AUTOMATED DEVELOPMENT OF WEB-BASED MODELING SERVICES FOR MSAAS PLATFORMS

Paolo Bocciarelli
Andrea D'Ambrogio
Antonio Mastromattei
University of Rome Tor Vergata
Via del Politecnico, 1
Rome, Italy
dambro@uniroma2.it

Andrea Giglio
Guglielmo Marconi University
Via Plinio 44
Rome, Italy

ABSTRACT

MSaaS (*M&S as a Service*) is gaining momentum as an effective approach to bring the benefits of service-oriented architectures and cloud computing into the M&S field, so as to enhance interoperability, composability, reusability and reduce the cost of M&S efforts. Such significant advantages can be further enhanced by introducing automated model transformations that support the various phases of a M&S effort, from simulation model building down to model implementation, deployment and execution. In previous contributions we have already addressed the use of automated model transformations that can be effectively adopted to provide *simulation services* for MSaaS platforms. This paper instead focuses on the automated development of *modeling services* for MSaaS, i.e., those services that allow platform users to easily build models in their own modeling language by use of a web-based user interface. Specifically, this work proposes an approach to automatically generate web-based visual editors from a metamodel that defines a given modeling language. Once generated, such editors can be made available on demand through a complete MSaaS platform, which also includes simulation services. The paper first describes the architecture of a MSaaS platform that includes modeling services, then illustrates the method for the automated development of web-based modeling services and, finally, gives a complete example application of the proposed method.

Keywords: web-based modeling, MSaaS, model transformation, model-driven development, visual editors.

1 INTRODUCTION

Modeling & Simulation (M&S) approaches are widely recognized as effective and valuable solutions to carry out system analysis activities at both design and operation time. The use of simulation models allows decision-makers and system designers to get the appropriate system understanding, according to the desired or required level of abstraction (Gianni et al. 2014, Sokolowski and Banks 2009).

The well known benefits of M&S approaches can be further enhanced by exploiting cloud-based infrastructures, in order to save the investments for developing and maintaining expensive platforms for building, deploying and executing simulation models. In this respect, *MSaaS* (*M&S as a Service*) is gaining momentum as an effective approach to bring the benefits of service-oriented architectures and cloud computing into the M&S field, so as to enhance interoperability, composability, reusability and reduce the costs of M&S efforts. MSaaS provides a system-level architecture for discovery, orchestration, deployment, delivery and management of M&S services, which can be defined as specific M&S-related capabilities

SpringSim-Mod4Sim 2017, April 23-26, Virginia Beach, VA, USA

delivered on demand by a provider to one or more consumers according to well defined contracts (Tsai et al. 2011, Siegfried et al. 2014).

The development of modeling and simulation services for MSaaS platforms may greatly benefit from the use of *model-driven development* (*MDD*), which enhances the typical advantages of model-based development (e.g., improved quality, enhanced communication and stakeholder engagement, increased productivity, enhanced knowledge transfer, and reduced risks) by considering models as first-class artifacts and increasing the level of automation by use of model transformations. The use of such approaches enables a radical shift in terms of modeling activities, from a strictly contemplative use of models to a more productive and powerful model use, and contributes to increase the level of automation at development time. Example uses of MDD approaches in the M&S field can be found in (Bocciarelli and D'Ambrogio 2014, Topçu et al. 2016).

The combined use of MDD and MSaaS allows M&S users to reduce both the effort of simulation model building and implementation, and the costs associated with establishing and maintaining a dedicated execution infrastructure, as highlighted in (Bocciarelli et al. 2013).

In this respect, this paper focuses on the automated (i.e., model-driven) development of *modeling services* for MSaaS, i.e., those services that allow platform users to easily build models in their own modeling language by use of a web-based visual interface. Specifically, this work proposes an approach to automatically generate web-based visual editors from a metamodel that defines a given modeling language. Once generated, such editors can be made available on demand through a complete MSaaS platform, which also includes *simulation services*. As an example, this paper addresses the *SOASim* platform, which provides simulation services for the performance analysis of systems and processes according to the MSaaS paradigm (D'Ambrogio et al. 2016).

To the best of our knowledge, no contributions can be found in literature which specifically address the automated development of modeling services. Some Eclipse projects address the issue of supporting the development of modeling tools, such as Eugenia (Eclipse Foundation 2016b) or RAP (Eclipse Foundation 2016c). Eugenia is a valuable project which enables the automated generation of editor environments, starting form domain metamodels. Unfortunately, the obtained editors cannot be used in web environment, as they are only available as plugins suitable in Rich-Client Platforms (RCPs) such as Eclipse. RAP is a promising project which aims to use existing RCP plugins on web, but currently it does not support the Eclipse Graphic Modeling Framework (GMF), which is at the basis of any Eclipse-based editor environment.

