

On the Use of Reference Nets for Building Cloud-based Scientific Workflows

Hayat Bendoukha¹, Sofiane Bendoukha², and Yahya Slimani³

¹ Department of Computer Science
Faculty of Mathematics and Computer Science
Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf, USTO-MB,
BP 1505, El M'naouer, 31000 Oran Algérie
`bendoukhat@univ-usto.dz`

² Theoretical Foundations of Computer Science (TGI)
Department of Informatics, University of Hamburg, Germany
`sbendoukha@informatik.uni-hamburg.de`

³ Dpartement Informatique Institut Supérieur des Arts Multimédia (ISAMM)
Université de la Manouba
Campus Universitaire, 2010 Manouba, Tunisie
`yahya.slimani@fst.rnu.tn`

Abstract. Cloud computing provides scientists with a large number of powerful resources. These resources enhance the productivity of the whole system. Running complex scientific workflows on Cloud resources rather than on-premise increases the performance of execution. Nevertheless, traditional scientific workflow management (SWfM) systems are not yet adapted for the Cloud. Creation of scientific workflows and their execution in the Cloud still face many challenges due to the complexity of the Cloud environment. Migrating a part or the whole scientific application to the Cloud is not trivial. It should be based on a solid strategy. Questions like: Where to store the data and where to execute the processes need to be investigated. In this paper, we first present RENEWGRASS, a tool for modeling and executing image processing workflows by *reference nets*. Then, we discuss the deployment of RENEWGRASS into the Cloud. This work includes also a use case example related to the remote sensing domain.

Keywords: Workflows, RENEWGRASS, Petri Nets, Cloud computing, Image processing

1 Introduction

Scientific workflows are a special class of workflows, which are characterized as large-scale, long-running and resource-intensive [20]. The major goal of scientific workflows is to allow scientists to focus on domain-specific (science) aspects of their work, rather than dealing with complex data management and software issues. How the sequence of the workflow tasks is represented can be handled in different ways. The sequence can be specified either by scripting languages

or through graph-based techniques such as Petri nets or π -calculus [22] [19]. Scripting languages are usually based on markup languages such as Extensible Markup Language (XML). They may be convenient for well skilled users and do not need to be converted to be run on a cloud environment. But, they are not user-oriented and do not permit to specify large and complex workflows manually [3]. In this work, Petri nets and more specifically *reference nets* [15] as a modeling technique suitable to model the workflow patterns in an elegant and easy way [4].

Nevertheless, due to the large amount of data and tasks, that need to be processed, the execution of such kind of workflows requires often to be mapped into external resources. During these a few years, Cloud computing is growing considerably and it is gaining popularity in both academia and industry. We believe that this technology is the suitable solution for the execution of scientific workflows. The reason is the features that clouds provide in terms of the powerful resources, which range from storage, computing and networks. Unfortunately, existing SWfM systems are not adapted to perform in the Cloud. They need to fit to the Cloud architecture. They also need to provide modeling means and migration mechanisms that enable a full integration between workflow concepts and Cloud technology.

The objective of the current work is twofold. First, we implemented a tool named RENEWGRASS for the specification and the execution of scientific workflows. Then, we, successfully, integrated RENEWGRASS in the **RE**ference **NE**ts **W**orkshop (RENEW) which is available at (www.renew.de), our chosen modeling and simulation tool for Petri nets. As a domain of application, the tool is suitable for remote sensing especially processing of satellite imagery. Modeling and implementing such kind of workflows need specific tools and techniques, which are unfortunately not provided by RENEW. Technically, the integration of RENEWGRASS consists on extending RENEW by modules and components of the Geographic Resource Analysis Support System (GRASS) GIS (Geographic Information System). This allows users to invoke Grass GIS modules directly from their Petri net models, which can be later executed. With the integration of RENEWGRASS into RENEW, the latter is now able to deal with other research domains such as the scientific domain. Moreover, as soon as the workflow requirements become locally unsatisfied, workflow tasks need to be mapped to distributed resources. With respect to the work presented in [5] and as an extension, this issue is also taken into account, since our long-term perspective is to provide a service-oriented environment built on top of Cloud resources and to allow flexible deployment of scientific workflows. Therefore, we will discuss the deployment of RENEWGRASS into the Cloud. Questions like: Where to store the data? Where to execute the activities? will be investigated. Here, we propose an agent-based architecture, where each functionality of the system is performed by a specific agent. In our approach, we follow the PETRI NET-BASED, AGENT-ORIENTED SOFTWARE ENGINEERING (PAOSE) paradigm for developing agent-based applications.

