Improving Performance of Complex Workflows: Investigating Moving Net Execution to the Cloud

Sofiane Bendoukha and Thomas Wagner

University of Hamburg, Department of Informatics
`http://www.informatik.uni-hamburg.de/TGI/`

Abstract. In this paper we propose and discuss mechanisms and implementation issues for moving the execution of computation- and time-consuming workflows into the Cloud. These complex workflows are specified by Petri nets, more precisely reference nets using the Renew tool. We believe that Cloud technology is a suitable solution to (i) overcome the lack of resources on-premises and to (ii) improve the performance of the whole system based on quality of service (QoS) constraints. As execution target for simulations, tests have been performed on an OpenStack Cloud. Furthermore, the integration and interfaces between workflows, Cloud computing and agent concepts are also addressed.

Keywords: Petri nets; Cloud Computing; Workflows; Multi-agent Systems; Reference Nets; Paose.

1 Introduction

Several long-running and high-throughput applications can be designed as complex workflows, which describe the order and relationships between the different activities and related data (input, output). In such scenarios, these tasks often need to be mapped to distributed resources, possibly due to a lack of on-premise resources or failures. Recently, Cloud computing has attracted more interest from both the industry and academic community. Cloud computing is a recent computing paradigm. It has its origin in distributed computing, parallel, utility and grid computing. The National Institute of Standards and Technology (NIST) defines Cloud computing as: *"A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* [19].

In fact, Cloud technology provides an environment that allows to dynamically allocate resources for the execution of workflow tasks following an on-demand and pay-as-you-go model. In this work, we aim to take advantage of these resources to improve the performance of the applications. These applications are, in our case, specified as Petri nets using the **RE**ference **NE**ts **W**orkshop (Renew) editor. In order to make this possible, we need to provide mechanisms and strategies that are based on the integration of workflow concepts and Cloud technology (and later the agent paradigm). There are different ways to address this. On the one hand, Cloud for workflow uses Cloud resources to execute complex workflows and especially scientific workflows [12] [13]. Such works are more resource-centric and focus on the computational tasks. On the other hand, Clouds need structured and mature workflow concepts and high-level languages to handle issues like managing complex task and data dependencies. It should be noted that we only consider moving the execution of entire nets or systems of nets into the Cloud. Further distribution aspects of the simulation/execution[1] are outside of the scope of this paper.

The research described in this paper focuses more on performance issues, which can be considerably improved by using Cloud resources. We present our approach to provide techniques and tools to move the execution of complex workflows modelled in Petri nets to the Cloud. The migration to the Cloud is based mainly on user requirements. Thus Quality of Service (QoS) parameters are specified in advance. We emphasise response time and cost constraints, but this can be easily extended to other QoS parameters such as service availability. Modelling and execution of Petri net models is performed exclusively through the Renew editor. Furthermore, we discuss different realisation possibilities. We examine three different types of interfaces, which define how input and output to the Cloud calls are defined. Simple interfaces provide only basic functionality to initiate Cloud workflows and receive results. Simulation interfaces are used to run extensive simulations of workflows in a Cloud environment. Lastly, advanced interfaces feature advanced mechanisms to process input and output data for the Cloud. The main avenue of thought for the advanced interfaces is to utilise autonomous software agents and their characteristics.

This paper is structured as follows. In Section 2, we present the conceptual and technical background as well as related work. Section 3 introduces the approach and methodology for moving net simulations to the Cloud. Section 4 proposes the different kinds of interfaces. Finally Section 5 discusses the approach and Section 6 concludes the paper and presents future work.

2 Background and Related Work

In this section we are going to discuss the conceptual and technical background for this work. For the specification of workflows Petri nets and especially *reference nets* are employed. Related work is also presented.

2.1 Reference Nets

Reference nets were introduced in 2002 by Olaf Kummer (see [14]). Reference nets are modelled and simulated using the Renew editor and simulation tool [16]. Both are described in the Renew manual[2]. In reference nets, tokens can be

[1] We use the terms simulation and execution interchangeably. If a distinction has to be made it will be clear from the context.

[2] The latest version of Renew, documentation and articles are available on (http://www.renew.de)

anonymous, basic data types or references to Java objects or other reference nets. The referenced objects can be of any class in the Java programming language.

Firing a transition can also create a new instance of a subnet in such a way that a reference to the new net will be put into a place as a token. This allows for a specific, hierarchical nesting of networks, which is helpful for building complex systems in this formalism. The creation of instances is similar to object instances in object-oriented programming languages and the usage of references allows to construct reference net systems, whose structures are not fixed at build time.

2.2 Renew

As mentioned above, we use Renew for the modelling of workflows. Renew is a graphical tool for creating, editing and simulating reference nets. It combines the 'nets within nets' paradigm of reference nets with the implementation power of Java. The Renew plug-in architecture, which was developed and introduced in [22], allows the extension of Renew with additional functionality through the use of interfaces between Renew components without changing the core of Renew. Additional functionality (e.g. additional net formalisms, simulation and verification tools, interface extensions) can be added to Renew by providing the Java classes and nets for the new plug-in. Many such plugins have already been developed, which makes Renew a versatile and extensive Petri net tool.

