Automation of Privacy Preserving BPMS in Collaborative Cloud-Based Business Processes

Sergei Makarov[1(✉)], Björn Schwarzbach[1], Michael Glöckner[1], Bogdan Franczyk[1], and André Ludwig[2]

[1] Information Systems Institute, Leipzig University, Leipzig, Germany
{makarov, schwarzbach, gloeckner, franczyk}@wifa.uni-leipzig.de
[2] Kühne Logistics University, Hamburg, Germany
andre.ludwig@the-klu.org

Abstract. Collaboration in business environments is an ongoing trend that is enabled by and based on cloud computing. It supports flexible and ad-hoc reconfiguration and integration of different services, which are provided and used via the internet, and implemented within business processes. This is an important competitive advantage for the participating stakeholders. However, trust, policy compliance, and data privacy are emerging issues that result from the distributed data handling in cloud-based business processes. In an earlier paper, we have discussed the requirements that a Business Process Management System (BPMS) should meet in order to enable privacy preserving business process as a service. This paper presents the necessary steps for implementing such a BPMS for an architecture to enable management of privacy-preserving collaborative business process.

Keywords: Privacy · Business process · BPMS · Cloud · Implementation

1 Introduction

Data privacy compliance, its handling as well as its security are the most concerning aspects for companies that collaborate in business environments, especially in the distributed ones such as cloud environments. One of the core components of such an environment are Business Process Management Systems (BPMS). An architecture for such an environment is proposed in [1, 2].

BPMS evolved from workflow and document management systems. They are able to cover full process management life-cycles in order to provide process design, modeling, execution, monitoring, and optimization [3]. However, focusing on the business process execution remains the most important task, as the other ones can be delegated to and taken over by the other architecture's components. The "divide-and-conquer" principal reflects the idea of service oriented architecture as it presets to decompose the problem into smaller manageable sub-problems. However, its management is one of the most crucial issues [4]. This comprises especially the reorganization of BPMS' core functions such as the facilitated communication between system elements and parallel providing of appropriate process execution without violation of both privacy and security.

The central role of BPMS according to [5] refers to the integration layer of distributed components. Such a distribution can be found in cloud architectures. The cloud is an appropriate environment for companies with a high necessity for collaboration. The combination of BPMS and cloud solutions can amplify the effectiveness of business collaborations by services-on-demand [6].

The key contribution of the paper is the design and implementation of a fully automated BPMS component in such a privacy-preserving cloud architecture. Furthermore, process design and performance optimization are addressed.

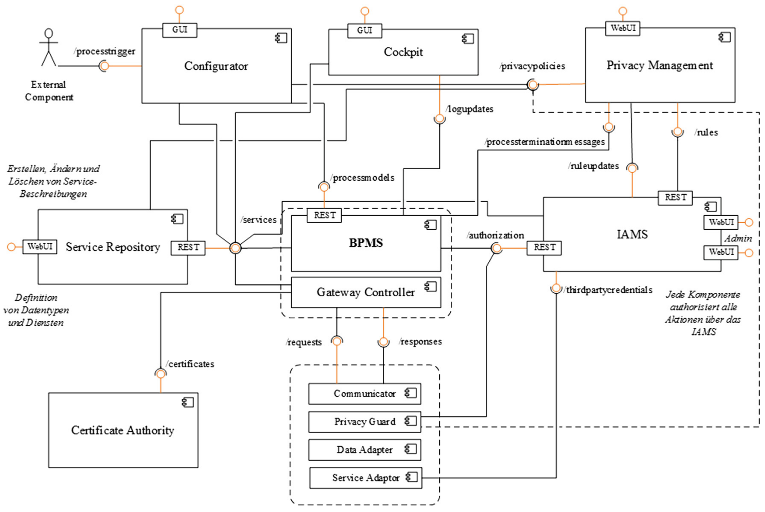

Fig. 1. Architecture for cloud-based collaborative BPaaS (Source: [1])

The first part of BPMS selection and its methodology was presented in [7]. This comprises project documentation analysis and literature research in order to determine requirements regarding software, BPMS, and further project-specific aspects. The result of that paper was the choice of jBPM as the BPMS tool that meets the requirements, and a first prototypical implementation. The architecture is shown in Fig. 1. The following sections present the implementation of automated collaborative processes enabling privacy compliance regarding the requirements defined in [1]. The architecture follows the SOA paradigm with comprehensive architecture and loose coupled components, which in turn can be denoted as web services in order to enable their automation.

