Workflow Engine Supporting RESTful Web Services*

Jerzy Brzeziński, Arkadiusz Danilecki, Jakub Flotyński,
Anna Kobusińska, and Andrzej Stroiński

Institute of Computing Science
Poznań University of Technology, Poland
{Jerzy.Brzezinski,Arkadiusz.Danilecki,
Anna.Kobusinska,Andrzej.Stroinski}@cs.put.poznan.pl,
Jakub.Flotynski@gmail.com

Abstract. An efficient business process execution and management are crucial for using a Service Oriented Architectures (SOA). Despite there are many applications offering such a functionality for Big Web Services, there is a lack of easy-to-use and well defined tools supporting the alternative approach, called ROA and RESTful Web-Services. In this paper the business process engine implementing a declarative business process language supporting web services compatible with REST paradigm is discussed.

Keywords: SOA, REST, business process engine, workflow, mashup.

1 Introduction

A *Service Oriented Architecture* (SOA) is currently a widely used approach to develop the complex distributed systems. A basic unit of the SOA is *a web service* — an independent software component offering functionality, which can be described, published, and discovered over the network using standard protocols. Web services can be used independently of each other, or they can be composed into a *business processes*, allowing the implementation of the complex system functionality by using already existing services performing simple functions. Besides providing a wide range of functionality, business processes also increase system component's life-time, re-usability and scalability.

The composition of a business process requires the definition of collaboration activities and data-exchange messages among involved web services. In order to enable this, the business processes description languages (workflow languages) are required. However, the description of the business process is not sufficient on its own, it needs some kind of environment that allows to execute it. Such an environment is provided by applications called *workflow engines*.

* Acknowledgment: The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

N.T. Nguyen, C.-G. Kim, and A. Janiak (Eds.): ACIIDS 2011, LNAI 6591, pp. 377–385, 2011.

There are many tools supporting the business processes description and execution, available up to date. Among them are either high level programing languages (e.g. BPEL [13], GWDL [7], BPML [8], AGWL [9], Java Orchestration Language [14], YAWL [12]), or the abstract graphical notations, such as BPMN [16].

Unfortunately, the existing solutions usually fully-support only one of the approaches used to implement the SOA, called in the literature Big Web-Services [10], and based on the SOAP standard. Big Web-Services use a huge stack of defined standards, which are required to be met by developers during the system implementation. An alternative approach to the SOA implementation is the Resource Oriented Architecture [15] and the REST paradigm[10]. RESTful Web Services (web services implemented accordingly to the REST paradigm) have an easy to use and understand interface, based on the HTTP protocol. Moreover, the communication between RESTful Web Services and clients avoids a ballast, associated mainly with a necessity of the XML processing, occurring in the Big Web-Services. The increasing attention the RESTful Web Services gained recently, motivated the need of introducing the workflow language well-suited to this technology [11] and developing the workflow engine for that language. This paper is devoted to the proposed workflow engine, and its architecture.

The paper is structured as follows: Section 2 presents the work related to the existing workflow engines. A Restful Oriented Workflow Language, called ROSWELL is mentioned in the Section 3. Section 4 is devoted to the architecture and implementation of the proposed ROSWELL engine. In Section 5 the results of proposed engine evaluation tests are presented, and finally in Section 6 the conclusions and the future work are described.

2 Related Work

Because of the growing interest in applications compatible with SOA, many tools supporting their creation were proposed. Main disadvantage of existing tools is their focus on Big Web-Services and marginalization of ROA and REST paradigm. Below, the most important currently existing languages and workflow engines are discussed.