2.3 Agents

We also utilise software agents for advanced features regarding the interface to the Cloud execution (see Section 4.3). We use the Mulan (**Mul**ti **A**gent **N**ets [21]) reference architecture and its implementation Capa (**C**oncurrent **A**gent **P**latform **A**rchitecture [10]). Both have been created and implemented using Renew and the majority of the executable code are in fact reference nets. Agents are executed in a distributed environment and generally communicate via standardised asynchronous messages. They can feature intelligence, reactive and proactive behaviour, and autonomy. These kinds of properties are utilised for the Cloud execution.

2.4 Related Work

Originally, WfMS were not conceived to be used in Cloud-like environments. With the growth of Cloud computing, several traditional WfMS improved their kernel and are now able to provide interfaces to communicate with external Cloud services. The prevalent (scientific) WfMS are: Taverna[18], Pegasus[9], Triana[23], Askalon[11], Kepler[2] and the General Workflow Execution Service (GWES)[1]. The originality of these systems is that they run on parallel and distributed computing systems in order to reach a high level of performance and get access to wide range of external resources. The Pegasus system allows scientists to execute workflows in different resources including clusters, Grids and Clouds.

This has been adapted later to execute scientific workflows in the Cloud (within an Amazon EC2 Instance) [17]. Compared to our work, the *migration* to the Cloud is almost the same, the difference lies at the modelling level, where we use Reference nets as modelling technique. GEWES is an interesting project that makes use of high-level Petri Nets (HLPN) for the description of workflows. The GWES coordinates the composition and execution process of workflows in arbitrary distributed systems, such as SOA, Cluster, Grid, or Cloud environments. In the workflow specifications, transitions represent tasks and tokens represent data flowing through the workflow.

There have also been many more efforts to infuse Cloud and distribution aspects into general workflow management. The ADEPT project [8, 20] focuses on flexible and adaptive workflow management but also deals with distribution and migration aspects to avoid performance bottlenecks in the network. Another interesting combination of Clouds and workflows is the OpenTosca project [5]. It utilises management plans implemented as workflows to configure Cloud applications for organisations. [24] also deals with configuration issues but focuses explicitly on the configuration of interorganisational business processes in the Cloud. The issues addressed by these and more publications represent advanced features of workflows in Clouds. They are outside the scope of this paper. Some of these issues are, however, considered future work.

3 Renew in the Cloud

Renew *in the Cloud* designs the process of simulating Petri net models not locally (i.e. on-premises) but in the Cloud. There are different reasons why we are moving the simulation to other execution environments but the main reason is to seek gains in performance. Especially (Petri net) models that contain complex and time consuming tasks are of interest here. In our approach the design/modelling step is performed at the user's side since it does not require computing or storage capabilities. After this, the models are pushed to the Cloud provider. The Cloud provider should be able to provide instances, that support Petri net simulations. Therefore, Cloud instances need to be *provisioned* by external Petri net editors and simulators. Since our chosen editor is Renew it will be installed and configured before starting the simulation. The whole process consists of the following steps:

1. modelling the workflow
2. configuring the Cloud instance
3. starting/connecting to the Cloud instance
4. uploading the required nets
5. executing the simulation and getting the results

Technically, our work is based on the Vagrant tool[3], which permits us to create reproducible development environments. According to the Vagrant homepage, Vagrant *"is a tool for building complete development environments. With*

[3] https://www.vagrantup.com/

an easy-to-use workflow and focus on automation". There are three ways to use Vagrant: with a virtual machine, a Cloud provider, or with VMware.

For creating a Vagrant machine the vagrant tool first needs to be installed as well as the VirtualBox. For both Vagrant and VirtualBox, the installation is possible on the three famous operating systems: Linux, Windows and Mac. Next, a configuration file called *Vagrantfile* is mandatory to configure a Vagrant machine. It is a Ruby file used to configure Vagrant and to describe virtual machines required for a project as well as how to configure and provision these machines. Finally, the guest Vagrant host can be started using the command *vagrant up.*

3.1 First Prototype (with VirtualBox)

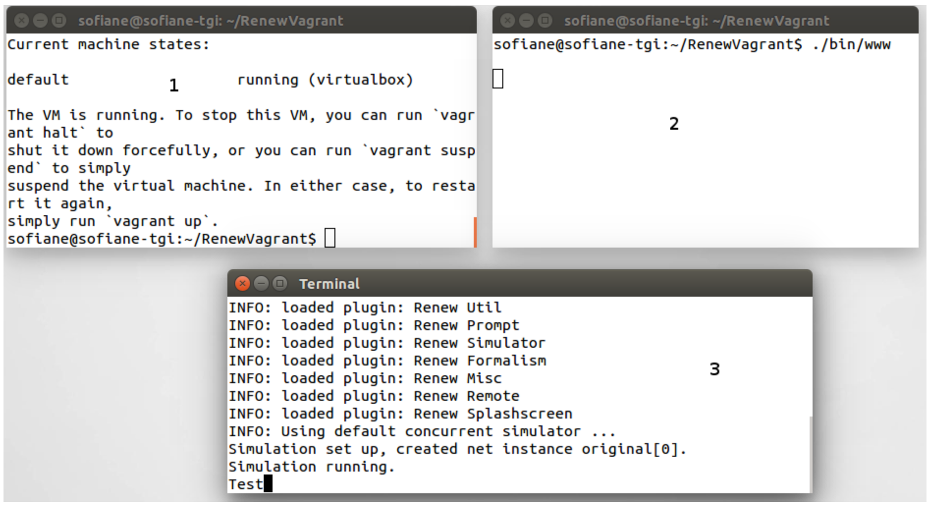
To run Renew and all required software on the host machine, configuration using a Vagrantfile is needed. The latter permits to provision the host machine(s) with additional softwares (in our case Renew). Since Renew requires Java 6 or later, this portion of code shows instructions that should be added.

