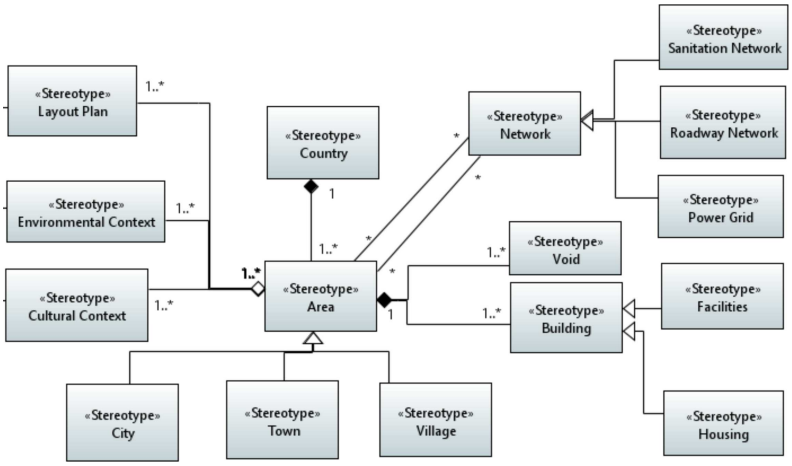

Fig. 5. Urban planning composition profile diagram.

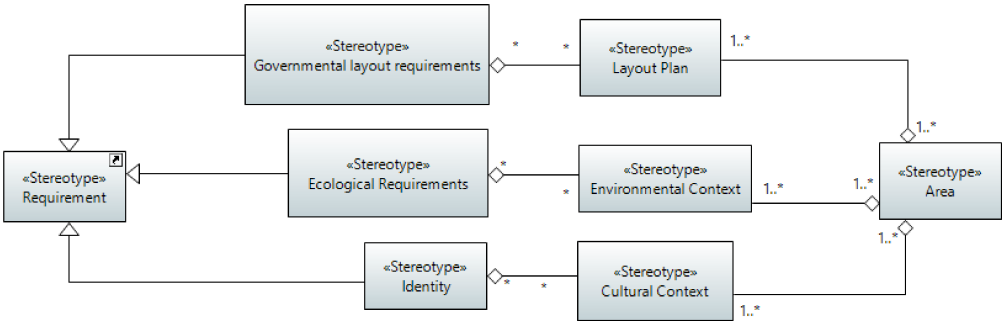

Fig. 6. Urban planning requirements profile diagram.

Similarly, each component of the area can be supplemented with properties depending on the needs. On the other hand, each area possesses its own cultural context defined by culture and customs that dictate the identity of the area, which urban planners must preserve in their plan. These factors incorporate how the population used the space over the years, therefore creating an identity that must be considered in any alteration in the urban space. The environmental context, determined by climate, geography (tropical weather, seismic activity), dictates ecological and sustainability requirements for the area. For our example, this is reflected as preservation and reclamation of land because it is scarce. Lastly, the example includes legal requirements, found in the local layout plan, that dictate proper sanitation, sufficient facilities for citizens and height limitations. In Fig. 5, all of the modeled concepts are an extension of the basic composition model element "Block". Once the composition concepts were defined, we added the "Requirement" model elements they must satisfy (Fig. 6). This profile diagram is only an example and should be supplemented with more composition-related model elements or other profiles for behavioral diagrams to encompass the urban planning process's needs fully.

Next, the Papyrus project that contains the profile diagram should be converted into a plugin to become a registered profile. This can be done by right-clicking the profile project in the project explorer view, choosing "configure" in the drop-down menu, and then converting to plugin project. Next, still in the project explorer view, open the "MANIFEST.MF" in the "META-INF" file of the project in order to configure the profile as a recognizable extension point to any other project. This can be done through the Extensions Points view by clicking the "add" button and inputting "org.eclipse.papyrus.uml.extensionpoints.UML. Profile" in the Extension Point ID and Name text zones (Fig. 7a). Then, in the Extension view, we add the extension point created to create the new extension (Fig. 7b), either to be added from the workspace (Fig. 8) or the registered profiles since it was transformed into an extension.

a) b)