Oracle BPEL Process Manager [5] is a commercial engine, which offers many advanced functions, such as management of process versions, tracking of process execution, process dehydration, interaction with a human and the creation of group of servers (*clustering*). Main disadvantages of this product is its high price and only a partial support for REST paradigm. Another BPEL based workflow engine is an open-source Apache ODE. Its interesting feature is the implementation of WSDL 1.1 HTTP binding extensions, which allows the usage of HTTP methods to invoke services. Unfortunately this approach is not sufficient to operate on RESTful Web Services as a set of resources, therefore resource oriented architecture must be hidden behind the standard Web-Service interface. JOpera [4] is a tool distributed as the Eclipse platform plug-in. It introduces the JOpera Visual Composition Language, which is a graphical notation allowing simple and

fast network service programming. Such a description can be transformed to the BPEL code. Other JOpera features are human interactions and Java snippets. Developers preferring Ruby language can choose the free Route-REST [6]. This environment makes the REST API available, extends the Ruby language and allows programming of business processes on the high level of abstraction. Processing — a list of connected activities — is modeled as a finite state machine. Unfortunately, a precise description of activities, business process participants and relations between them is still lacking. The main drawback of a Route-REST is a very inaccurate documentation.

3 ROSWELL — RESTful Oriented Workflow Language

The important issue when choosing a workflow engine, is its support for a programming language. A declarative business process description seems to be a good alternative to the existing imperative languages. It focuses on defining goals to achieve and restrictions to fulfill, rather than listing successive steps in the application. Moreover, because declarative approach is more similar to a natural way of thinking, it represents a higher abstraction level and can speed up a programming process. Unfortunately, the declarative way of describing the business process is still not well supported by the existing workflow engines. There are only a few solutions executing a declarative business process descriptions, their documentation is however usually definitely unsatisfactory.

A business process engine presented in this paper is the implementation of the ROSWELL — a declarative language supporting service oriented architecture and the REST paradigm [11]. The supported language has a syntax similar to a Prolog language, enriched with instructions supporting REST paradigm. Similarly, to the declarative program structure, the ROSWELL is a set of goal declarations. The goal is described by requirements that must be fulfilled (*goal body*) and which are available by a predefined interface (*goal header*). The goal body is a list of *logical conditions,* which can be a *complex logic condition* (*alternative* or *conjunction)* consisting of other *alternatives*, *conjunctions* or *simple logical conditions* (*restrictions*). The *Restriction* can be defined as a goal, math operator or predefined keyword, such as *true*, *false* or a keyword supporting REST paradigm. The last important feature of the ROSWELL is the data representation, which is based on: constants (numbers and strings), variables and structures. A variable type is not declared, but it is set dynamically during a variable substitution. The structure is a hierarchic data type consisting of other structures and variables. Structures can be modified dynamically during the business process execution.

In the presented approach, SOA is implemented as a group of resources available through HTTP protocol. ROSWELL offers a native support for SOA and REST paradigm by special instructions associated with the individual ROA properties [15]: uniform interface, addressablility, statelessness and connectedness. The uniform interface enables access to resources by HTTP protocol methods. The considered language introduces some special keywords — *onGet*, *onPut*,

onDelete, *onPost*, *get*, *put*, *delete*, *post* — allowing defining an uniform interface and calling it from other resources [11]. Addressability is a REST paradigm feature, which means that each resource has its own unique address. This address is specified by resource creation, and during the remote resource calling. Statelessness means that any state can be stored without the specified address between two requests from the same client. ROSWELL, which the presented in this paper workflow engine implements, has a special instruction allowing this, which can be useful especially for applications using HTML forms. Connectedness means that representation of a resource has some links to the associated resources. This is reached by special annotations, which are placed by a programmer in the HTML form and automatically replaced by a business process engine during the business process execution. The accurate description of all mentioned instructions of ROSWELL is contained in [11].

An example of ROSWELL construction is shown below:

```
onPut("http://hospital.pl/patients/{Name}", PData, PResp) :-
    PData->Age>18, addPatientToDB(PData),
    get("http://laboratory.pl/patients/{Name}", Req, Resp),
PResp->Body=Resp.
```

In the considered example, a new resource accessed by HTTP PUT method with the specified address is created, with patient's name, input (`PData`) and output (`PResp`) data. In the first step it is checked whether the patient's age is grater than 18 years. Next, a new patient is added to a database. Finally patient's data from laboratory is got and it is sent to a client as a response.

4 ROSWELL Workflow Engine

Taking pros and cons of the existing workflow engines, described in the Section 2, there is a lack of easy to use, well supported workflow engine fully supporting the REST paradigm. Therefore a new workflow engine implementing ROSWELL language [11], described in the previous Section, has been proposed and implemented. Some of the main features of the proposed solution are discussed below.