Figure. 2 shows the steps to follow for the execution (simulation) of a workflow (Petri net). First of all, the workflow is modelled using Renew and generates .rnw files. It should be noted that, for now, we focus solely on the simple *execution* of workflow nets in the Cloud. Workflow *management* aspects are currently considered in the background. For example, human interaction with the workflow, e.g. a user executing a task, is currently only simulated by the system. Later on it is possible to incorporate a workflow management system in the Cloud which would support these kinds of aspects. Workflow management within Renew implemented as a reference net (agent) system, which would be executed in the Cloud, is already possible [25]. For now, the vagrant machine is equipped with a Renew version without a graphical user interface, i.e that we are obligated to run the simulation with the command line. The correspondent console command is *startsimulation.* The syntax of the command is:

startsimulation <net system > <primary net> [-i]

- *net system*: The compiled net files (.sns files, Shadow Net System).
- *primary net*: The name of the net, of which a net instance shall be opened when the simulation starts. Using the regular GUI, this equals the selecting of a net before starting the simulation.
- *-i*: This must be set before starting the simulation (only for this prototype). Concretely, we use -r, which means to run the whole simulation without steps. More information about this command can be found in [15, p.106].

For testing purposes we created a simple net (primary net) that contains a single transition that prints a string on the screen. Since the reference net formalism allows using java code, this is done simply by the instruction System.print.out("message") (see Figure. 3). Once the required files are prepared (.rnw and .sns), they are sent to the Vagrant machine. The nets are either copied to the synced directory with the Vagrant machine or with *scp.* To start the simulation on the guest machine, there are three possibilities: (i) by a command



```
sofiane@sofiane-tgi: ~/RenewVagrant
Current machine states:

default                    running (virtualbox)

The VM is running. To stop this VM, you can run `vagr
ant halt` to
shut it down forcefully, or you can run `vagrant susp
end` to simply
suspend the virtual machine. In either case, to resta
rt it again,
simply run `vagrant up`.
sofiane@sofiane-tgi:~/RenewVagrant$
```

```
sofiane@sofiane-tgi: ~/RenewVagrant
sofiane@sofiane-tgi:~/RenewVagrant$ ./bin/www
```

```
Terminal
INFO: loaded plugin: Renew Util
INFO: loaded plugin: Renew Prompt
INFO: loaded plugin: Renew Simulator
INFO: loaded plugin: Renew Formalism
INFO: loaded plugin: Renew Misc
INFO: loaded plugin: Renew Remote
INFO: loaded plugin: Renew Splashscreen
INFO: Using default concurrent simulator ...
Simulation set up, created net instance original[0].
Simulation running.
Test
```

Fig. 1. Run Simulation in a Vagrant Machine

line (using nohup and ssh) (ii) through a web Gui (using NodeJS) (iii) from a reference net directly (inscribed to transitions). Figure 1, shows the process of starting a vagrant machine and launching Renew and the simulation. Executing the command in **2**, launches a new terminal and starts Renew and simulate the net on the Vagrant machine. (1) The Vagrant machine should be up and running (2) The web server (NodeJS) is started (3) Renew is launched and a simulation is started with the required nets.

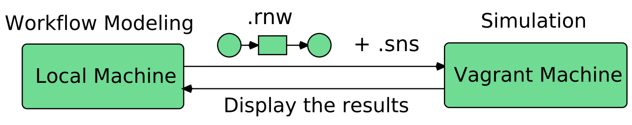


Fig. 2. Remote Simulation with Vagrant

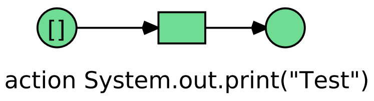


Fig. 3. The Original Net (.rnw)

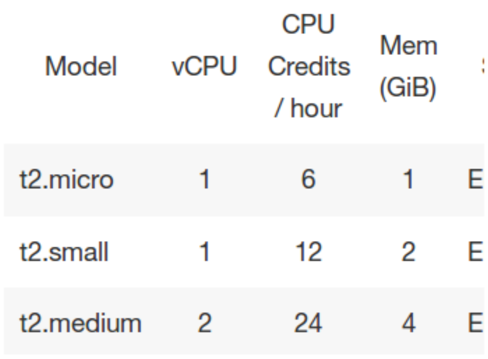
3.2 Second Prototype (with OpenStack)

