Petri Net Simulation as a Service

Petr Polasek, Vladimir Janousek, and Milan Ceska

Faculty of Information Technology, BUT,
IT4Innovations Centre of Excellence,
Bozetechova 1/2, 612 66 Brno, Czech Republic
{polasek,janousek}@fit.vutbr.cz
http://www.fit.vutbr.cz

Abstract. *This paper presents an approach to integrating a Petri net simulator into a service-oriented simulation architecture in order to provide on demand simulation as a service. As a concrete example the simulation tool Renew is wrapped as a service and used to simulate a sample traffic control system represented as a Petri net model that is able to adjust its parameters via reflective simulation in a distributed simulation environment. The simulation architecture where modeling and simulation is treated as a service (MSaaS) is presented. The main components, their roles and attributes are briefly discussed.*

Keywords: simulation as a service, Petri nets, integration of tools, simulation architecture, web services, system design, simulation model, model specification, reflective simulation

1 Introduction, motivation and context

Modeling and Simulation (M&S) plays an unsubstitutable role in successful design of systems. A need for timely, cost-effective and resource-effective development requires to have a model of a system that can be analyzed, simulated, verified and even validated. As one may expect, modeling and simulation of complex and heterogeneous systems requires the application of more than one approach. Although developers have dozens of tools available that support various formalisms and allow them to use different M&S practices and methodologies, their integration into a simulation environment is rather a challenging task. Proper interaction and communication is a non-trivial task. Enabling access to functionality of an integrated application to a group of people is a challenge. The requirement for scalability and on demand computational power in simulation environment may represent another complication.

As a small and motivated team with limited resources, we have relied on properly chosen software for modeling and simulation tasks and inclined to integrate existing software and used a set of tools as a whole to achieve our goals. This was always rather a challenging task as the integration of different modeling and simulation tools was impossible without additional work. Our efforts devoted to standardize communication processes, simplifying the integration, deployment

and additional requirements like remote simulation and model manipulation lead to the creation of a *service oriented architecture* where modeling and simulation is treated as a service (MSaaS). The goal was to have extensible architecture that allows integration of various modeling and simulation tools with different levels of functionality that can be used as a multiparadigm platform for modeling and simulation.

In this paper, we present the architecture and show how the Petri net simulation tool *Renew* [2], [3] can be integrated as a modeling and simulation service and used to simulate a sample traffic control system represented by a Petri net model that is able to adjust its parameters via reflective simulation in a distributed simulation environment.

2 Service Oriented Architecture for Modeling and Simulation

Service Oriented Architecture (SOA) as a software architecture with loosely coupled services is not tied to specific technology and independent services can be accessed without knowledge of their underlying implementation. It can be applied in many different scenarios including web-based services running on different network nodes (servers) or in more challenging cloud environment. The comparison with existing architectures is in [4]. We employ the service-oriented approach mainly to:

1. Enable integration of existing software to supplement our tools that support dynamic and interactive development of systems with unclear specification.
2. Allow reusability of existing tools/applications/services so that they can be widely used.
3. Enable remote simulation and model management.
4. Simplify the installation, deployment and administration of services
5. Allow better scalability and on demand computing power in a distributed simulation environment

2.1 Architecture Details

The simulation architecture is based on web services. Web services stand for discrete web-based applications that provide a software function over a network. Web service has an interface described in *Web Service Description Language (WSDL)* and interacts with other systems in a manner prescribed by its description using a *Simple Object Access Protocol (SOAP)* that defines a messaging framework.

The advantages of web services allowed us to create a simulation framework with well-defined API that can be used by developers. On one side, there is a set of possibly interconnected services and on the other side, there is a client using this interface to perform various tasks, all depending on abilities of the service.

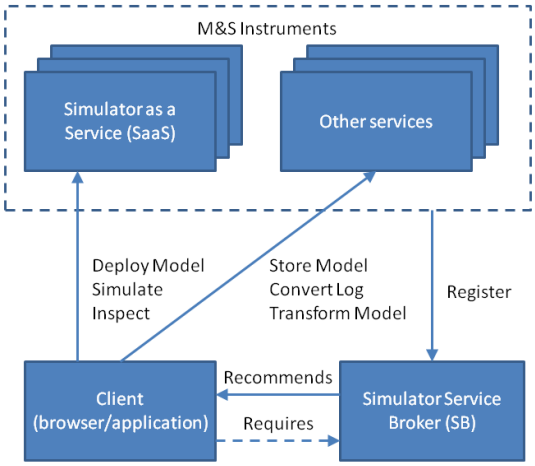

Fig. 1. Service oriented architecture

The architecture is shown in Fig. 1 and its main components are: *simulator as a service (SaaS)*, *service broker (SB)*, *other services* and *client*.