This paper instead introduces an automated approach to obtain a web-based visual editor for building models that conform to a metamodel defining the modeling language for a given application domain.

The reminder of the paper is structured as follows: Section 2 provides an overview of SOASim, the reference MSaaS platform. Section 3 illustrates the proposed method for generating the web-based visual editor and provides a detailed view of the underlying transformations. Section 4 shows the application of the method to an example case study and finally Section 5 gives the final remarks.

2 OVERVIEW OF THE SOASIM ARCHITECTURE

This section outlines the SOASim architecture, which has been selected as the reference MSaaS platform for deploying the web-based modeling services automatically generated according to the model-driven approach proposed in this paper.

SOASim is a model-driven framework that aims to support and ease the performance-oriented and simulation-based analysis of systems and processes. A first version of SOASim has been proposed in (D'Ambrogio et al. 2016). This paper contribution enhances the original SOASim architecture in order

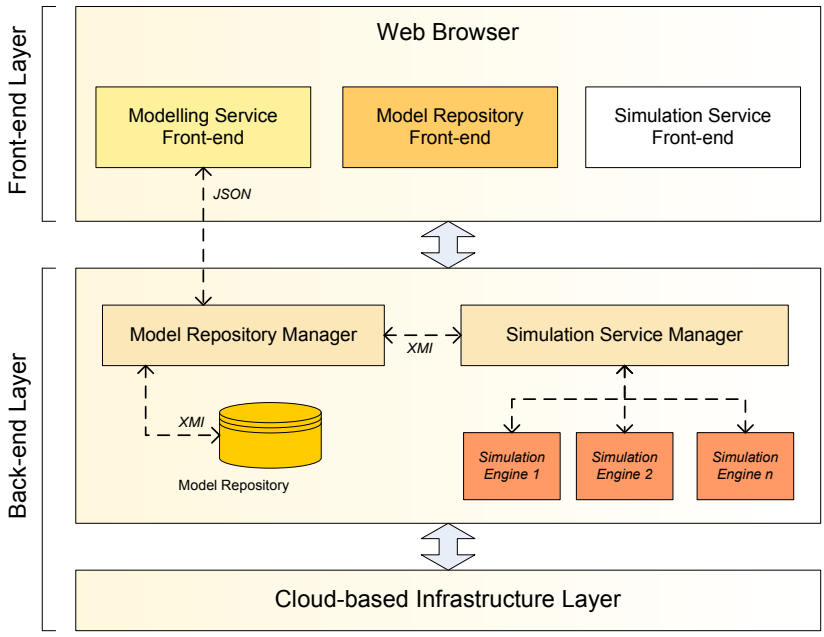

Figure 1: SOASim reference architecture.

to make it compliant to innovative web-based modeling services. The novel architecture is shown in Figure 1. Next subsections provide a description of both the front-end layer and the back-end layer, respectively.

2.1 Front-end Layer

The *front-end layer* provides a web-based interface that allows users to interact with the SOASim framework. This layer is made up of the following components:

- **Modeling Service Front-end**: modeling services are provided as a web-based component that allows platform users to create, edit and store models. According to the main idea at the basis of this contribution, in order to make SOASim flexible enough to fit the needs of several application domains and modeling languages, the web modeling service front-end can be automatically generated from the metamodel that defines the modeling language relevant to a specific domain. Section 3 provides a detailed description of the modeling service design and implementation, while Section 4 illustrates the application to an example modeling language.
- **Model Repository Front-end**: this component provides the web-based visual interface to manage the repository of models created by use of the modeling service front-end component.
- **Simulation Service Front-end**: this component allows platform users to manage the several operations required to carry out a simulation-based analysis of a system, which is accomplished through the interaction with the Simulation Service Manager of the back-end layer.