The second version of the implementation is based on a concrete Cloud environment. The instances are not launched in a virtual machine at the host machine, but in a Cloud (see Figure. 5). We mentioned before that *Vagrant* uses specific providers. The default one is *VirtualBox*[4]. Other built-in providers are VMWare[5], Docker[6] and Hyper-V[7]. When executing *vagrant up* we will have a virtual machine created on the local host. If we require only one VM then it is enough to work locally. Nevertheless, when the number of VMs grows we will face an overload due to a lack of resources. The natural solution is to look for external resources which, in our case, are available in a Cloud. Due to financial and technical constraints, in our testbed we use an open source Cloud framework called *OpenStack*[8]. *OpenStack* is an open source software for creating private and public Clouds. It is installed on a CentOS Linux operating system. Thanks to the plug-in architecture that *Vagrant* is based on, we are able to connect to different Cloud providers and launch our instances. This is performed by a plug-in called *vagrant-openstack-provider*[9]. This plug-in permits to control and provision machines within an OpenStack Cloud. Other features are for instance: Create and boot OpenStack instances, SSH into the instances and suspend and resume instances. The principles for running Renew simulation in the Cloud are almost the same as presented in the previous section. We still need to upload the required nets (.rnw and .sns) to the VM. The difference is at the configuration level, which is realised by the *Vagrantfile*. A minimal configuration consists of the following:

```
require 'vagrant-openstack-provider'
Vagrant.configure('2') do |config|
  config.vm.box         = 'openstack'
  config.ssh.username = 'stack'
  config.vm.provider :openstack do |os|
  os.openstack\_auth\_url = 'http://keystone-server.net/
  v2.0/tokens'
  os.username              = 'openstackUser'
  os.password              = 'openstackPassword'
  os.tenant\_name            = 'myTenant'
  os.flavor                = 'm1.small'
  os.image                 = 'ubuntu'
  os.floating\_ip\_pool     = 'publicNetwork'
  end
end
```

[4] www.virtualbox.org
[5] www.vmware.com
[6] www.docker.com
[7] www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx
[8] www.openstack.org
[9] https://github.com/ggiamarchi/vagrant-openstack-provider

The configuration presented above concerns only the credentials and the image used to boot the instances. The important next step is to configure these instances to be able to handle Renew simulations. The configuration is performed exactly in the same way as working with virtual machines (VirtualBox). Configuring instances plays an important role and directly affects the performance of the system. Although, for testing purpose, we worked on a private OpenStack Cloud , our implementation can be integrated within commercial Cloud providers like Amazon[10], Windows Azure[11] or HP[12]. With providers, Cloud consumers can configure their instances based on a pay-as-you-go model. Resources provided by commercial Cloud providers are not free, which can negatively affect the choice of the Cloud consumers. With respect to the application requirements, there are different types of instances which depend on the Cloud provider. Instance types describe the compute, memory and storage capacity of the instances that Cloud consumers use for hosting (computing) their applications. Therefore, the requirements for the applications should be clearly specified as QoS parameters. This issue has been already addressed in [3]. QoS parameters can be specified as inputs to the transitions. For example, with OpenStack these are called by names such as "*m1.large*" or "*m1.tiny*". Figure. 4 shows the characteristics of T2 instances.



Model	vCPU	CPU Credits / hour	Mem (GiB)	
t2.micro	1	6	1	E
t2.small	1	12	2	E
t2.medium	2	24	4	E

Fig. 4. Amazon T2 Instance Characteristics

[10] http://aws.amazon.com/
[11] http://azure.microsoft.com
[12] http://www.hpcloud.com/

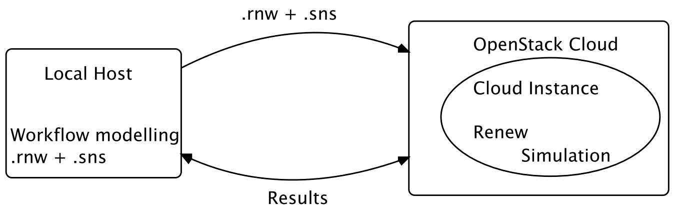


Fig. 5. RENEW Simulation in the OpenStack Cloud

4 Interface

In the previous section we have described how to enable RENEW simulations in a Cloud environment. The basic technical realisation bundles up and simply executes a workflow net system and its shadow net representation. In order to practically utilise this execution we need to define an interface for it. We have examined possible interfaces that can be grouped into three categories: Simple, Simulation, Advanced. These categories will be discussed in Sections 4.1 through 4.3.

Prototypes for the simple interfaces already exist. More features for these interfaces as well as the simulation and advanced interfaces are currently under development. They will be discussed on a conceptual level here.

Note that how exactly a simulation as a Cloud functionality is called has already been discussed in the previous section. Generally it can be called either via the console, a web interface or directly within a (local) running net system. If an interface restricts these possibilities it will be shortly addressed.

4.1 Simple Interfaces

Simple interfaces offer basic, yet versatile functionality that can later be utilised in more complex settings. The input for simple interfaces remains the workflow system and its shadow net representation. The output options vary, but share that any results obtained are returned as simple data or objects. Simple interfaces do not support any kind of intelligence or autonomy. They are simply called when needed and report back the predefined results.

Console Interface This interface uses either the internal RENEW console or the general system console as the output medium. Consequently it already directly works with reference nets. By simply inscribing a `System.out.println(textVariable)` to any transition of the net system being executed in the Cloud the String representation of the object *textVariable* is printed on the console. Figure. 1 already shows a working prototype using such a console interface.