Simulation service allows clients to deploy and simulate models and can be used to inspect running simulations. Every service has a defined interface and allows the client to perform a set of operations. A service doesn't need to support every possible operation and may publish just a few of them - in practice there are a variety of services with different features and clients can choose the proper one that fits their needs. Operation categories are shown in Table 1.

Services are registered in and located by *Service broker* that acts as *UDDI registry (Universal Description, Discovery and Integration)*. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with services listed in its directory. Service broker is used by clients also for a service recommendation and may have implemented a set of rules to decide what service to retrieve. A simple recommendation mechanism may be based on model format or preferred formalisms. Non-simulation services are also registered via Service broker and a client may get its description documents and start using them.

Client is a service consumer. It can use Service broker's registry to locate requested or recommended service and then it communicates directly with simulation or other services. The communication between the client and a service is done via SOAP and it may be a simple request to start a simulation of a specific model: if the model is accepted and the simulation is started, the simulation service responds with the simulation identifier for further reference. Additional configuration parameters may be provided by the client in the request and the model payload can be sent either directly in the XML document, via *SOAP*

attachments or using SOAP *Message Transmission Optimization Mechanism (MTOM)* [9].

Table 1. Service operations

Operation category	Description/Purpose
Administration and configuration	Simulation control Simulator/Service configuration
Inspection and monitoring	Model inspection Simulation inspection Gathering log information Simulator or service monitoring Event Notifications
Model/Simulation Manipulation	Model design Editing actions over models Actions over simulation objects
Storage access	Model repository Simulation repository

Other services provide non-simulation but not less important functions. One of them is the *log presentation service* that can be used to convert raw outputs from simulators to more comprehensive and human readable graphical reports. Our implementation of a log presentation service exists in a Squeak environment [20] empowered with the SoapOpera and was integrated into the simulation environment as a web service. *Model transformation service* is another useful service for developers that is designed to translate models between different formalisms so that a developer can use one formalism for the design and another for the simulation. We have experimented with simple Petri net models that were translated to models in classic DEVS formalism (Discrete Event Systems Formalism) [1] according to the approach presented in [11]. *Model library* service provides very basic functionality. It is a common place where models together with other related documents can be stored as project resources and shared with other clients. It may also offer more sophisticated access control.

2.2 Integration of Existing Tools

To integrate and deploy existing applications or tools and to transform them into simulation services, we use *Apache Axis2* [12] as a core engine for web services. It is a re-designed and re-written successor to the widely used *Apache Axis* [13] SOAP stack with implementations available in Java and C languages. Axis2 provides us with a capability to add a web service interface to the existing application and can function as a standalone server where a service can be deployed without restart. It supports WSDL 2.0 and SOAP 1.2 which allows us to easily build stubs to access remote services.

Renew as a Service Petri net simulation tool Renew is a Java-based multi-formalism editor and simulator that provides a flexible modeling approach based on *reference nets* [15]. We have extended the Renew tool so that it can be deployed in Axis2 server and may act as a service that clients can use for simulation purposes - moreover, the client can be another Renew service and this will allow us to execute nested simulations remotely in the case study below.

To develop a web service, we use a *bottom-up approach*, where implementation of a service has to be coded first. A service operation logic is in Java classes where methods have simple contracts that allow a caller to start or stop simulations, to configure the service, to get logs, etc. This implementation requires the Renew application to be installed on a target system, where it is executed as an external program. Service classes are used to create a deployable service artifact - it is a single archive file and we deploy it in the Axis2 server where it is immediately accessible without reload. We use scripts that automate the service deployment and the installation of the Renew application. Java libraries that may be needed for a model execution can be either pre-installed together with Renew or can be provided by the service later as well.

To be able to invoke service methods over a network, a client that understands the service API has to be implemented. For this purpose, we can use a service description file, i.e. a WSDL file that Axis2 server generates on demand for every deployed service. Then Apache Axis2 tools can be used to create stub classes that may be called from another Java program to invoke the remote Renew service.

Although Renew already supports remote simulation with RMI (Remote Method Invocation), we use SOAP as a messaging framework for communication between services. SOAP has significant advantages over RMI. It uses XML, it is language independent and allows platform independent services whereas RMI is Java-centric. SOAP as a more robust technology allows functionality to be accessible to a variety of clients and it allows a loosely coupled architecture. Although RMI has smaller latency, it is more suited to smaller applications where objects must stay in sync in all applications.