2.2 Back-end Layer

The *back-end layer* consists of the server components implementing the logic needed to manage the modeling and simulation activities. This layer is made up of the following components:

- **Model Repository Manager**: component that provides the services for storing and retrieving models to/from the repository. It also provides a service for converting the JSON (JavaScript Object Notation) representation of models produced by the modeling service front-end (see Section 3) to the corresponding XML format.
- **Model Repository**: component which contains the XML representation of both models specified by the modeling service and models generated by the simulation services throughout a set of model-to-model transformations.
- **Simulation Service Manager**: component in charge of managing the required server-side tasks needed to carry out a simulation-based analysis. Specifically, it provides the following activities:
  - execution of the required model-to-model transformation chain for generating a simulation-based performance model from a system model;
  - management of the simulation model partitioning, in case a distributed simulation execution platform is used;
  - execution of the required model-to-text transformations to generate the code and the required configuration files, needed to execute the simulation model onto the simulation engine;
  - deployment of the simulation components onto the target local or distributed simulation platform.
- **Simulation Engines**: the actual simulation engines that execute the relevant simulation models. Currently SOASim provides the two following engines: jEQN, for executing performance-based simulation models of extended queueing network type (D'Ambrogio et al. 2006), and eBPMN, for executing performance-based business process simulation models (Bocciarelli et al. 2014).
- **Cloud-based Infrastructure Layer**: this layer deals with the interaction with the cloud-based infrastructure used for deploying and executing the local or distributed simulation. Currently SOAsim exploits the PaaS (Platform as a Service) paradigm and adopts the Docker open source project (Docker 2016) to manage the deployment of simulation components over the related virtual containers.

3 AUTOMATED GENERATION OF WEB-BASED MODELING SERVICES

3.1 Overview

The main idea of this contribution is to automate the generation of web-based modeling services from the abstract specifications of the modeling language of the domain under study. In this respect, the generation process involves several models, metamodels and model transformations, as summarized in Figure 2 and hereby explained.

The *generator metamodel*, described in Section 3.2, is a MOF-compliant metamodel (OMG 2015), which describes the elements of the web-based visual editor, e.g., stencils, containers, links, etc.

The *domain metamodel* is a MOF-compliant metamodel describing the modeling language of the domain under study (e.g, the BPMN metamodel, the UML metamodel, the DEVS metamodel or a custom metamodel). An example custom metamodel is introduced in Section 4.

The *interchange metamodel*, described in Section 3.3, is a MOF-compliant metamodel used to drive the serialization of models, so as to be compliant with the Model Repository Manager (see Figure 1) and with third-party tools, as well.

The *generator model* is at the core of the proposed approach. It is an instance of the generator metamodel and includes a reference to the domain model. The generator model allows one to specify which metamodel elements have to be considered in the web-based visual interface, as well as the related role. In other words

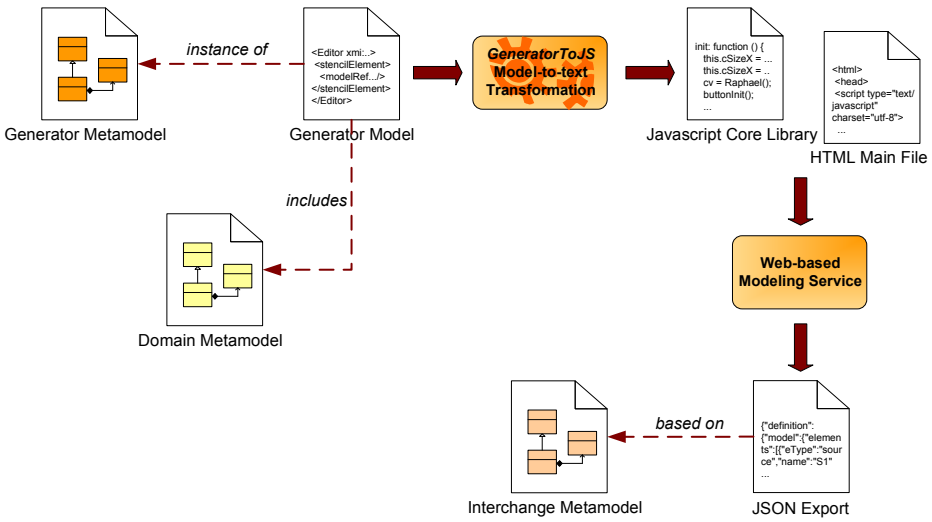

Figure 2: Model-driven process for the automated generation of web-based modeling services.

it specifies which elements of the domain model have to be included in the editor stencil, what are the related icon images, what is the appearance of links, and similar details. An example of generator model is provided in the case study described in Section 4.

The *GeneratorToJS* model-to-text transformation takes as input the generator model and produces as output the Javascript core library implementing the web editor business logic and the main HTML page that activates the editor. The library has to be deployed onto the server which executes the Modeling Service Front-end, along with the HTML main page and the required icon images. Such a model transformation is further described in Section 3.4.