The rest of the paper is organized as follows. Related work as well as the conceptual and technical background of this work are presented in Section. 2. RENEWGRASS is described in Section. 3. How RENEWGRASS can be deployed in the Cloud is investigated in Section. 4. Section. 5 discusses . Section. 6 concludes the paper with summary and future works.

2 Related Work and Background

It is noteworthy to introduce some concepts, techniques and tools that constitute the conceptual and functional background of this work. These are essential to understand how the contributions work.

2.1 Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). Cloud computing can be viewed as a collection of services, which can be presented as a set of loosely-coupled layers [17]. These service models are closely related and can be seen as three layers, which are the Infrastructure as a Service (IaaS), the Software as a Service (SaaS) and the Platform as a Service (PaaS). Before the emergence of the Cloud technology, there were significant research projects dealing with the development of distributed and scientific workflow systems with the grid paradigm. Workflow enactment service can be built on top of the low level grid middleware (eg. Globus Toolkit⁴, UNICORE⁵, EGI⁶ and Alchemi⁷), through which the workflow management system invokes services provided by grid resources [23]. At both the build-time and run-time phases, the state of the resources and applications can be retrieved from grid information services. There are many grid workflow management systems in use; like these representative projects: ASKALON⁸, Pegasus⁹, Taverna¹⁰, Kepler¹¹, Triana¹² and Swift [24]. Most of these projects, have been investigating the adaptation of their architectures to include the Cloud technology. For instance, the Elastic Compute Cloud (EC2) module has been implemented to make Kepler supports Amazon Cloud services. Launched in 2012, the Amazon Simple Workflow (SWF)¹³ is an orchestration service for building scalable applications. It maintains the execution state of the

⁴ <http://www.globus.org/toolkit/>

⁵ <http://www.unicore.eu/>

⁶ <https://www.egi.eu/>

⁷ <http://www.cloudbus.org/~alchemi/>

⁸ <http://www.askalon.org/>

⁹ <http://www.pegasus.org/>

¹⁰ <http://www.taverna.org.uk/>

¹¹ <https://kepler-project.org/>

¹² <http://www.trianacode.org/>

¹³ aws.amazon.com/swf

workflow in terms of consistency and reliability. It permits structuring the various processing steps in an application running on one or more systems as a set of tasks. These systems can be Cloud-based, on-premise, or both. But they lack of use of standard specification tools and notations. Contrarily to Petri-nets, The specification tolls of these present also the disadvantage of missing a verifying tool.

2.2 Reference Nets

As stated above, we use Petri nets and more specifically *reference nets* [15]. The latter extend the colored Petri net formalism by combining the concepts of synchronous channels. Reference nets are the implementation of the concept of *nets-within-nets* [21], which allows tokens to be nets again. With *reference nets*, Petri nets are not only useful for modeling and analyzing of systems but also for implementation. Their advantage is that the model is transformed into an implementation without changing the formalism. Thus, the gap between modeling and implementation is diminished [9]. Reference nets are object-oriented high-level Petri nets and are based on the nets-within-nets formalism introduced by [21], which allows tokens to be nets again. They extend Petri nets with dynamic net instances, net references, and dynamic transition synchronization through synchronous channels. Reference nets consist of places, transitions and arcs. The input and output arcs have a similar behavior to ordinary Petri nets. Tokens can be available of any type in the Java programming language. In opposite to the net elements of P/T nets, reference nets provide supplementary elements that increase the modeling power. These elements are: virtual places, declaration and arc types. The places are typed and the transitions can hold expressions, actions, guards, etc. Firing a transition can also create a new instance of a subnet. The creation of the instances is similar to object instances in object-oriented programming. This allows a specific, hierarchical nesting of networks, which is helpful for building complex systems.

The nets shown in Fig. 1 are to illustrate two important features of reference nets, which are the notion of *synchronous channels* and net inscription (Java). These two features are used frequently in this work. The figure shows the modeling of a simple Cloud-based storage workflow. The model is composed of two different nets that need to communicate with each other in order to store files in the Cloud using a Cloud service (DropBox). The net (a) represents the Web authentication step to the DropBox service¹⁴. It consists on setting the user credentials up. This is performed by creating an instance of the Java class *DropTransition* and checking the credentials by calling the method *connect()*. If the above step succeeds, then a URL¹⁵ is generated. After completion, Net (a) creates a new instance of Net (b), which will use the generated information from Net (a) to upload files to the repository. This is performed by using synchronous channels (down and up-links).

¹⁴ <https://www.dropbox.com/>

¹⁵ The generated url is the IP address of the authentication page of the DropBox service.