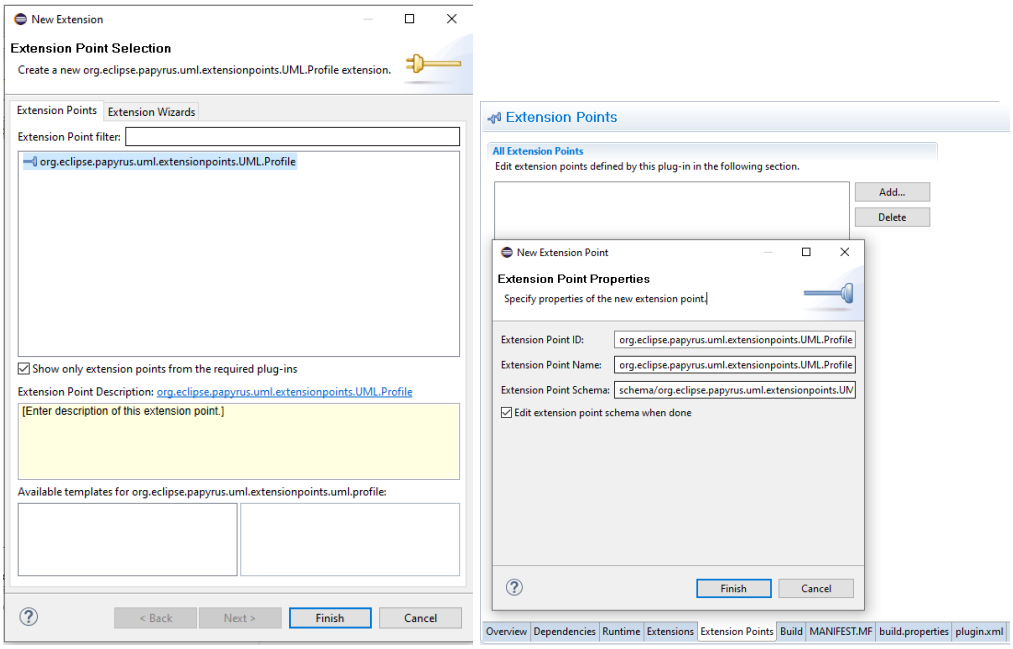

FIG. 7. a) Manifest.mf File Extension Points view, b) add an extension via the extension point.

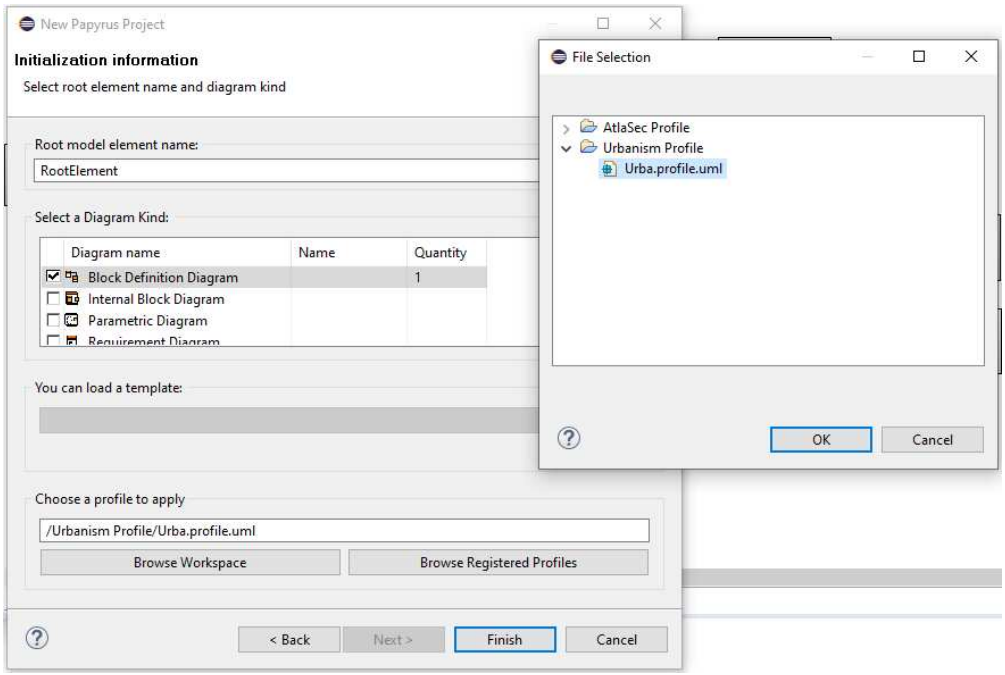

FIG. 8. Applying the profile to the requirement diagram.

4.2. Phase two: modeler configuration

Now, the new modeling project using the SysML profile can be created. In the project creation wizard, choose to create a new SysML diagram and name the new project. During the wizard's next step , add the defined profile and apply it to the requirement diagram. This is because the profile we created extends the semantics of the block diagram. The profile can either be added from the workspace (Fig. 8) or the registered profiles since it was transformed into an extension.