3.2 Generator Metamodel

The *generator metamodel* is illustrated in Figure 3. It describes from an abstract point of view the characteristics of a visual editor and specifies the relevant elements that are to be made available in the stencil. Specifically, three different stencil elements have been identified:

- **Partition**: a container of other elements, e.g., a swimlane in a UML activity diagram or a pool in a BPMN diagram. The `Partition` metaclass is described in terms of visual dimension (i.e., `width` and `height`) and whether or not it is horizontally o vertically resizable (i.e., `resizableWidth` and `resizableHeight`, respectively).
- **Link**: a connection between two elements. The `Link` metaclass is described in terms of attributes that specify the link appearance (e.g., style and color).
- **Shape**: a visual element in the editor stencil. The `Shape` metaclass is described in terms of the `iconPath` attribute, which stores the path of the production server where the shape icon image is located, and the `incomingEdge` and `outgoingEdge` attributes, which are provided to specify the link that represents incoming and outgoing edges, respectively.

Finally, in order to realize the association between the generator model and the domain metamodel, the abstract superclass `StencilElement` provides the attribute `modelRef`, which links each stencil element of the generator model to the corresponding metaclass of the domain metamodel.

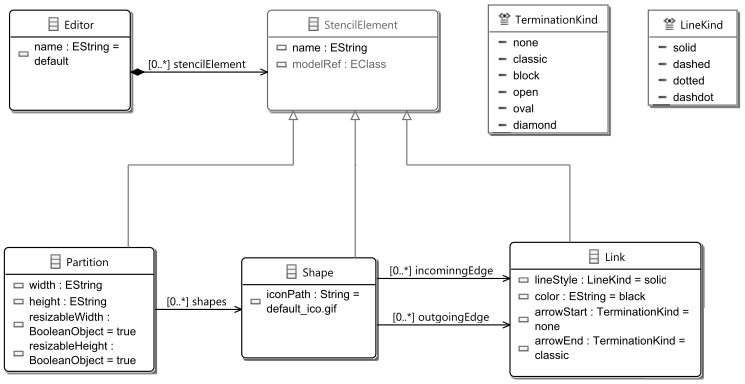

Figure 3: Editor Generator Metamodel.

The *generator metamodel* has been implemented by use of EcoreTools, the Ecore diagram editor part of the Eclipse Modeling Project. The same editor has been used for instantiating the *generator model* from its metamodel.

3.3 Interchange Metamodel

The ability of serializing and exporting a model in a structured textual format is an essential feature provided by most modeling tools. In this respect, the web-based visual editor provides a JSON-based export feature. In order to be compliant with the SOASim platform, which includes a repository to store models in XML format (see Figure 1), the JSON export feature has been based on the MOF-based *interchange metamodel* depicted in Figure 4, so as to fully specify the interchange format and drive the implementation of the Repository Manager module that maps the JSON document to the equivalent XML model.

The interchange metamodel root is the `definition` metaclass, which contains the two following metaclasses:

- Model: the semantic description of the model in terms of its elements and links. Such items are described by the `Element` metaclass and the `Link` metaclass, respectively. In order to represent the model topology, the `Element` metaclass includes the attributes `outgoingRef` and `incomingRef`, which reference `Link` elements. Similarly, the `Link` metaclass provides the attributes `source` and `target`.
- Diagram: the visual description of the model, in terms of the visual characterization of edges and shapes.

It has to be noted that the interchange metamodel should be considered as a *minimal* requirement that has to be satisfied in order to ensure compliance with SOASim. Specifically, as the visual editor is dynamically generated from a generic metamodel, its internal representation of a model consists of a data structure based on the interchange metamodel, which is also extended with domain-related attributes specified in the actual domain metamodel.