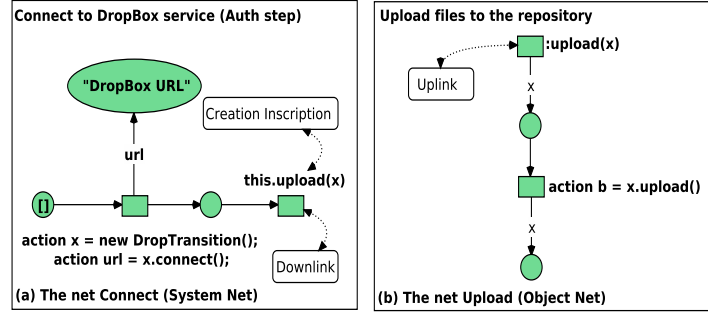


Fig. 1. (a) The System Net, (b) The Sub-net Upload

2.3 Renew

The **RE**ference **NE**ts **W**orkshop (RENEW), which is available at (<http://www.renew.de/>) is a graphical tool for creating, editing and simulating reference nets. It combines the *nets-within-nets* paradigm with the implementing power of Java. With RENEW it is possible to draw and simulate both Petri nets and reference nets. During the simulation, a net instance is created and can be viewed in a separate window as its active transitions fire. Simulation is used in Renew to view firing sequences of active transitions in reference nets. Simulation can run in a one step modus where users can progress in steps where only one transition fires. RENEW also offers the possibility to set breakpoints to hold the simulation process. Breakpoints can be set to places and transitions. By changing the compiler, RENEW can also simulate P/T nets, timed petri nets, Workflow nets, etc.

RENEW is an editor as well as a simulator for Petri nets and *reference nets*. Since the version 1.7, RENEW is built on a highly sophisticated plug-in architecture, which was developed and introduced in [18]. It allows the extension of RENEW with additional functionality through the use of interfaces from RENEW components without changing the core of RENEW.

2.4 The Grass GIS

Grass is a multi-purpose open source GIS, which can be used for geoprocessing applications such as: geospatial data production, analysis and mapping. It can handle raster as well vector data. The most important specificity of the Grass GIS is its modularity, which diminishes overhead. This allows to run only the required modules (same as RENEW). These modules are organized in categories (general GIS modules, raster modules, vector modules, etc.). Table. 1 shows the modules provided by the Grass GIS [16].

In order to allow RENEW executing these commands directly from the Petri net transitions, RENEWGRASS offers a wrapping layer, which makes the Grass modules available at runtime. Before using the modules for processing the data,

Table 1. Grass GIS Commands (from [16])

Prefix	Function class	Type of command
d.*	display	graphical output
db.*	database	database management
g.*	general	general file operations
i.*	imagery	image processing
m.*	misc	miscellaneous commands
ps.*	postscript	map creation in Postscript format
r.*	raster	2D raster data processing
r3.*	3D raster	3D raster data processing
v.*	vector	2D and 3D vector data processing

the latter should be first imported into a Grass *DATABASE*.

Within the *DATABASE*, the projects are organized as subdirectories called "LOCATIONS". Each *LOCATION* can have one or more *MAPSETS*. Each *MAPSET* may represent a sub region within a given *LOCATION*. These are mostly the important variables that need to be set.

3 RenewGrass

In this section, we show how the Grass GIS has been integrated in RENEW. First, a short overview of the Grass GIS is given. This includes the Grass modules as well as the required structure of a GIS project. Next, we introduce the architecture to show all the components taking part to the integration.

3.1 Integration Issues

The first obstacle we faced when trying to integrate the Grass GIS with RENEW is that these tools are written in different programming languages. (The Grass GIS is written in C and RENEW in Java), which makes a direct communication between them arduous. Thus the Grass GIS needs to be adapted to the running environment of RENEW. Moreover, a proper environment variables need to be pre-specified. In order to achieve this integration, there are three possibilities:

- *Desktop integration*: this means that the Grass GIS is locally deployed and interfaces are provided to use geoprocessing functions from RENEW.
- *Web-based integration*: in this case, the objective is to publish and execute geo-processes over the web, following the Web Processing Service (WPS) interface specification¹⁶ from the Open Geospatial Consortium (OGC)¹⁷.

¹⁶ <http://www.opengeospatial.org/standards/wps>

¹⁷ <http://www.opengeospatial.org/>

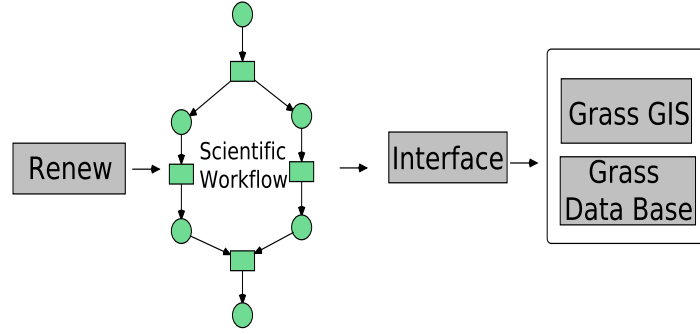


Fig. 2. Grass GIS Integration with RENEW

- *Remote execution (Vagrant)*: this is another alternative to deploy the Grass GIS into the Cloud and is discussed in Section. 5.