For very simple use cases (e.g. testing a certain outcome of the net system) this is already sufficient, but in most cases any obtained result should automatically be made available to the caller in a more utilisable way. This can be realised

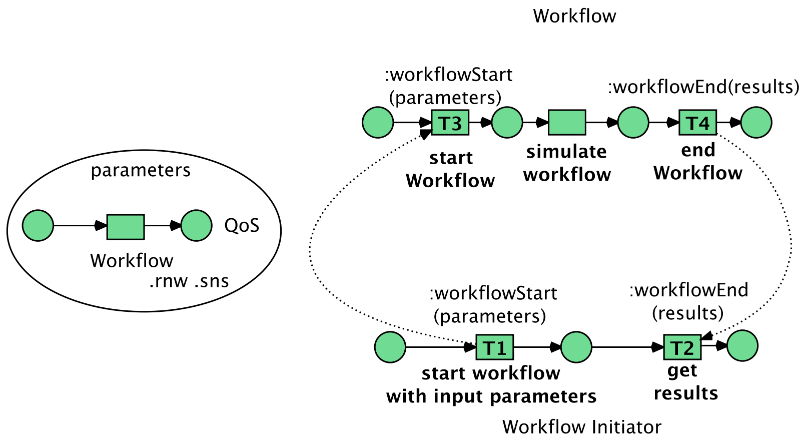


Fig. 6. Synchronous Channels Interface

by reading any output in the console and combining these outputs into a result object that is passed back when the execution has been completed. This way more complex use cases and computations can also be supported even with this very simple interface. One problem with this approach is that it is not standardised or regulated by the modelling approach. This is a general problem that will be discussed in Section 5.

Synchronous Channel Interface Realising the interface through synchronous channels is another way of providing a simple interface. Synchronous channels, in general, are a mechanism to allow data and object transfer between net instances. They were first introduced in [7] and are fundamental to the reference net formalism. Within the Cloud context synchronous channels allow for data objects created and modified during the execution of a workflow to be transferred back to its initiator or even directly into other running (local or remote) net systems. Consequently, the full potential is realised when the Cloud call is incorporated into a net system. There are a number of ways in which synchronous channels can be incorporated into an interface for Cloud-based workflows. The simplest way is to explicitly inscribe an output channel to a transition in the net. When this transition fires the synchronous channel is called and the specified data object transferred to the Cloud call initiator. By extending this to multiple transitions we can realise a kind of continuous feedback for the initiator. Whenever a transition inscribed with the synchronous channel would fire a result would be send to the initiator.

Figure 6 illustrates the approach mentioned above. There are two main nets: *Workflow Initiator* and *Workflow*. The Workflow Initiator manages the workflow locally and is responsible for the communication with the Cloud provider. On the other side, the Workflow is executed in the Cloud. After modelling the workflows, the model is saved in Renew (.rnw) and Shadow net (.sns) files. These files are required for the Workflow to be executed. The communication between both nets is possible through synchronous channels. For instance, T1 and T4 are for sending data; T2 and T3 are for receiving data. Furthermore, all the parameters can be put into a place instead of synchronous channels.

There are two main problems when using synchronous channels. First of all, similar to the console interface, this interface is not structured. Careless modellers may set output channels to incorrect transitions so that results may not be valid. Another issue is related to the continuous update mechanism. If (possibly partial) results are transferred back to the initiator at multiple times, it may be difficult to work with these results. Depending on the net a modeller would have to explicitly build against that specific interface in order to aggregate the results into a valid composition. For this simple interface it would be cumbersome and inefficient. This is one of the issues addressed by the advanced interfaces described in Section 4.3.

Up until now, synchronous channels have only been discussed for output scenarios. Incorporating synchronous channels for the input of the Cloud-based workflows is also possible. In the simplest option this would only be used to incorporate initial input data. This would not change the basic functionality all too much, as initial data can easily be supplied via the console or simply as the initial marking of the workflows. It would make it easier though to change the initial marking. If called from a running net system the Cloud workflow could be initiated with runtime information. A synchronous input channel would simply pass the data object directly into the workflow in the Cloud. Without synchronous channels a new net system with the specified initial marking would have to be created or the console call would have to be tailored to the runtime information.

It is also possible to transfer data into the running Cloud workflow. This would require the initiator to be able to maintain a connection with the Cloud system. This is mostly feasible when the Cloud call is initiated by a running net system which would continue with its own execution and provide additional data to the Cloud net system at some later point. Certain transitions in the Cloud net system could then be inscribed with an input channel over which this additional data could be received. Ensuring the correct connection and synchronisation between local and Cloud net systems is the main challenge in this context. This is currently considered future work and outside of the scope of this paper.

Using synchronous channels in the proposed ways has some disadvantages though. Without any restrictions to modelling the placement of input and output in the net would affect any verification of workflow correctness or other Petri net properties. This is discussed further in Section 5.

4.2 Simulation Interfaces

The simulation interfaces are not so much interfaces, as they are a utilisation of Renew in a Cloud environment. Instead of executing a net system remotely once for some direct usage these interfaces execute the net system a large number of times. The information about these simulation runs is then reported back to the initiator. This constitutes the output of these interfaces. The input consists, beside the net system and shadow net representation, of simulation parameters (e.g. number of simulation runs). The advantage of running these simulations in a Cloud environment is that it frees up the modellers local machine.

Result Simulation One possibility is to run a set of simulations and have the system report back the results of each run. With the same initial marking different simulations may still produce different results. This could be due to race conditions, non-deterministic behaviour, etc. With these results the modeller could validate assumptions about the net system or determine possible error sources.

This kind of simulation could be extended by enabling variable initial markings. Simulating a net system with differing parameters might influence the results and help modellers even more.

Timed Simulation Another possibility is to run a set of simulations and compare the time it takes to complete them. This kind of simulation is more useful for testing the performance or new features in the runtime environment, in our case Renew. Running the simulation with new features enabled and comparing the results obtained without them can yield information about new algorithms.