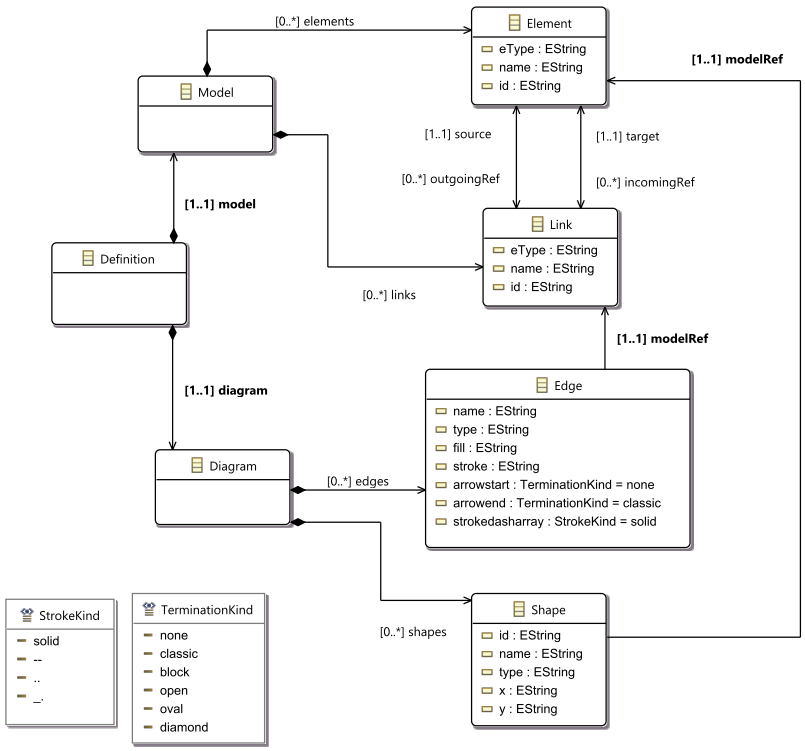

Figure 4: Semantic and Visual Interchange Metamodel.

3.4 Model-to-Text Transformation

The `GeneratorToJS` *model-to-text* transformation is the core of the model-driven process described in Figure 2. It takes as input the generator model specifying the visual editor characteristics and yields as output both the *Javascript* core library and the HTML main file, which provides the actual implementation of the editor for a given modeling language (as defined by the relevant domain metamodel). The proposed transformation is based on OMG's MDA standards and technologies and its implementation exploits the Eclipse Luna IDE and the Eclipse Modeling Framework (EMF). Specifically, the transformation has been specified by use of the OMG MOF model-to-text transformation language (MOFM2T) (OMG 2008) and implemented by use of Acceleo plugin (Eclipse Foundation 2015). The metamodel specification and implementation has been based on the Ecore Tools project (Eclipse Foundation 2016a), which provides a set of tools to create, manage and serialize metamodels. Such a project aslo include Ecore, the Eclipse reference implementation of OMG's Essential-MOF (EMOF) (OMG 2015).

The Javascript core library that implements the web-based editor has been designed according to the *model-view-control (MVC)* pattern. In this respect, it has been structured in a single file containing the various function objects that provide the implementation of model, view and control elements. According to the structure of such a library, the `GeneratorToJS` transformation provides the following modules, as reported in Listing 1:

- *generateHTML*, for generating the HTML main file;
- *generateControllerClass*, for generating elements of the *control* layer;
- *generateShapeModel*, for generating model objects related to shapes stencil elements;
- *generateLinkModel*, for generating model objects related to links elements;

- *generateViewClass*, for generating elements of the *view* layer;
- *generate*, for managing the overall transformation.

```
[module generate('http://www.eclipse.org/emf/2002/Ecore', 'http://www.example.org/
    WebEditor']
[import ... /]

[template public generateElement(editor : Editor) {LinkList : Sequence(Link) = editor.
    stencilElement->filter(Link)->asSequence(); } ]
[comment @main/]

[comment generation of HTML file /]
[file (editor.name.concat('.HTML'), false, 'UTF-8')]
[editor.generateHTML()/]
[/file]

[file (editor.name.trim().concat('.js'), false, 'UTF-8')]

[comment generation of Controller class /]
[editor.generateControllerClass()/]

[comment generation of Model classes for Shape and Link elements /]
[for (s : Shape | editor.stencilElement->filter(Shape))]
[s.generateShapeModel()/][/for]
[for (l : Link | editor.stencilElement->filter(Link))]
[l.generateLinkModel()/][/for]

[comment generation of View class /]
[editor.generateViewClass()/]
[/file]
[/template]
```

Listing 1: *GeneratorToJS* main module.

4 EXAMPLE APPLICATION

This section describes the application of the proposed model-driven process to the automated generation of a visual interface for a modeling language specified by use of a custom domain metamodel.

4.1 Domain Overview

The example application specifically deals with the development of a visual interface to create, edit and serialize simple business process models. It is assumed BPMN is far too complex for such an application and thus a simpler modeling language is introduced and specified by a custom metamodel, namely the *SimpleBP* metamodel, which is shown in Figure 5.