Fig. 2 is a simple overview of the integration of Grass GIS as desktop application. With RENEW scientific workflows are specified as Petri net models. When some of workflow tasks require Grass commands, this can be easily performed directly at the transitions. In order to communicate directly with the Grass GIS, an interface or a wrapper is necessary.

Over the last few years some effort has been dedicated to leverage the strength of Java and Grass GIS. There are some contributions dealing with this issue [14]. Two candidate projects are interesting for our work, which are the *JGrasstools*¹⁸ and the *vtkGRASSBridge*¹⁹. The *vtkGRASSBridge* provides a VTK/C++ interface to most of the GIS GRASS raster, voxel and vector C library functions. This library can be used to build comprehensive 3D visualisation of GIS GRASS data with Java, Python and C++. Although the project seems promising, it was quickly rejected, due to building issues. We realized, that future users of the tool will certainly run into similar issues, when modeling and executing workflows with RENEWGRASS if we followed this approach. For this work, we chose to take advantages from the JGrasstools project. JGrasstools is a library that is extracted from the Java Geographic Resources Analysis Support System (JGrass) project.²⁰

3.2 Architecture

As mentioned above, RENEW's architecture has been decomposed into several components. Each component is characterized as a plug-in. This provides more flexibility and extensibility. Thus new features can be easily integrated. The basis

¹⁸ <http://moovida.github.io/jgrasstools/>

¹⁹ <https://code.google.com/p/vtk-grass-bridge/>

²⁰ JGrass is a free, multi platform, open source GIS based on the GIS framework (see www.jgrass.org).

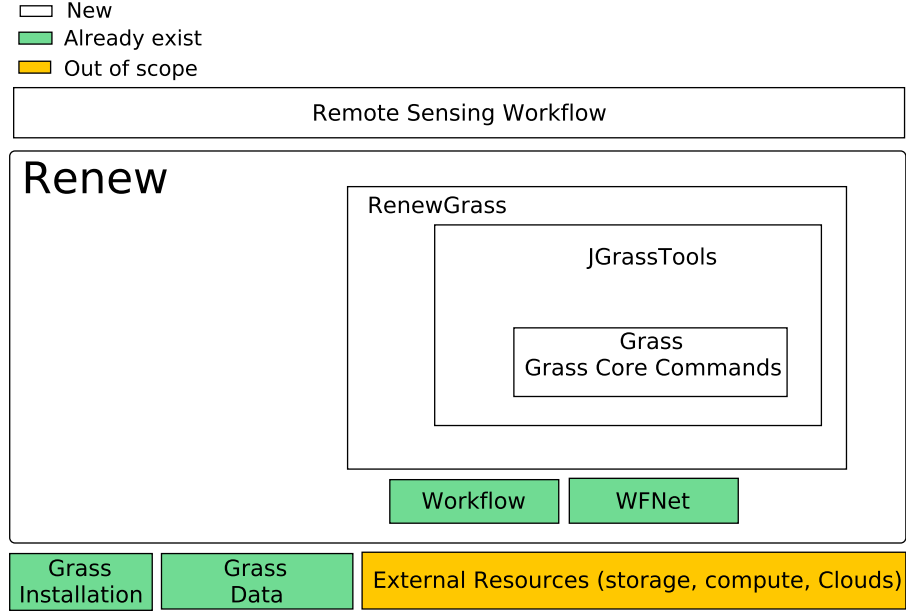


Fig. 3. Architecture of the RENEWGRASS Tool

for this is the RENEW plug-in system [10], which is responsible for the addition and the removal of plug-ins at runtime.

New features include some new plug-ins as for instance the *Workflow* and the *WFNet*, which provide workflow management functionality. The original functionality was already presented in [13]. The RENEWGRASS tool is also built following the plug-in architecture of RENEW. Fig. 3 shows a simplified view of the position of RENEWGRASS in RENEW.