Focussing more on the performance of the net system it might be of interest to the modeller to determine the impact of different initial markings. Varying over the initial marking of the net system could then help modellers determine performance bottlenecks. When using (reference) Petri nets for processes in practical software engineering within the Paose (Petri net-based, Agent- and Organization-oriented Software Engineering [6]) development approach for example, such simulations and their results become especially useful and interesting.

4.3 Advanced Interfaces

The advanced interfaces go beyond simple call interfaces like the ones discussed in Section 4.1. They utilise these simple interfaces but add another layer of abstraction to them. This leads to additional characteristics like certain degrees of intelligence and autonomy. They can also feature mechanisms to manage and store known net systems so that they may even serve as a kind of directory service. They can also aggregate results, enforce quality of service concerns or choose the best from a set of results. Consequently, no general statements about input and output can be made.

Agent Interface Using agents for an advanced interface to the Cloud execution of Petri net systems has a number of intrinsic advantages. Agents possess autonomy and a certain degree of intelligence. Reactive and proactive agent behaviour can also be utilised.

In an advanced interface an agent would serve as a kind of gateway between the local net systems and the Cloud net systems. For the Mulan and Capa

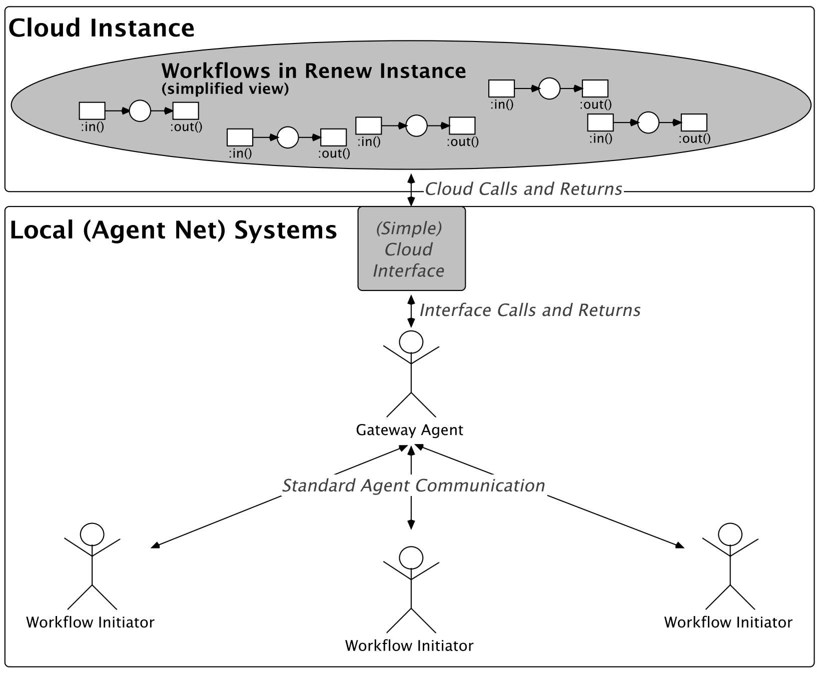


Fig. 7. Agent Interface Illustration

agents we utilise this would expand upon the ideas introduced by the WebGateway agent [4] towards Cloud calls. The WebGateway agent serves as a kind of bridge between the net execution of a Renew environment and the web environment. Agents in Renew can then offer their functionality as web services and also access remote web services.

For the Cloud context agents would serve in a similar fashion. The idea is illustrated in Figure 7. Some agents would be responsible for the net systems. They would take on the role of the initiator. They could act autonomously or be controlled by a human user via some kind of user interface.

These agents would control and/or create the workflows which should be executed in the Cloud. They would send requests and data to the gateway agent[13]. The gateway agent would then use a simple interface (see above) in its internal functionality to initiate the workflow in the Cloud on behalf of the other agents. Any result obtained in the Cloud would be send back to the gateway agent which would then forward it to the other agents.

[13] Alternatively the workflows could be stored in a database known to all agents. In that case the initiator agents would simply send requests and identifiers of the workflows to the gateway agent.

At this point the characteristics and advantages of software agents can be utilised. In the following we will cover some ideas of how, starting from the relatively simple approach described above, this can be done.

The gateway agent can aggregate the results of the Cloud calls into more informative composite results. Partial results could be incorporated into the workflows with standardised instructions for the gateway agents to combine them after the execution has been completed. The gateway agent can also instantiate the workflow multiple times and choose the best (or fastest) result. Of course, the gateway agent has to be equipped with mechanisms to aggregate or assess results in these fashions. This is, however, simply a question for the technical implementation and not a conceptual one. Aggregation of results is especially interesting for simulation purposes. The gateway agent could automatically create composite results for modellers to inspect. It could also automatically vary over the initial parameters based on the initial results (e.g. to validate results or test certain outlier data).

The gateway agent can also react to errors or other problems occurring during the execution in the Cloud. If the Cloud execution returns an error the gateway agent can retry the instantiation. If the error was caused by the call it can also adapt the call (e.g. if input parameters had incorrect types like a string representation of an integer value). This would happen transparently to the initiator of the call which would only have to be involved if the gateway agent was unable to find a solution to the problem.

Using proactive behaviour the agent can also support the execution of Cloud workflows. For example, it could restart workflows if the returned result strongly deviated from expected results. Or it could prepare or even already initiate recurring net executions.