A *SimpleBP* model, created by instantiating elements of the *SimpleBP* metamodel, is a collection of `node` elements connected by `edge` elements. Among the several existing BPMN `node` elements, the following subset of flow objects has been used as part of *SimpleBP* notation: `StartNode`, `EndNode`, `Task` and `Gateway`. In addition, two different kinds of edges are included: `SequenceFlow` elements, with the `condition` attribute specifying the condition under which a control token is routed to the edge, and `MessageFlow` elements, to represent messages exchanged between nodes. It is assumed that a `StartNode` element must not have any incoming sequence flow and, conversely, an `EndNode` element

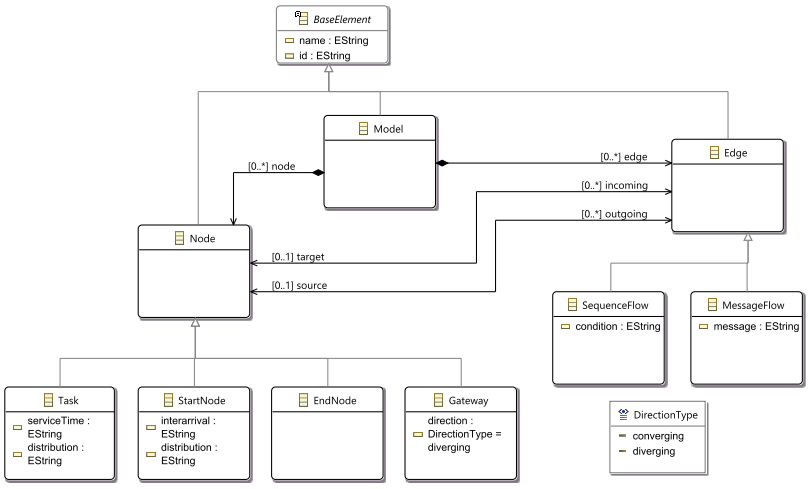

Figure 5: SimpleBP Metamodel.

must not have any outgoing sequence flow. Finally, message flow edges are allowed only between tasks. Such constraints are not specified in the *SimpleBP* metamodel, but could be easily added by use of OCL (Object Constraint Language) statements.

4.2 Generator Model

The second step of the proposed method consists in the definition of the generator model, which is created from the generator metamodel and used to specify which elements of the SimpleBP metamodel have to be included in the editor stencil.

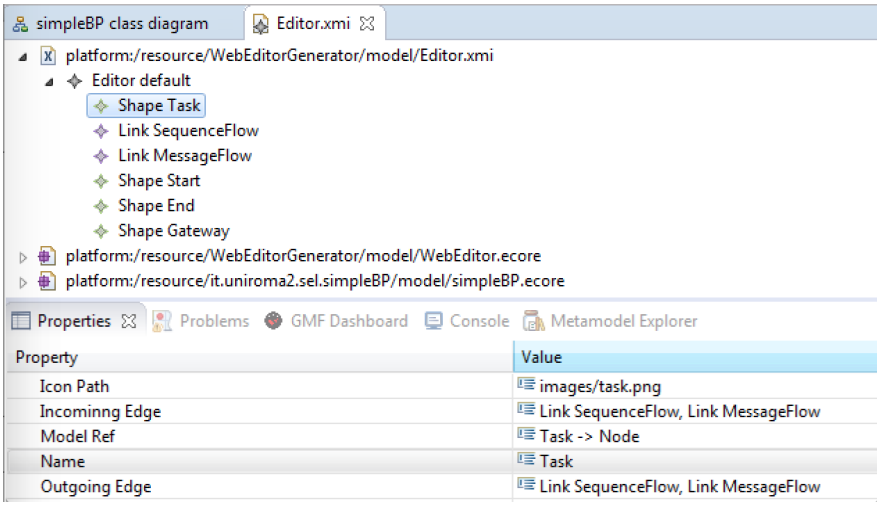

Figure 6: SimpleBP Generator Model.

Figure 6 shows a screen snapshot of the Eclipse EMF editor used to specify the model, while Listing 2 provides the related XML serialization. The generator model identifies four `Shape` elements, corresponding to the four subclasses of the *SimpleBP* `Node` metaclass, and specifies `SequenceFlow` and `MessageFlow` elements as instances of the `Link` metaclass. Finally, the generator model includes the necessary information to complete the visual and the semantic specification of the *SimpleBP* editor, i.e.,