The *Workflow* and the *WFNet* plug-ins are not required when using RENEWGRASS. They are used especially when the users want to integrate workflow management functionality such as log-in, tasks management, etc. As you can see in the figure, the RENEWGRASS plug-in is built on top of the JGrasstools, which was adapted for RENEW. The main requirements are the Grass GIS installation and the Grass data. The first one provides all the modules presented in Section. 2.4. The Grass data is a directory, that holds all the required files (raster or vector images). Both Grass GIS installation and the Grass Data path should be specified to RENEWGRASS prior any utilization. The actual implementation of the tool allows local execution only, since both RENEW and Grass GIS are installed on-premise. The use of external resources, mainly Cloud computing resources, is not excluded, but it is out of scope of this paper. RENEWGRASS is freely available via www.paose.net and <http://sofiane.github.io/>.

4 RenewGrass in the Cloud

RENEWGRASS has been successfully tested in a local environment. All the required components were hosted on-premise (RENEW, Grass GIS and the data). Nevertheless, as soon as the number of tasks increases and the size of data become large, we start facing computing and storage issues. This is due to insufficient resources on the local site. This section presents our vision to deploy the current implementation of the RENEWGRASS tool onto the Cloud services. First, different possibilities to Cloud-enable an application in general are shortly illustrated. These possibilities are formulated in form of patterns. The illustration is based on the work of [12] and [1]. They investigate and strive to answer questions like: where to enact the processes? Where to execute the activities? and Where to store the data? Thus the entities taken in account are: (1) the process engine (responsible for the execution and the monitoring of the activities) (2) the activities that need to be executed by the workflow and (3) the Data. Next, we propose an architecture to introduce a new pattern and an appropriate methodology to enable remote execution in the Cloud. This has been also described in a previous work [6].

4.1 Migration Patterns

Moving an existing application to the Cloud should be based on a solid strategy. Providing the business management system (or WfMS) or a part in the Cloud raises a series of concerns about ensuring the security of the data and the performance of the system. For example, Cloud users could lose control on their own data in case of a fully Cloud-based solution. Some activities, which are not compute-intensive can be executed on-premise rather than moving them to the Cloud. Unfortunately, this transfer can be time and cost-consuming because of the pay-per-use model and the nature of the workflow tasks.

In the following, the patterns from [12, 1] are shortly introduced. The first pattern designs the traditional scenario where all the components of the workflow system are hosted at the user-side (on-premise). The second scenario represents a case when users already have a workflow engine but the application contains compute or data-intensive activities, so they are moved to the Cloud for acquiring more capabilities and better performance. The third case designs a situation, where the end-users do not have a workflow engine, so they use a Cloud-based workflow engine, which is provided on-demand. In that case, workflow designers can specify transfer requirements of activity execution and data storage, for example, sensitive data and non-compute-intensive activities can be hosted on-premise, and compute-intensive activities and non-sensitive data can be moved to the Cloud [12]. The last scenario presents a situation where all components are hosted in a Cloud and accessed probably from a Web interface. The advantage is that users do not need to install and configure any software on the user side. To make an analogy with the elements of our approach, RENEW is the process engine, the activities are the geoprocessing tasks (performed by the Grass services), which are related to the satellite images (data). The latter (data

Local Execution	Cloud Execution
(1) Traditional WfMS Data Grass services Renew	
(2) Local WfMS with cloud distribution Data Grass services Renew	Data Grass services
Data Grass services	(3) cloud execution & local distribution Data Grass services Renew
	(4) cloud WfMS Data Grass services Renew

Fig. 4. Patterns for Cloud-based Workflow Systems (Adapted from [12])

Local	Cloud
Data Grass services Renew A	Data Grass services Renew B

Fig. 5. A Pattern Designing Multiple Process Engines Integration

and Grass GIS) can be either on-premise or hosted in the Cloud. Based on these elements and the illustration presented above, Fig. 4 presents an overview of the diverse approaches (patterns) to design our workflow system based on the Cloud technology.

We have noticed that both [12] and [1] do not address all possible situations. For instance, the following situation has been not addressed: the process engine is available on the user side but due to circumstances (internal failure, not sufficient compute or storage resources), remote process engines need to be integrated and remotely invoked. Fig. 5 shows approximately how this scenario looks like. Our solution consists on transferring the data and executing the process by another process engine. This is discussed in the following sections.

4.2 Architecture

Fig. 6 shows the architecture to integrate the current implementation into a Cloud system. While RENEWGRASS is already implemented and successfully integrated in RENEW (see section 3), most efforts are dedicated now to move the execution of the geoprocessing tasks to the Cloud and the provision of an interface to invoke these services directly from workflow models.

In our work, we follow an agent-based approach, i.e., many components functionalities are performed by special agents. In summary, the role of each agent used in the approach is described below.