The presented ROSWELL workflow engine is compatible with the SOA and the REST paradigm. It is available as the RESTful service and supports the HTTP methods such as GET, PUT, POST and DELETE. The engine enables the interaction with a human, who is either a principal, or the other participant of the business process. A simple web browser can be used as the engine client, due to the fact, that the whole communication between all business process participants goes through the HTTP protocol. It is important to mention that all features of ROA paradigm [15] are supported by the ROSWELL workflow engine.

There are also other features that characterize the presented solution: portability (possibility of installation on many different operating systems), scalability (possibility of multiplication and cooperation between many business process engines), availability (new business process installation doesn't interfere with another users activities) and lightness (listed features are implemented in the way which minimizes system load).

4.1 Architecture Basic Assumptions

The main factor impacting the business process engine architecture is the manner of processing the declarative business process language. The ROSWELL workflow engine was decided to be implemented as a *language translator.* This approach means that ROSWELL language is translated into one of existing programming languages and then application is performed by existing tools and technologies. The main advantage of this solution is the simplicity of translator creation, in comparison with the alternative approach — interpreter implementation. Additionally, the translated and later compiled code usually works much more faster then interpreted code. The portability of this solution depends on the existing tools for the target language. The disadvantage of this approach comes form the fact that all implemented business processes depend on these tools, so any change of a business process implies effects for all existing applications.

4.2 Modular Architecture

The ROSWELL workflow engine has a modular construction. It consists of five modules, which create a chain of processing. The modules are logically separated, they offer the specific functionality and cooperate with other modules in order to reach the listed goals. All modules of the proposed business process engine are presented in Figure 1 and described below.

The *Network interface* is the module responsible for receiving clients requests, unpacking declarative language code and directing it to the next module. An

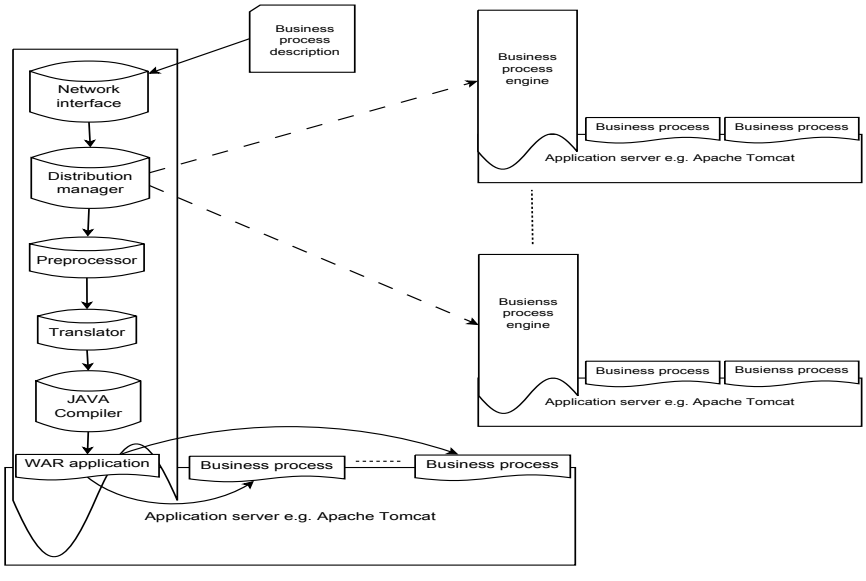

Fig. 1. Architecture and functioning of ROSWELL workflow engine

open-source platform implementing JSR-311 standard, called Jersey, was used to provide compatibility with REST paradigm. Applications supporting this paradigm can be started on any server supporting Java servlet technology. The *Distribution manager* is the module responsible for dividing description of business process into smaller parts, which can be installed and executed by other instances of the proposed business process engine. Such an approach enables distribution and multiplication of the executing business processes, so an efficiency and availability of the whole system is increased. A decision how to divide a process is made by a programmer or a dedicated algorithm. The *Preprocessor* is a module responsible for transformation of the input business process description into a more suitable for the next module, called translator, form. The preprocessor operates only on business process declarative language, so an output description is in the same language (the code is optimized). The first preprocessor task is a deletion of the redundant language constructions i.e. an exchange of facts to rules with an empty body. The second task is an expansion of the special language constructions i.e. generation of the additional code for Java snippets, to make it similar to other declarative languages rules. Preprocessor implementation was developed using Sun JavaCC [3] tool, which is a parser generator for Java language and supports LL grammars.