the appropriate references to *SimpleBP* metaclasses associated to stencil elements, appearance information, constraints on allowed edges and icon paths. As an example, the bottom part of Figure 6 gives the values of `Task` attributes in terms of reference metaclass (`Task -> Node`), icon filename path (`images/task.png`) and allowed incoming and outgoing edges (`Link SequenceFlow` and `Link MessageFlow` for both).

```
<?xml version="1.0" encoding="ASCII"?>
<WebEditor:Editor xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xmlns:WebEditor="http://www.example.org/
    WebEditor"
    xsi:schemaLocation="http://www.example.org/WebEditor WebEditor.ecore" name="
        SimpleBP Visual Editor">
  <stencilElement xsi:type="WebEditor:Shape" name="Task" iconPath="images/task.png"
      incominngEdge="//@stencilElement.1 //@stencilElement.2" outgoingEdge="//
      @stencilElement.1//@stencilElement.2">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//Task"/>
  </stencilElement>
  <stencilElement xsi:type="WebEditor:Link" name="SequenceFlow">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//SequenceFlow
        "/>
  </stencilElement>
  <stencilElement xsi:type="WebEditor:Link" name="MessageFlow" lineStyle="dashed">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//MessageFlow"
        />
  </stencilElement>
  <stencilElement xsi:type="WebEditor:Shape" name="Start" iconPath="images/start.png"
      outgoingEdge="//@stencilElement.1">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//StartNode"/>
  </stencilElement>
  <stencilElement xsi:type="WebEditor:Shape" name="End" iconPath="images/end.png"
      incominngEdge="//@stencilElement.1">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//EndNode"/>
  </stencilElement>
  <stencilElement xsi:type="WebEditor:Shape" name="Gateway" iconPath="images/gateway.
      png"incominngEdge="//@stencilElement.1" outgoingEdge="//@stencilElement.1">
    <modelRef href="../../it.uniroma2.sel.simpleBP/model/simpleBP.ecore#//Gateway"/>
  </stencilElement>
</WebEditor:Editor>
```

Listing 2: SimpleBP Generator Model.

The generator model is then given as input to the `GeneratorToJS` model transformation, which yields as output the required Javascript and HTML code implementing the SimpleBP web editor.

4.3 Editor execution

Figure 7 shows a screen snapshot of the obtained SimpleBP visual editor. The example SimpleBP model created by use of the visual editor includes a node, a gateway, three tasks and an end node. According to the generator model, sequence flows are shown as solid links while message flows are depicted as dotted lines. A message is exchanged between the *SelectProduct* task and the *PrintInvoice* task. The figure also shows the dialog (which can be activated by a double click on the selected shape or line) used to set the properties of the *ShipProduct* task. The parameterized model can then be simulated by use of *simulation services* provided by the SOASim framework, as described in (D'Ambrogio, Bocciarelli, and Mastromattei

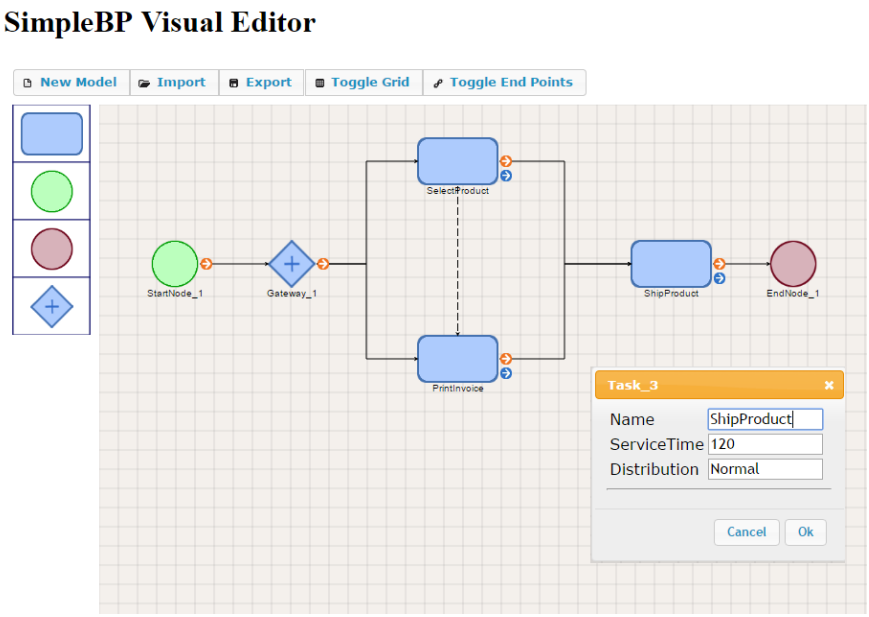

Figure 7: Visual interface of the SimpleBP editor.

2016). Finally, the upper part of the window includes the buttons area which provides features to create a new model, to import/export a model and to adjust the diagram appearance.