The remainder of this paper is structured as follows: After the introduction, the theoretical background is briefly shown. Subsequently, the applied research methodology is outlined. Section 4 presents the implementation of the automated BPMS. Next, the results are discussed. The final section concludes the paper and gives an outlook on future research issues.

2 Theoretical Background

The essential objective of Business Process Management (BPM) is the planning and allocation of effective and efficient process solutions [8]. Identification and deployment of BPM focus on achieving business targets and customer benefits. Reference [3] emphasizes the scope on process identification in terms of business unit processes, content, and direction. In contrary, [8] refers to process allocation as institutional, permanent, goal-oriented planning, implementation, control and improvement during the entire process management lifecycle.

Concerning BPMS being deployed and utilized in Cloud environment, [6, 9] highlight the following advantages: scalability on IaaS layer (e.g. scalability of process instances); steadily available on-demand performance (e.g. CPU or storage capacity, especially in conjunction with data/predictive analytics for each process task); broad range of available SaaS providers for business activity monitoring tools to control activities of process instances; elasticity and availability. In addition, [10] consider BPMS independent from the implemented services. Thus, the changes in the process can be done without affecting target components, e.g. via business rules. Referring to [6], three following categories of systems with cloud-based models can be defined: interconnecting systems with IaaS and PaaS solutions for information exchange through EDI-systems, adaptive systems with SaaS models for monitoring of internal operations, and specialized systems such as CRM on SaaS layer.

The most proper steps for the companies' existing processes on the way to the cloud are outlined by [11] and comprise five following phases: Knowledge Externalization as representation of cloud services in man-computer readable form; BPaaS-Design via modular blocks; BPaaS Allocation as workflow orchestration of services, systems and components or even development of cloud-specific applications regarding user demands (can be referred to as Workflow-as-a-Service); BPaaS Execution as cloud orchestration between cloud services on the top abstract level (knowledge-based execution through the rules based on DMN format); BPaaS Evaluation as collection of log files from BPaaS deployment environment. However, it is not the only problem that should be addressed referencing migration of business processes. Further major concerns, addressed by [12], are data privacy and how private data can be prevented from unauthorized access, conflict of interests as well as physical resource isolation in such distributed environments with its scalability, service abstraction and local transparency. Moreover, data privacy and data integrity cannot be ensured only by securing business process operations but also security of the cloud environment itself should be taken into account. Hence, authentication and preventive measures are a major concern. Two problems are defined by [13–15] regarding data security. First, business processes are designed with architecture-specific restrictions in mind without concern of security requirements. Second, security aspects are integrated into an application in an ad-hoc manner - either during process implementation, administration or via outsourcing. The authors of [15] define the following five perspectives to be considered during process design in order to guarantee process integrity and consistency: Information Perspective for structuring and building relationships between information units; Function Perspective for representation of process activities and data flow between them; Dynamical Perspective for state representation of both information unit and status transition; Organization Perspective to determine by whom and where the process was executed; Business Process Perspective to represent the whole process as activity, and information data flow.

Regarding ad-hoc problematic, [14] consider following three basic causes: none or minor security aspects' specification during design, faulty infilling of security characteristics by non-security-experts, and impairment of cryptographic methods due to facilitated validation. The last one claims to find a trade-off between proper process model extension by complex security components such as cryptographic protocols and utilization of formal methods in order to provide straightforward implementation on abstract level. This problem can be solved by building a single secure environment in which the process has to be executed.