The gateway agent can also handle quality of service (QoS) concerns. As stated in 3, QoS are specified as parameters either in transitions or places. The second scenario is the more appropriate since it use synchronous channels. In this situation, in addition to the workflow model (and its related files .rnw and .sns) modellers also include QoS parameters. In this work, we focus on time and budget, but modellers can include other constraints. The gateway agent can consequently play another role, which is Cloud brokering. By brokering we mean that the agent looks for the suitable Cloud provider to execute the workflow based on its requirements. This can be useful when working with multiple Clouds.

One disadvantage of using a gateway agent for the Cloud is that it centralises the communication. This decouples the communication aspects from the individual agents, but gives the system a single point of failure. Only one agent in the system, the gateway agent, possesses the functionality and mechanisms to invoke Cloud systems. This makes other agents simpler and possibly more efficient to execute, but if the gateway agent fails communication with the Cloud is lost. This could be remedied by implementing a solution with multiple gateway agents and distributing the functionality. If one gateway agent failed others could take its place.

Entity Interface The term entity describes a hybrid construct between an agent and a workflow. Depending on the runtime needs they can act as an agent (e.g. for communication), a workflow (e.g. for task deployment and execution) or something between the two (e.g. as a mobile process). Entities and modelling with them is currently ongoing research. The Cloud context enhances the capabilities of entities in many regards.

From the interface point of view an entity possesses all the characteristics of agents and has access to the entire functionality described in the previous paragraph for an advanced interface provided through an agent. But this interface is extended even more because of the additional possibilities gained through the workflow properties of an entity. Entities are, in one perspective, a (workflow) process. This automatically entails a certain behaviour-centric structure and purpose to the modelling.

By structuring the calls and instantiations of the Cloud net systems as a process itself the modeller is directly supported. While anything can be achieved through regular, less-rigidly structured modelling, restricting the modeller into such a process perspective is still beneficial. Considering process order, task subdivisions, processing of partial results and other aspects of a process are direct requirements in this perspective. Consequently they are obligatory to the modeller here. But that means that these aspects, which range from helpful to essential, can also not be ignored or omitted. This is what the entities add on a conceptual level to the advanced interface of agents.

5 Discussion

One issue that was raised in Section 4 concerned the restrictions on modelling and the placement of input and output in a net for the interfaces. If that placement is unrestricted it may be error-prone and puts the responsibility solely on the modeller without any support. An effort could be made to restrict input and output to the initial and exit places of the workflow. This would ensure only full results are returned to the caller and make it easier to verify workflow net properties. However, there are cases in which partial results (e.g. status updates) *during* the execution of a workflow net are desirable. The restriction would preclude this. A compromise would be to allow simple status reports from anywhere in the net (e.g. via the console), but only complete results from the final place or transition of the workflow (e.g. via synchronous channel). Only these complete results would then be made available for further operations in the workflow initiator.

Without any restrictions it would also be impossible to make any statements about the correctness of the executed workflows. For practical purposes allowing input into already running workflows and arbitrary input/output locations might be helpful to some use-cases. But from a verification and validation point-of-view these mechanisms are problematic. Incorporating concepts like workflow correctness into the Cloud calls and interfaces is currently ongoing work but outside the scope of this paper.

The question of restrictions raises another interesting point. This paper is focused on the execution of workflow nets. Arbitrary workflow net systems can be executed in the Cloud. That includes scientific and interorganisational workflows.

From a technical standpoint though, it is possible to execute any net system in the Cloud. The only precondition is that a plugin for the net formalism in question is provided for the Renew instance running in the Cloud. Renew plugins for many formalisms already exist (e.g. P/T nets, nets supporting time annotations) and more can be added.

When allowing arbitrary net systems without restrictions to the interface or without any structured modelling these arbitrary net systems might pose challenging to modellers in terms of efficiency and manageability. For this reason it is advisable to use structured modelling paradigms, like agents or entities, for the Cloud net systems as well. In the following paragraphs we will examine how this would affect the advanced interfaces described in the previous section.

By executing the agent interface within the Cloud (as opposed to outside the Cloud as described in Section 4.3) the communication can be simplified. In this scenario the net system executed in the Cloud is a Capa agent platform with a running gateway agent. The gateway agent is accessible for other agents via the standardised FIPA compliant asynchronous message communication supported in Capa. This would "move" the interface from the local execution into the Cloud, since to other agents it does not matter where the gateway is executed. They communicate with him in the same way as any other local or remote agent. This would lead to efficiency gains as the gateway agent could access resources in the Cloud environment directly. The technical capabilities of the gateway agent would also be improved. Other properties of the interface would largely remain the same.

The entity interface would benefit in the same way as the agent interface. In addition it would also affect the modelling abstraction of the entity, as it could be considered a (workflow) process in the Cloud executing other (workflow) processes. This is especially interesting in the interorganisational workflow setting which we are researching for entities. The entity in the Cloud could be considered as the overall interorganisational workflow while the workflows it controls are the subworkflows for each involved organisation.

6 Conclusion

In this paper we presented our approach for moving net executions to the Cloud. The paper described the technical aspects, implementation and methodology. From a technical point of view it is possible to execute any net system supported by Renew in the Cloud. However, for the purpose of this paper we focused on workflows. For this context the notion of Cloud interfaces was introduced. These interfaces can be classified as simple, simulation and advanced depending on how the communication and the transfer of data are performed. Furthermore, we discussed the integration of agent concepts in order to provide gateways to the Cloud.