2.3 Simulation as a Service in Cloud Environment

The deployment of simulation services in cloud environments boosts efficiency, allows scalability, simplifies deployment and administration tasks and allows better control over the simulation environment. Especially in cases when the tool integration, installation and service configuration is rather a complicated task, it is worth having a virtual machine with a service or a set of services preinstalled and available in the cloud environment. By maintaining a preconfigured image you can avoid complicated service setup. An easy deployment of virtual machines from captured image with pre-configured services increases architecture scalability for demanding scenarios in multi-simulation environments. We have used Windows Azure [14] to successfully deploy and manage simulator services in the cloud environment.

3 Case study

To demonstrate a Petri Net simulator as a service, we have created a sample model of a traffic control system that controls lights on a simple crossroad. The model is able to adjust its configuration via reflective simulation. Renew was effectively used with all its advantages to design and build the model and when the tool was deployed as a service, it could operate as a remote simulator and as a client at the same time allowing nested simulations to be executed remotely.

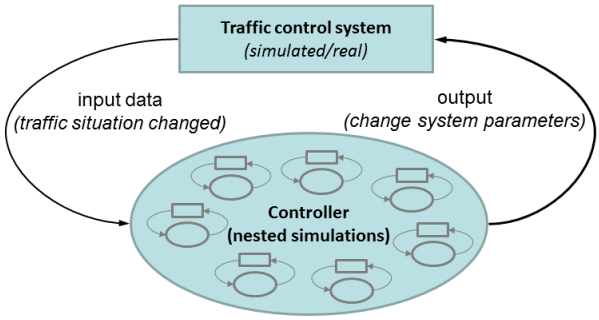

Fig. 2. Concept of reflective simulation with main components

3.1 Reflective simulation

The main concept of reflective simulation is depicted in Fig. 2 and Fig. 3. In Fig. 2 we can see a traffic model with its control system and the way they influence each other. The control system reacts to changes in the traffic (e.g. increased volume of vehicles) which may trigger a change of traffic control system settings (e.g. signal timing plans) and this further influences the traffic (e.g. improves throughput) which may again require a change to control system settings and so on. In this scenario an undesired oscillation may happen - imagine a situation when increasing the green interval in one direction may cause excessive queuing of vehicles waiting in the other direction. In this case, it would be better if the control system could anticipate the effect of the change, infer on its own future behavior and adjust the signal timing appropriately. The control system is able to trigger nested simulations, to simulate its own behavior and to perform decisions based on collected results. We call this a reflective simulation. You can compare it with the approach to nested simulations presented in [17] or [18].

3.2 Model

The model comprises of three main components, each of them being a Petri net: the *traffic model*, the *traffic control system* and the *simulation control model*. The

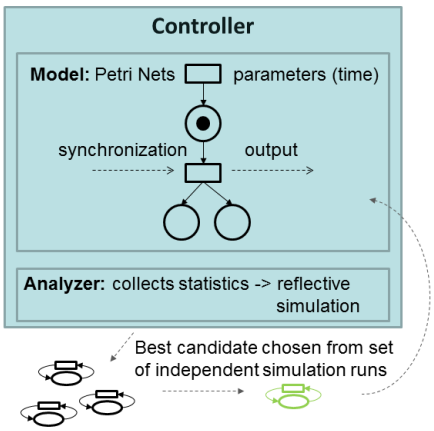

Fig. 3. Simulation control model and its role in reflective simulation

crossroad and its traffic control model were inspired by the example presented in [16] and further modified to our purpose. Other components - the odel of traffic and the simulation control model (in Fig. 4) were created from scratch.

Traffic Model The traffic flow at the crossroad is divided into three phases with blocking and non-blocking directions. In the traffic model net, tokens represent vehicles that travel through the intersection. Each lane and each direction has assigned a time it takes every vehicle to travel through the crossroad. The model contains a statistics part that is dedicated to gathering statistics about the traffic. This comprises the number of vehicles waiting to cross the intersection in each lane and the number of vehicles that successfully travelled through the crossroad in each phase. A *Synchronous channel* concept (uplinks and downlinks) is used to trigger statistics gathering and to deliver results to the simulation control model. The traffic model net has a part that allows its initialization when running in a nested simulation - e.g. it sets a number of waiting vehicles before the simulation starts.