1. *Workflow Holder Agent*: The Workflow holder is the entity that specifies the workflow and in consequence holds the generated Petri net models. This entity can be either human or a software component. The specification of the image processing workflow is performed using RENEWGRASS, which provides a modeling palette or downright predefined modeling blocks.
2. *Cloud Portal Agent*: provides the *Workflow Holder Agent* a Web portal as a primary interface to the whole system. It contains two components: *Cloud Manager* and *Workflow Submission Interface*. The latter provides a Web interface to the workflow holders to upload all necessary files to execute the workflow. This includes the workflow specification (RENEW formats²¹), input files (images). It also serves getting notifications from the *Cloud Broker Agent* about the status of the workflow or the availability of the Cloud provider. The role of the *Cloud Manager* is to control the Cloud instances (start and stop or suspend).
3. *Cloud Broker Agent*: It is a critical component of the architecture, since it is responsible of (i) the evaluation and selection of the Cloud providers that fits the workflow's requirements (e.g., data volume and computing intensities) and (ii) mapping the workflow tasks. Both activities require information about the Cloud provider, which are available and provided by the Cloud Repository Agent.
4. *Cloud Repository Agent*: The Cloud repository register the information about the Cloud providers and the state of their services. These information are saved in a database and are constantly updated, since they are required by the Cloud Broker Agent. To avoid failure scenarios (repository down, loss of data), we use distributed databases, which allows high availability and fault-tolerant persistence.
5. *Cloud Provider Agent*: The role of this agent is to control the instances and to manage the execution of the tasks. Regularly, the Cloud providers need to update their status and send it to the Cloud Repository Agent. The status concerns both the instance and the services (Grass services).

Concerning the *Cloud Broker Agent*, the evaluation and the selection of the Cloud providers are critical processes for the *Workflow Holders*. In Cloud computing there are various factors impacting the Cloud provider evaluation and selection [11, 2] such as: computational capacity, IT security and privacy, reliability and trustworthiness, customization degree and flexibility/scalability, manageability/usability and customer service, geolocations of Cloud infrastructures. For this, in our work, brokering factors are limited to the computational capacity and the customization degree.

²¹ RENEW supports various file formats saving (XML, .rnw, .sns, etc.)

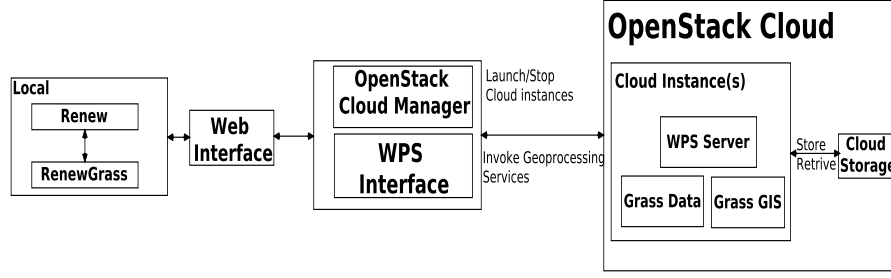


Fig. 6. The Architecture of the Cloud-based Workflow System

4.3 Cloud Configuration

Concerning the customization degree, there are some requirements for a successful deployment onto the Cloud. In general, a common procedure to deploy applications onto Cloud services consists of these two main steps:

1. *set up the environment*: this mainly consists of the provision of Cloud instances²² and the configuration the required softwares properly. Essential are the environment variables, which differ from the local implementation such as JAVA configuration for the Web server.
2. *deploy the application*: it consists of the customization of the Cloud instance with the appropriate softwares. For our work, RENEW and the Grass GIS should be correctly and properly configured, especially the database and the installation path.

Furthermore, the Grass commands can be invoked in different ways. Either through a wrapper like in the original implementation of RENEWGRASS or provided as Web services. For the latter, we follow a Web-based approach with respect to the Open Geospatial Consortium (OGC) Web Processing Service (WPS) interface specification. Thus the Grass GIS functionalities are provided as Web services instead of desktop application. To achieve this, we chose the 52North²³ as a WPS server as well as the wps-grass-bridge²⁴.

4.4 Execution Scenario

Considering the proposed architecture and the agent roles described above, a typical deployment scenario is broken into the following steps:

1. Workflow holders specify their image processing workflows (data and control-flow) using Petri nets for example the NDVI workflow (see section 3).

²² The Cloud instances should be in priori customized, i.e, they need to have the Grass GIS as back-end, the WPS server and RENEW. We assume that the Cloud storage is a service, which is configured by the Cloud provider itself.