Direct future work is related to agents and especially the entity concept. This paper described how agent and the entity concepts can realise advanced interfaces for the Cloud net systems. The other direction is currently also being researched. In general, opening up the capabilities of entities to Cloud functions is already beneficial in of itself. But agents and especially entities can also feature very complex behaviour. In fact, some processes of entities can be regarded as fully-fledged subsystems. Relocating these subsystems to the Cloud can improve the performance of entity systems greatly.

Concerning workflow complexity, we are also currently working on a concrete scientific workflow application. This application is related to the remote sensing domain, especially image processing of satellite imagery. Most of the work has been achieved: we have implemented an image processing tool that allows modelling and execution of remote sensing applications specified by reference nets. The next natural step is to execute those workflows in the Cloud based on the results presented in this paper. Furthermore, this work should be evaluated in terms of performance. This concerns running several simulations in parallel (in different virtual machines) in the Cloud.

In conclusion, the realisation of Renew in a Cloud opens up a number of advantages w.r.t. performance, availability, flexibility, etc. Some of these have already been discussed in this paper. Other will become more noticeable with the ongoing work. The continued incorporation of the Cloud aspects with complex workflow and agent systems is just one of the possible avenues of thought, albeit the most promising one currently.

References

1. Martin Alt, Sergei Gorlatch, Andreas Hoheisel, and Hans-Werner Pohl. A Grid Workflow Language Using High-Level Petri Nets. In Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *LNCS*, pages 715–722. Springer-Verlag, 2006.
2. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424, June 2004.
3. Sofiane Bendoukha and Lawrence Cabac. Cloud transition for QoS modeling of inter-organizational workflows. In Daniel Moldt, editor, *Modeling and Buisness Environments MODBE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 355–356. CEUR-WS.org, June 2013.
4. Tobias Betz, Lawrence Cabac, Michael Duvigneau, Thomas Wagner, and Matthias Wester-Ebbinghaus. Software Engineering with Petri Nets: A Web Service and Agent Perspective. In Maciej Koutny, Serge Haddad, and Alex Yakovlev, editors, *Transactions on Petri Nets and Other Models of Concurrency IX*, Lecture Notes in Computer Science, pages 41–61. Springer Berlin Heidelberg, 2014.
5. Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak, and Sebastian Wagner. OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13)*, volume 8274 of *LNCS*, pages 692–695. Springer Berlin Heidelberg, December 2013.

6. Lawrence Cabac. Multi-agent system: A guiding metaphor for the organization of software development projects. In Paolo Petta, editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 1–12, Leipzig, Germany, 2007. Springer-Verlag.
7. Søren Christensen and N. D. Hansen. Coloured Petri Nets Extended with Channels for Synchronous Communication. In Valette, R., editor, *Lecture Notes in Computer Science; Application and Theory of Petri Nets 1994, Proceedings 15th International Conference, Zaragoza, Spain*, volume 815, pages 159–178. Springer-Verlag, 1994.
8. Peter Dadam and Manfred Reichert. The adept project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development*, 23(2):81–97, 2009.
9. Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, July 2005.
10. Michael Duvigneau. Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, December 2002.
11. T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wieczorek. Askalon: a grid application development and computing environment. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 10 pp.–, Nov 2005.
12. Christina Hoffa, Gaurang Mehta, Timothy Freeman, Ewa Deelman, Kate Keahey, G. Bruce Berriman, and John Good. On the use of cloud computing for scientific workflows. In *eScience*, pages 640–645. IEEE Computer Society, 2008.
13. G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, and P. Maechling. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59 –66, dec. 2009.
14. Olaf Kummer. *Referenznetze.* Logos Verlag, Berlin, 2002.
15. Olaf Kummer, Frank Wienberg, Michael Duvigneau, and Lawrence Cabac. *Renew – User Guide (Release 2.4).* University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg, April 2013. Available at: http://www.renew.de/.
16. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Michael Köhler, Daniel Moldt, and Heiko Rölke. Renew – the Reference Net Workshop. In Eric Veerbeek, editor, *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003).*, pages 99–102, June 2003.
17. A. Nagavaram, G. Agrawal, M.A. Freitas, K.H. Telu, G. Mehta, R.G. Mayani, and E. Deelman. A cloud-based dynamic workflow for mass spectrometry data analysis. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 47–54, Dec 2011.
18. Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Tim Carver, Matthew R. Pocock, and Anil Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20:2004, 2004.
19. Tim Grance Peter Mell. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory, 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.
20. Manfred Reichert, Thomas Bauer, and Peter Dadam. Flexibility for distributed workflows. In Minhong Wang and Sun Zhaohao, editors, *Handbook of Research on*

Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments, pages 137–171. IGI Global, Hershey, New York, July 2009.

21. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
22. Jörn Schumacher. Eine Plugin-Architektur für Renew – Konzepte, Methoden, Umsetzung. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, October 2003.
23. Ian Taylor, Matthew Shields, Ian Wang, and Omer Rana. Triana applications within grid computing and peer to peer environments. *Journal of Grid Computing*, 1(2):199–217, 2003.
24. W.M.P. van der Aalst. Business process configuration in the cloud: How to support and analyze multi-tenant processes? In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 3–10, Sept 2011.
25. Thomas Wagner. A centralized Petri net- and agent-based workflow management system. In Michael Duvigneau and Daniel Moldt, editors, *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, number FBI-HH-B-290/09 in Bericht, pages 29–44, Vogt-Kölln Str. 30, D-22527 Hamburg, September 2009. University of Hamburg, Department of Informatics.