Traffic Control System The Petri net representing the traffic control system is presented in [16]. It reflects three phases of a crossroad, each with signal lights for pedestrians and vehicles. In every phase, only vehicles from the corresponding phase in the traffic model net are allowed to move through the intersection. The actual behavior of the traffic control system is determined by its main parameters that contain time values for signal timing plans. In comparison with [16] the control system net was enhanced to allow changes to its configuration (signal timing plans) and to communicate the active phase to the traffic model net.

Simulation Control The Petri net in Fig. 4 controls the simulation and acts as a controller as shown in Fig. 3. It configures the traffic control system

according to collected data from the traffic model and based on results from nested simulations. More specifically, it:

1. Initializes traffic model and traffic control system nets
2. Starts the simulation
3. Gathers and analyzes data from the traffic model
4. Communicates with other simulation services
5. Triggers nested simulations of its own model with different configurations
6. Collects and interprets logs from nested simulations
7. Changes traffic control system settings according to results from nested simulations

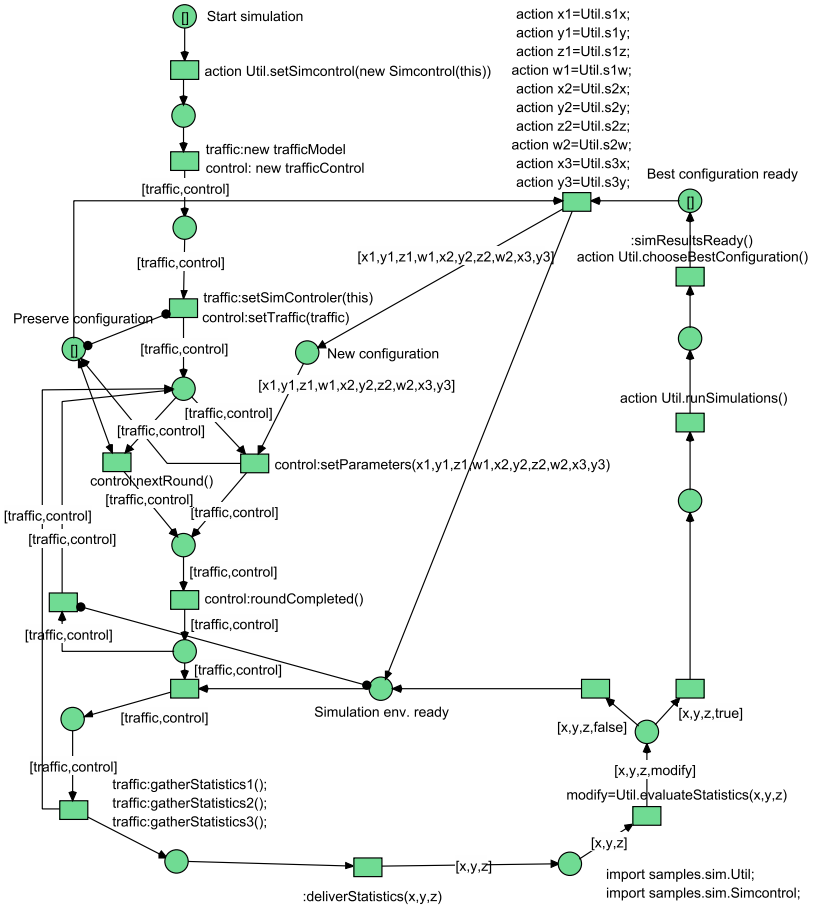

Fig. 4. Simulation control

As reference nets work seamlessly with Java programs, part of the model is implemented in Java language and it is called via action inscriptions or method invocations. This enables access to remote services directly from the Petri net model. For this purpose two Java classes are imported in the bottom right corner in Fig. 4. The *Util* class is used to analyze traffic data and to spawn nested simulations. Under the hood, a small framework with a set of Java libraries is used for communication with remote simulators. The *Simcontrol* is a stub class that wraps the simulation control net. It forwards its method calls to synchronous channels of the wrapped net instance and enables them to be used from Java.

The simulation control net instantiates the traffic net and the traffic control system net and keeps their references, named as *traffic* and *control*.

After the initial configuration the system starts to control the traffic. According to statistics gathered from the traffic model (*traffic: gatherStatistics()*) that are evaluated (*modify=Util.evaluateStatistics()*), the system may decide to reconfigure signal timing plans in traffic control net to increase the throughput. Then a set of configuration candidates is suggested and a nested simulation is executed for each of them to see if it brings some improvement (*action Util.runSimulations()*). The master simulation control model acts as a client. Each simulation is controlled by a single Java thread that communicates with either a local or a remote simulation service(s). Although we can configure the level of nesting and a model in a nested simulation may further trigger other nested simulations of itself, our tests show that there is no significant benefit, especially when compared to increased demand for resources.