The *Translator* is the module responsible for transformation of a business process description received from the preprocessor into a description in particular programming language. Currently Java language is supported by presented business process engine. Implementations of Java Virtual Machines are available on the most popular operating systems, such as Windows or Linux, so this choice fulfills the requirement of portability. Similarly to the preprocessor, the translator module is generated by the JavaCC tool. The *JAVA Compiler* is the module responsible for compiling code generated by the translator to an executed code. In the discussed projectth eJava language compiler is used. The *Application server* is the outer tool, allowing the execution of the compiled applications. Such a role can play any Java application server, but the proposed business process engine uses Apache Tomcat [2].

4.3 Performance Tests

Tests comparing implemented ROSWELL engine (working on Apache Tomcat 5.5.28) with the Oracle BPEL Process Manager (Oracle SOA Suite 10g) were performed. The platform on which tests were performed has a subsequent parameters: AMD Sempron 3400+ processor, Gigabyte GA-M55plus-S3G mainboard, 900 MB Kingston CL5 RAM, operating system Windows XP. During the tests the speed of the execution of main language instructions, and the speed of performing main operations the engines offer were evaluated. A special application (a network service) was written for each test. Due to the limited number of article pages, only the supported declarative language application is quoted. The BPEL language listings are omitted.

The first tests type embraces:

- the substitution of the variable for constant string value "`test string`"
  ```
  onGet("Test1/" , Req1 , Res) :- X = "test string".
  ```
- the substitution of the variable for other variable integer value
  ```
  onGet("Test2/" , Req1 , Res) :- INT = 3, X = INT.
  ```
- the switch instruction comparing a variable value with a constant integer value
  ```
  onGet("Test3/" , Req1 , Res) :- X = 5, X1 == 100; X = 30.
  ```

Instructions listed above are simple, and perform very quickly, so they are repeated many times to obtain the reasonable tests results. Thus, instructions performed during tests were executed in a loop. The time was registered always before and after the loop. Each loop provides one average of the instructions execution speed. During tests, speed differences of both engines came to light. Thus, to obtain the credible results, numbers of repetitions should vary for both engines. It was intended to minimize outer short-lived factors, which could had some influence on the tests results. Therefore, loops were also performed many times and after each loop there were some breaks for a short time. A result of the test describes the average value of all executions performed in all loops. The precision of performed tests is 1ms.

The second type of tests contains subsequent comparisons: code compilation speed, the speed of application installations on the engine, utilization of RAM, and the speed of the synchronous requests realization. Each of the above tests were repeated 10 times. First two tests relied on the compilation and installation of the most simple application (doing nothing) on the tested engines. The precision of these tests is determined by Oracle JDeveloper 10g tool and equals 1s. Test of memory utilization was done using "Windows task manager". Memory utilization was measured each time before and after starting of workflow engine. The precision of such a test is 1 KB.

The last of listed above tests — speed of synchronous requests realization — was performed by the application Apache JMeter 2.4 [1]. 11 measurements were obtained for the following numbers of synchronous threads: 1, 100, 200, ..., 1000. Each thread simulated one client connecting to the engine and calling a test service. The time dependencies between starting threads were not set. A number of repetitions was fixed to serve always 10000 requests. The JMeter measured an average time of request realization — time from getting a message to sending a response. As previously, the most simple network service was used. Precision of such a test is 1 ms. Tests results are presented in the table 1.