As the profile is properly added to the project, the palette needs to be configured to show the new stereotypes. First, right-click anywhere in the palette and choose "Customize" in the drop-down menu. Then, click the "Plus" button to start the wizard for the creation of a new palette (Fig. 9a). Under "Available Tools", the user can find the profile they created and add the stereotypes to the "Palette Preview". Each stereotype selected has a set of information to be configured, such as name, description and an icon for its display in the palette. Under "Aspect Actions", the appearance and attributes of the different stereotypes can be customized as well. The resulting palette is shown in (Fig. 9b).

a) b)

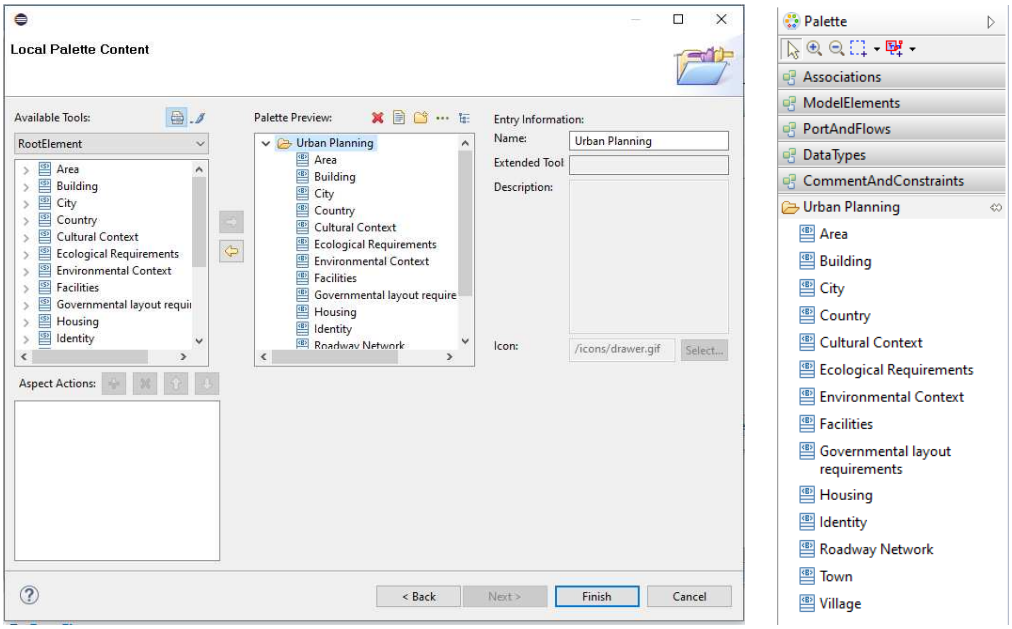

Fig. 9. a) Palette configuration interface, b) resulting palette.

The example chosen to highlight modeling urban planning components is the country of the Netherlands. One-third of the country is below sea level hav-

ing to deal constantly with flooding from the sea. With the advent of climate change, we stand to learn how this country deals with rising sea levels while also reclaiming drowned land for agriculture. In fact, 17% of the country was reclaimed from the sea [35]. This context and implemented solutions can be modeled, as shown in Fig. 10. First, the requirements are defined and decomposed into sub-requirements that can be satisfied with a model element. Then, we add the solutions that satisfy those requirements. The Beemster Polder [36] is a land reclaimed by building a network of windmill-powered waterwheels to extract water from the land to canals and evacuate it. Then, moats and dikes were added to secure the plots of land reclaimed from future flooding threats. Lastly, bridges and roads were built to connect all of these sections. Additional information can be added to each element in the model to allow testing. For example, each facility's functional capacities and limitations can be used in contrast with each region's geological and climate characteristics, either for testing or to guide future design. Moreover, further requirements can be added to abide by the regional or national urban plan in the future.