²³ <http://52north.org/>

²⁴ <https://code.google.com/p/wps-grass-bridge/>

2. They send a request to the *Cloud Broker* via the *Cloud Portal*.
3. The *Cloud Broker* checks for available Cloud providers, which provide geo-processing tools (Grass GIS). This information is retrieved from the *Cloud Repository*.
4. The *Cloud Broker* sends a list to the *Workflow Holder* (through the Cloud Portal) to accept or to reject the offer.
5. If the offer is accepted, the *Workflow Holder* submits the workflow specification (.rnw + .sns) to the selected *Cloud Provider*.
6. Launch a customized Cloud instance with RENEW and Grass GIS running in the background.
7. After simulation/execution of the workflow, results (in our prototype it consists of calculating the NDVI value) are transmitted to the *Workflow Holder* through the *Cloud Portal*.

Rejecting an offer does not conclude the execution process immediately. Since the list transmitted by the *Cloud Broker* is updated constantly, it might be that new Cloud providers are available and fits the requirements. Therefore, from step (3), the process is iterative until the satisfaction of the *Workflow Holder*. Regarding step (5) and (6), RENEW supports starting a simulation from the command line. This is possible by using the command *startsimulation (net system) (primary net) [-i]*. The parameters to this command have the following meaning:

- *net system*: The .sns file.
- *primary net*: The name of the net, of which a net instance shall be opened when the simulation starts.
- *-i*: If you set this optional flag, then the simulation is initialized only, that is, the primary net instance is opened, but the simulation is not started automatically.

5 Discussion

In the previous sections we presented RENEWGRASS and its integration in the modeling and simulation tool RENEW. We explained that there are many possibilities to deploy the tool. Either on-premise or in the Cloud. For now RENEWGRASS has been successfully deployed in local environment following *Desktop integration* (see Section. 3). Grass GIS modules can be easily invoked from Petri net models. Concerning the deployment of RENEWGRASS into the Cloud, the proposed solution involves the provision of Grass functionalities as Web services using the WPS specification (see Section. 4). Here we discuss another alternative to deploy RENEWGRASS into the Cloud. Our solution consists of creating customized Cloud instances, which includes RENEW and the Grass GIS. Thus all activities are performed in the Cloud. The idea is that Cloud customers have the possibility to specify image processing workflows using Petri nets. The latter are then pushed to the Cloud, where they are executed/simulated. This scenario is summarized in Fig. 7. Our concept is based on enabling RENEW simulations in

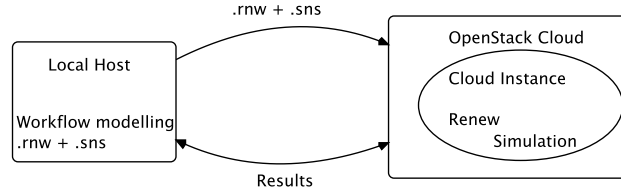


Fig. 7. RENEW Simulation in the Cloud (from [7])

the Cloud. We already provide mechanisms to Cloud customers to create Cloud instances and provision them by Java, RENEW and other softwares that they need for running their applications [7].

Unfortunately, the current solution raises other interesting issues that need more investigation. For example, our proposition is based on the fact that workflows are executed once as a single process. There is at the moment no mechanism to control the simulation of the workflow in the Cloud. We mean by *control*: the management of firing the Petri nets transitions in the Cloud instance. RENEW already provides the possibility to control the simulation by specific commands like: *simulation run*, *simulation halt* or *simulation stop*. Nevertheless, it is not possible to use these functionalities within a Cloud instance. This should be in priori customized. What about the ability to decompose the workflow in order enable selected execution of the tasks composing the workflow? For this purpose we are investigating other alternatives as the remote execution of RENEW. One of these solutions have been already discussed in [7]. In the latter work we provide mechanisms and implementation for moving the execution of computation- and time- consuming workflows into the Cloud. Different kind of interfaces are provided to enable remote execution of RENEW in the Cloud. These interfaces define how input and output to the Cloud calls are defined. They range from simple, simulation and complex.

Interesting for the current work are the complex interfaces. Through such kind of interface we seek bringing intelligence and autonomy to the managing system. Concretely, special *agents* are used as *gateways* between the workflow (Petri net) model and the Cloud. With respect to the MULAN/CAPA framework, there exists a *WebGateway Agent* [8]. It plays the role of a gateway between RENEW and the Web environment. Since Grass GIS commands can be also published as Web services (see Section. 4) coupling the WebGateway functionality into the architecture proposed in Section. 4 will certainly enhance building agent-based scientific workflows.

6 Conclusion

The objective of the work presented in this paper is twofold. On the one side, we provide techniques and tools to support scientists building their applications. For

this purpose, a geoprocessing tool named RENEWGRASS is presented. The tool has been implemented and successfully integrated in the modeling and simulation tool RENEW. The application domain of RENEWGRASS is the remote sensing, especially image processing, a kind of scientific workflows. Therefore, we afford scientists with a palette of processing functionalities based on the Grass GIS. Furthermore, we discuss the extension of the current work by the integration of the Cloud technology. For this purpose, we introduce migration patterns and introduced our architecture for the deployment of workflows onto Cloud providers. The natural next step is to concertize the deployment mechanisms introduced in Section 4. This means concretely to implement the functionality of each agent. In our perspective, this can be performed by using the MULAN/CAPA framework and following the PAOSE approach.