After some time, remote simulations are stopped and logs are acquired and interpreted. The simulation that improved the throughput more than others is chosen as the best candidate and its configuration is applied to the traffic control system when possible. A brute force computing is used to achieve continuous improvement and more than one round of nested simulations may be executed. However, the model quickly converge to the optimal configuration, especially when changes in the traffic flow are less extreme.

A model containing all three nets (simulation control net, traffic control system net and traffic model net) is sent to the simulator service along with all necessary libraries, configuration and startup parameters. As we use Renew's non-graphical simulator for nested simulations, the model must be sent as a single merged file in shadow net format. As a result the transmitted model contains only the semantic information and not the visual appearance and it can be used only by a non-graphical simulator. To preserve the graphical information the PNML (Petri Net Markup Language) may be used instead when transferring the model.

Acknowledgement. *The work was supported by the EU/Czech IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070 as well as the internal BUT projects FIT-S-12-1 and FIT-S-14-2486.*

References

1. Zeigler, B.P., Kim, T.G., Praehofer, H.: Theory of Modeling and Simulation, Second Edition. Academic Press (2000)
2. Renew - The Reference Net Workshop, http://www.renew.de
3. Kummer, O., Wienberg, F., Duvigneau, M., Kohler, M., Moldt, D., Rolke, H.: Renew - the Reference Net Workshop. In: Tool Demonstrations, pp. 99–102. 24th International Conference on Application and Theory of Petri Nets 2003. Department of Technology Management, Technische Universiteit Eindhoven, Beta Research School for Operations Management and Logistics (2003)
4. Janousek, V., Polasek, P.: Modeling and Simulation Management in Distibuted Environment Using Web Services, In: WOSC 2008 - 14TH International Congress of Cybernetics and Systems. Wroclaw (2008)
5. Johns, K., Taylor, T.: Professional Microsoft Robotics Developer Studio, Paperback, Wiley (2008)
6. Coen-Porsini, A., Gallo, I., Zanzi, A.: Integration of web based simulators in the SINPL platform. In: Proceedings of ESM 2006. Ghent, BE, pp. 259–263 (2006)
7. Dahmann, J.S., Fujimoto, R.M., Weatherly R.M.: The Department of Defense High Level Architecture. In: Proceedings of the 1997 Winter Simulation Konference (1997)
8. Multi-Simulation Interface (MSI) Brochure, http://msi.sourceforge.net
9. W3C - SOAP Message Transmission Optimization Mechanism http://www.w3.org/TR/2005/REC-soap12-mtom-20050125
10. SmallDEVS, www.fit.vutbr.cz/~janousek/smalldevs
11. Jacques, C.J.D., Wainer, G.A.: Using the CD++ DEVS Tookit to Develop Petri Nets. In: Proc. of the 2002 Summer Computer Simulation Conference, San Diego, CA, USA (2002)
12. Apache Axis2 - Apache Axis2/Java - Next Generation Web Services http://axis.apache.org/axis2/java/core
13. Web Services - Axis http://axis.apache.org/axis
14. Windows Azure cloud platform http://www.windowsazure.com
15. Cabac, L., Duvigneau, M., Moldt, D., Rölke H.: Modeling dynamic architectures using nets-within-nets. In: Proceedings, volume 3536 of Lecture Notes in Computer Science, pp. 148–167. 26th International Conference, Applications and Theory of Petri Nets 2005, Miami, USA (2005)
16. Turek, R.: Modelovani vybranych dopravnich problemu s vyuzitim Petriho siti (in Czech) [Modeling of Chosen Traffic Problems with Petri Nets]. In: Posterus.sk, Portal pre odborne publikovanie [Portal for Scientific Publications], vol. 3, nr. 12 (2010)
17. Sklenar, J. - Introduction to OOP in Simula (Nested Simulation) http://staff.um.edu.mt/jskl1/talk.html
18. Kindler, E. - Reflective Simulation - Simulation of Systems That Simulate. In: Proc. of ESM - European Simulation and Modeling, Porto, Portugal (2005)
19. Chandrasekaran, S., Cardoso, J., Silver, G., Miller, A.J., Sheth, A.P.: Web service technologies and their synergy with simulation. In: Proceedings of the 2002 Winter Simulation Conference, pp. 606–615. San Diego, California (2002)
20. Squeak http://www.squeak.org