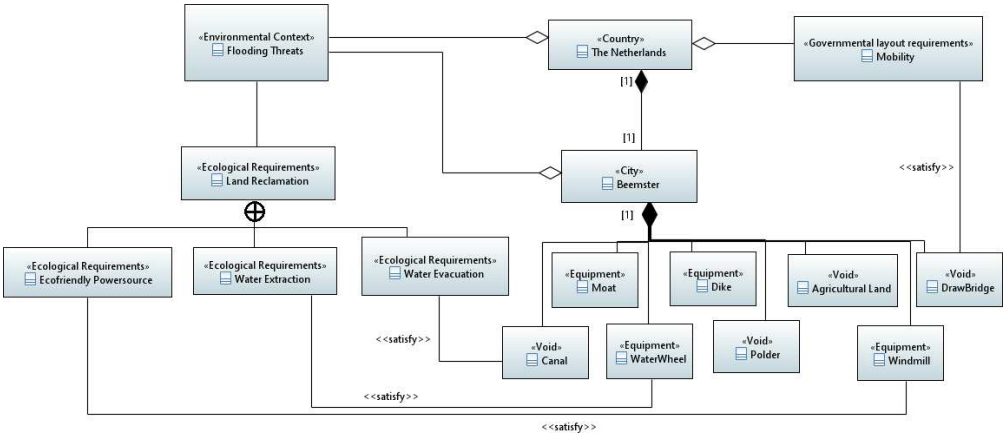

FIG. 10. Instances of the urban planning stereotypes.

5. Conclusion

In this article, we showcased the importance of graphical editors for DSLs in the improvement of designs in systems engineering. In fact, the modus operandi in MBSE is to address complexity through formal modeling languages. This results in a design that abides by the defined restrictions such as engine performance requirements or urban architecture plans. Moreover, limitations and constraints of any design element can be modeled through the parametric or requirement diagrams to make sure said design is kept in check. This results in higher quality designs without increased costs and facilitates risk and defect

analysis. This has been proven through the success of modeling languages such as UML and SysML. Since these two standards support the creation of new DSLs through extension, it allows niche subcategories in systems engineering to create their customized modeling languages. However, these profiles do not have access to the rich platforms and tools that support MBSE since they do not have dedicated modelers. This paper enables easy access to graphical editors for SysML-based DSLs by adapting an existing modeler to the defined profiles. We also illustrate the necessary steps to create the modeler.

Firstly, we presented an overview of the SysML language and the diagrams it proposes. Such an approach can model behavior, composition and their requirements. Thus, we can extend these diagrams to create a new language that also models the same aspects but uses the domain's concepts. Secondly, we showed how this new language could obtain its own modeling tool. There are many ways of creating this modeler, but we opted for extending an existing rich modeling platform for ease of use and benefit from its functionalities. We determined that the adequate environment must be extensible to allow the addition of the new stereotypes. The environment must also be modular to provide access to existing plugins and tools and facilitate the addition of new ones. It must also provide the necessary support for non-experienced users and have difficult programming hurdles. Using the formal AHP method, we identified that Eclipse Papyrus is the best choice on all fronts. This procedure can be redone for different modeling tools in the future using different criteria. Then, we presented the guide that allows the adjustment of Eclipse Papyrus into a graphical modeler for any SysML profile. However, this guide only provides a prototype for a modeler for single-use purposes or experimentation. We provided a guide for the adaptation of the modeler illustrated by an urban planning example. Areas of improvement include the creation of a plugin for DSL modeling based on graphical SysML profiles hosted on a newer and more stable IDE. This would allow the development of more DSLs and amplify progress within MBSE.

References

1. NDIA Systems Engineering Division, Final Report of the Model Based Engineering (MBE) Subcommittee, February, pp. 60, 2011, http://www.ndia.org/-/media/sites/ndia/meetings-and-events/divisions/systems-engineering/modeling-and-simulation/reports/model-based-engineering.ashx.

2. P. Micouin, *Model-Based Systems Engineering*, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2014.

3. M. Fowler, R. Parsons, *Domain Specific Languages*, Addison-Wesley Professional, 2010.

4. N. Shevchenko, *An Introduction to Model-Based Systems Engineering (MBSE)*, 2021, https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/.

5. ISO, Unified modeling language specification version 1.4.2, *ISO/IEC 19501:2005(E)*, vol. 4, no. 1, pp. 25–59, 2005, https://www.iso.org/standard/32620.html.

6. O. Casse, *SysML Action with Cameo System Modeler*, Chapter 1: SysML: Object Management Group (OMG) Systems Modeling Language, pp. 1–63, ISTE Press Ltd and Elsevier Ltd, 2017, doi: 10.1016/B978-1-78548-171-0.50001-3.

7. Autodesk, ECAD and MCAD software, https://www.autodesk.com/solutions/ecad-and-mcad-software.

8. MEGA, Hopex Platform, https://www.mega.com/en/hopex-platform.

9. SparxSystems, Full Lifecycle Modeling for business, software and systems, https://sparxsystems.com/products/ea/index.html.