References

1. T. Anstett, F. Leymann, R. Mietzner, and S. Strauch. Towards bpm in the cloud: Exploiting different delivery models for the execution of business processes. In *Services - I, 2009 World Conference on*, pages 670–677, 2009.
2. Lee Badger, Tim Grance, Robert Patt-Corner, and Jeff Voas. Cloud computing synopsis and recommendations. Technical report, NIST Special Publications 800-146, 2011. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.
3. Hayat Bendoukha, Yahya Slimani, and Abdelkader Benyettou. Uml refinement for mapping uml activity diagrams into bpm specifications to compose service-oriented workflows. In Rachid Benlamri, editor, *Networked Digital Technologies*, volume 294 of *Communications in Computer and Information Science*, pages 537–548. Springer Berlin Heidelberg, 2012.
4. Sofiane Bendoukha. *Multi-Agent Approach for Managing Workflows in an Inter-Cloud Environment*. PhD thesis, University of Hamburg, Germany, 2017.
5. Sofiane Bendoukha, Hayat Bendoukha, and Daniel Moldt. Renewgrass - a tool for building scientific workflows: Application to the remote sensing domain. In *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*, pages 311–317, Aug 2015.
6. Sofiane Bendoukha, Daniel Moldt, and Hayat Bendoukha. Building cloud-based scientific workflows made easy: A remote sensing application. In Aaron Marcus, editor, *4th International Conference on Design, User Experience and Usability*, volume 9187, pages 277–288. Springer, 2015.
7. Sofiane Bendoukha and Thomas Wagner. Improving performance of complex workflows: Investigating moving net execution to the cloud. In Daniel Moldt, Heiko Rölke, and Harald Störrle, editors, *Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings*, volume 1372 of *CEUR Workshop Proceedings*, pages 171–189. CEUR-WS.org, 2015.
8. 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.

9. Lawrence Cabac. Net components: Concepts, tool, praxis. In Daniel Moldt, editor, *Petri Nets and Software Engineering, International Workshop, PNSE'09. Proceedings*, Technical Reports Université Paris 13, pages 17–33, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse, June 2009. Université Paris 13.
10. Michael Duvigneau. *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen*, volume 4 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010. Pdf: <http://www.sub.uni-hamburg.de/opus/volltexte/2010/4662/index.html>.
11. Chris Haddad. Selecting a cloud platform : A platform as a service scorecard. Technical report, 2011. URL: <http://wso2.com/download/wso2-whitepaper-selecting-a-cloud-platform.pdf> [accessed December 2, 2014].
12. Yan-Bo Han, Jun-Yi Sun, Gui-Ling Wang, and Hou-Fu Li. A cloud-based bpm architecture with user-end distribution of non-compute-intensive activities and sensitive data. *Journal of Computer Science and Technology*, 25(6):1157–1167, 2010.
13. Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of workflow systems using reference nets – security and operability aspects. In Kurt Jensen, editor, *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, August 2002. University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.
14. Fabrice Kordon and Daniel Moldt. Introduction to the special issue from petri nets 2016. *Sci. Comput. Program.*, 157:1–2, 2018.
15. Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
16. Markus Neteler, M. Hamish Bowman, Martin Landa, and Markus Metz. {GRASS} gis: A multi-purpose open source {GIS}. *Environmental Modelling & Software*, 31(0):124 – 130, 2012.
17. 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>.
18. Benjamin Schleinzer, Lawrence Cabac, Daniel Moldt, and Michael Duvigneau. From agents and plugins to plugin-agents, concepts for flexible architectures. In *New Technologies, Mobility and Security, 2008. International Conference, NTMS '08, Tangier, Morocco. Electronical proceedings*, pages 1–5. IEEE Xplore, 5.–7.November 2008.
19. Howard Smith and Peter Fingar. Workflow is just a pi process. *BPTrends*, January 2004.
20. Mirko Sonntag and Dimka Karastoyanova. Model-as-you-go: An approach for an advanced infrastructure for scientific workflows. *J. Grid Comput.*, 11(3):553–583, 2013.
21. Rüdiger Valk. Petri nets as token objects - an introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal*, number 1420, pages 1–25, 1998.
22. W.M.P. van der Aalst. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype", 2003.
23. Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, 3(3-4):171–200, 2005.

24. Yong Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *Services, 2007 IEEE Congress on*, pages 199–206, July 2007.