5 CONCLUSIONS

The paper has illustrated a model-driven automated approach to the development of web-based visual editors. The proposed approach makes use of the SOASim MSaaS platform and exploits model-driven principles and tools to automate the development of a web-based visual interface for a modeling language specified by use of a domain metamodel. Specifically, the domain metamodel is taken as input by a *model-to-text* transformation along with a descriptor, named *generator model*, which specifies the visual properties of the editor and the role played by metamodel elements. The transformation yields as output the Javascript core library implementing the web-based visual editor and the HTML page that activates the editor. The paper has also presented an example *proof-of-concept* application, in order to show the feasibility and the effectiveness of the proposed approach. Ongoing work includes the extension of the proposed *model-to-text* transformation to metamodel elements and patterns which are not currently addressed (e.g., complex containment relationships, such as BPMN sub-processes or UML Activity Diagram sub-activities) and the implementation of an additional SOASim module to store and exchange models in JSON format.

REFERENCES

Bocciarelli, P., and A. D'Ambrogio. 2014. "Model-driven method to enable simulation-based analysis of complex systems". In *Modeling and Simulation-Based Systems Engineering Handbook.*, pp. 119–148.

Bocciarelli, P., A. D'Ambrogio, A. Giglio, and D. Gianni. 2013. "A SaaS-based automated framework to build and execute distributed simulations from SysML models". In *Proceedings of the 2013 Winter Simulation Conference*, WSC '13, pp. 1371–1382.

Bocciarelli, P., A. D'Ambrogio, and E. Paglia. 2014. "A language for enabling model-driven analysis of business processes". In *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, pp. 325–332.

D'Ambrogio, A., P. Bocciarelli, and A. Mastromattei. 2016. "A PaaS-based Framework for Automated Performance Analysis of Service-oriented Systems". In *Proceedings of the 2016 Winter Simulation Conference*, WSC '16, pp. 931–942.

D'Ambrogio, A., D. Gianni, and G. Iazeolla. 2006. "jEQN a Java-based Language for the Distributed Simulation of Queueing Networks". In *Proceedings of the 21st International Conference on Computer and Information Sciences*, ISCIS'06, pp. 854–865. Berlin, Heidelberg, Springer-Verlag.

Docker 2016. "Docker". Website: https://www.docker.com.

Eclipse Foundation 2015. "Acceleo". Website: https://eclipse.org/acceleo/.

Eclipse Foundation 2016a. "EcoreTools". Website: http://www.eclipse.org/ecoretools/.

Eclipse Foundation 2016b. "EuGENia". Website: http://www.eclipse.org/epsilon/doc/eugenia/.

Eclipse Foundation 2016c. "RAP - Remote Application Plaftorm". Website: http://www.eclipse.org/rap/.

Gianni, D., A. D'Ambrogio, and A. Tolk. (Eds.) 2014. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press.

OMG 2008. *MOF Model To Text Transformation Language, version 1.0*.

OMG 2015. *Meta Object Facility, version 2.5*.

Siegfried, R., T. van den Berg, A. Cramp, and H. W.. 2014. "M&S as a Service: Expectations and Challenges". In *In proceedings of the Fall Simulation Interoperability Workshop*.

Sokolowski, J., and C. Banks. 2009. *Principles of Modeling and Simulation*. Wiley.

Topçu, O., U. Durak, H. Oğuztüzün, and L. Yilmaz. 2016. *Distributed Simulation: A Model Driven Engineering Approach*. Springer.

Tsai, W.-T., W. Li, X. Bai, and J. Elston. 2011. "P4-SimSaaS: Policy specification for multi-tendency simulation Software-as-a-Service model". In *Proceedings of the 2011 Winter Simulation Conference (WSC'11)*, pp. 3067–3081.

AUTHOR BIOGRAPHIES

PAOLO BOCCIARELLI is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research addresses the application of M&S and model-driven development to software and systems engineering and business process management. His email address is paolo.bocciarelli@uniroma2.it.

ANDREA D'AMBROGIO is associate professor at the Dept. of Enterprise Engineering of the University of Roma TorVergata (Italy). His research interests are in the fields of model-driven systems and software engineering, distributed and web-based simulation. His email address is dambro@uniroma2.it.

ANDREA GIGLIO is assistant professor at the "Guglielmo Marconi" University. His research addresses model-driven approaches to business process modeling and analysis. His email address is a.giglio@unimarconi.it.

ANTONIO MASTROMATTEI is a PhD student at the University of Rome "Tor Vergata" (Italy), currently working on business process modeling and M&S solutions for MSaaS platforms. His email address is antonio.mastromattei@uniroma2.it.