According to the obtained results, the ROSWELL engine outperforms the Oracle BPM. The ROSWELL workflow engine results turned out to be better in all

Table 1. Tests results

Test	Oracle BPM	Business process engine	Relation OBPM/BPE
Substitution of variable by constant value [ms]	35,2047	0,0019	18895
Substitution of variable by other variable's value [ms]	1,4667	0,0178	82
Switch instruction [ms]	1,4312	0,0167	86
Speed of code compilation [s]	11,60	0,30	38
Speed of installation [s]	28,80	7,30	4
Utilization of RAM [KB]	654046	70863	9

performed tests, especially in the speed of instructions realization. Certainly, it is caused by the compilation of the declarative language to an efficiently executed Java code. The biggest difference was obtained for the test of substitution variable with a constant value. It shows that operations on string values are not a strong point of the Oracle BPM. Execution of the main administration task and utilization of the RAM came out better for ROSWELL engine with the minimal advantage of 4 times. Figure 2 presents times of requests handling in relation to the number of synchronous threads for both engines.

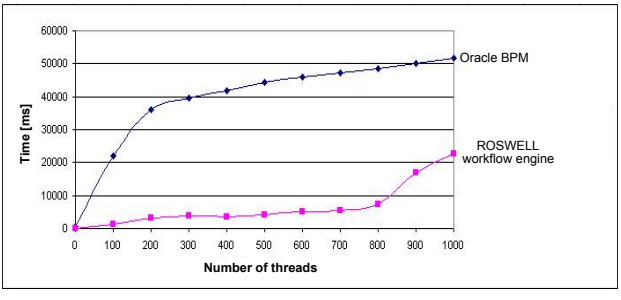

Fig. 2. Times of requests realization in relation to number of synchronous threads

The ROSWELL engine obtained a significant advantage, especially for the number of threads in the range 1..700 — the time of one requests handling is several times lower. There are several reasons of business process engine superiority in all the performed tests. Firstly, the declarative language is translated to the Java code, which is next compiled and executed very fast. The second important issue is the necessity of the XML processing through the Oracle BPM at each time when a new request occurs. This problem doesnot exist in the case of the business process engines with the discretionary form of requests. Moreover, the Oracle BPM is a very complicated system offering many advanced functions, so its memory and computational power demand is higher.

5 Conclusions and Future Work

The main target of the presented work was the creation of an efficient and lightweight business process execution environment. Such an environment should be able to execute a declarative processing and to allow RESTful web-services composition, according to the ROSWELL language assumptions. The presented performance results show a significant advantage of proposed solution — ROSWELL workflow engine — in comparison with the commercial tool, Oracle BPEL Process Manager. The presented solution could be an alternative to existing commercial, complicated, and mainly SOAP oriented systems.

Several extensions of the proposed workflow engine are planned, which can facilitate its usage. The first extension is tracking and modifying business processes

execution. Such a useful solution is currently available in the Oracle BPM. Another proposed improvement is the implementation of the described *Distribution manager* module. It would considerably increase the efficiency and reliability of the discussed environment. The last proposition is an implementation of all extensions, which are expected for the implemented ROSWELL language, such as asynchronous requests or parallel rule execution.

References

1. Home site of Apache JMeter project, `http://jakarta.apache.org/jmeter`
2. Home site of Apache Tomcat project, `http://tomcat.apache.org`
3. Home site of JavaCC project, `https://javacc.dev.java.net`
4. JOpera for Eclipse, `http://www.jopera.org/docs/help/jop.html`
5. Site of Oracle BPM project, `http://www.oracle.com/us/technologies/bpm/index.html`
6. Site of Ruote-Rest project, `http://github.com/jmettraux/ruote-rest`
7. Alt, M., Gorlatch, S., Hoheisel, A., Pohl, H.W.: A grid workflow language using high-level petri nets (2006)
8. Arkin, A.: Business process modeling language (2002)
9. Fahringer, T., Qin, J., Hainzer, S.: Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language (2005)
10. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000), `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`
11. Flotynski, J., Stroinski, A.: Declarative business process description supporting the restful paradigm. Tech. rep., Poznan University of Technology (2010)
12. Hofstede, A.H., van der Aalst, W.M.: Yawl: yet another workflow language (2005)
13. Louridas, P.: Orchestrating web services with BPEL (2008)
14. Pautasso, C., Alonso, G.: Jopera: A toolkit for efficient visual composition of web services (2004)
15. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly, Sebastopol (2007)
16. White, M.A.: Introduction to BPMN (2004)