10. Eclipse, Sirius, https://www.eclipse.org/sirius/.

11. Eclipse, Papyrus, https://www.eclipse.org/papyrus/.

12. O. Badreddin, K. Rahad, The impact of design and UML dodeling on codebase quality and sustainability, [in:] *CASCON '18: Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, October 2018, pp. 236–244, 2018.

13. A. Nugroho, M.R.V. Chaudron, Evaluating the impact of UML modeling on software quality: an industrial case study, [in:] A. Schürr, B. Selic [Eds], *Model Driven Engineering Languages and Systems. MODELS 2009.* Lecture Notes in Computer Science, vol 5795, pp. 181–195, Springer, Berlin, Heidelberg, 2009, doi: 10.1007/978-3-642-04425-0_14.

14. Object Management Group, D. Number, M.C. Files, Object Constraint Language, February, 2014, http://www.omg.org/spec/OCL/2.4.

15. Sun Microsystems, JavaDoc – The Java API documentation Generator, 2011, https://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/javadoc.html.

16. Sphinx Python documentation generator overview, https://www.sphinx-doc.org/en/master/.

17. D.R. Price, Concurrent Versions System – Overview, 2012, https://savannah.nongnu.org/projects/cvs.

18. L. Torvalds, Git – About, https://git-scm.com/about.

19. Eclipse Foundation, Graphical Modeling Framework/Tutorial/Part 1, http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1.

20. Eclipse, Acceleo – Home, https://www.eclipse.org/acceleo/.

21. Eclipse, Xtext – Language Engineering Made Easy, https://www.eclipse.org/Xtext/.

22. D.S. Kolovos, A. Garcıa-Domınguez, L.M. Rose, R.F. Paige, Eugenia: towards disciplined and automated development of GMF-based graphical model editors, *Software & Systems Modeling*, **16**: 229–255, 2015, doi: 10.1007/s10270-015-0455-3.

23. A. Zolotas, R. Wei, S. Gerasimou, H. Hoyos Rodriguez, D.S. Kolovos, R.F. Paige, Towards automatic generation of UML profile graphical editors for Papyrus, [in:] A. Pierantonio, S. Trujillo [Eds], *Modelling Foundations and Applications. ECMFA 2018*, Lecture Notes in Computer Science, vol. 10890, Springer, Cham, 2018, doi: 10.1007/978-3-319-92997-2_2.

24. N. Taylor, *Urban Planning Theory Since 1945*, Sage Publications, London, Thousand Oaks, New Delhi, 1998.

25. C. Loisel, F. Le Roux, *Yona Friedman: Architecture mobile =Architecture vivante*, Press Release, Cité de l'Architecture & du Patrimoine, Paris, France, 11 May 2016.

26. A. Petruccioli, *After Amnesia: Learning from the Islamic Mediterranean Urban Fabric*, ICAR, University of Virginia, 2007.

27. Le Corbusier, *The Athens Charter*, Grossman Publishers, New York, 1973.

28. R. Berardi, The spatial organization of Tunis Medina and other Arab-Muslim cities in North Africa and the Near East, [in:] *The City in the Islamic World (2 vols.)*, pp. 269–293, 2008, doi: 10.1163/ej.9789004162402.i-1500.70.

29. Object Management Group, UML for Systems Engineering Request for Proposal, pp. 49–68, https://sysml.org/.res/docs/refs/UML-for-SE-RFP.pdf.

30. Object Management Group, OMG Meta Object Facility (MOF) Core Specification, pp. 76, 2003, https://www.omg.org/spec/MOF/2.5.1/PDF.

31. T. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, McGraw-Hill, 1980.

32. Catia No Magic – Dassault Systèmes, MagicDraw, https://www.nomagic.com/products/magicdraw.

33. IBM, IBM Engineering Systems Design Rhapsody, https://www.ibm.com/products/systems-design-rhapsody.

34. Eclipse, Eclipse IDE Mars 2 Packages, https://www.eclipse.org/downloads/packages/release/mars/2/eclipse-modeling-tools.

35. E.F.J. De Mulder, B.C. De Pater, J.C. Droogleever Fortuijn, *The Netherlands and the Dutch: A Physical and Human Geography*, Springer International Publishing, 2018.

36. *Droogmakerij de Beemster (Beemster Polder)*, UNESCO World Heritage Center, https://whc.unesco.org/en/list/899/.

Received May 29, 2021; revised version December 19, 2021.