Unfortunately, common measures are not always sufficient. This problem is addressed in [12] by a four-tier model with security components provided on each level, namely Software, Platform, Infrastructure Security as well as Auditing and Compliance. The latter one has been extended with such components as User and Authorization Management systems as well as Access and SLA Management ones. Furthermore, [16] consider Trusted Third Party (TTP) services within cloud environment as ideal solution to warrant integrity, confidentiality and credibility. TTPs are operative, associated certificate paths which provide acquaintance about Public Key Infrastructure (PKI) and support the following security aspects: strong Authentication, Authorization, as well as data confidentiality. The major benefit of PKI, referring to [16], is coupling of directories such as Lightweight Directory Access Protocol (LDAP). Such directories in conjunction with PKI can be utilized among others for messaging as well as for private key definition. The most common approach in cloud computing is to use PKI together with Single Sign On technique in order to facilitate user experience with multiple applications within a single architecture.

3 Research Methodology

The evaluation of the presented artifacts is based on the Framework for Evaluation in Design Science Research (FEDS), developed by Venable [17]. Artifacts that have been created by a methodology based on Design Science in Information Systems Research [18] can be evaluated by this framework in order to ensure rigor. The framework offers several strategies the could be pursued depending on the characteristics of the designed artifacts. Generally, the FEDS regards evaluation as an ongoing process during design science research in order to improve the artifacts iteratively. Several characteristics influence the evaluation's purpose (why?), point of progress of the design process (when?), strategy (how?) and the artifact itself (what?). The characteristics and resulting strategies are briefly introduced. By outlining the characteristics of the current research, a strategy is chosen and the resulting methodological steps are described.

The framework distinguishes between formative and summative evaluation [19]. The main purpose of formative evaluation is to improve the results of an artifact in the ongoing research process. On the contrary, summative evaluations have the purpose to create a shared meaning of the artifact concerning distinct contexts of application.

The question about the point of progress of design evaluation can be chosen ex-ante or ex-post [19] during the continuous design process. While ex-ante evaluations are more predictive in order to e.g. select a certain technology alternative, ex-post approaches are used to assess developed artifacts in terms of applicability or degree of achievements of objectives. With this, a greater likelihood of ex-post evaluation can be expected for summative evaluations but is not obligatory [17]. Goals of evaluations can be for different purpose: either achievement of environmental utility, or usefulness of solving a specific problem, or comparative advantage over existing solutions, or a complex composite of criteria (e.g. functionality, completeness, consistency), or other impacts (side effects), or reason artifact's functioning.

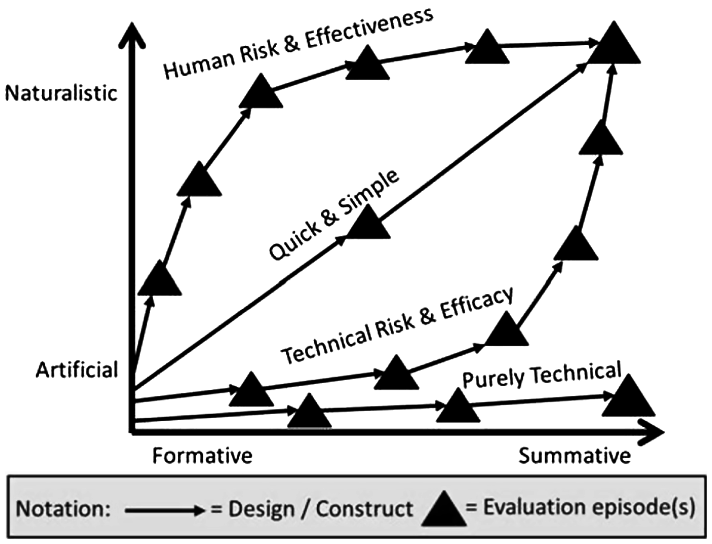
Naturalistic
Artificial
Formative
Summative
Human Risk & Effectiveness
Quick & Simple
Technical Risk & Efficacy
Purely Technical
Notation: = Design / Construct = Evaluation episode(s)

Fig. 2. Framework for Evaluation in Design Science (FEDS) with evaluation strategies (Source: [17])

The framework is displayed in Fig. 2, it comprises two dimensions. On the x-axis, the distinction between already described formative and summative evaluation purpose is located. The y-axis contains a distinction on how to evaluate with either artificial or naturalistic setup. While an artificial setup is used to prove general functionality of a concept, naturalistic evaluations prove an artifacts functionality in real environments, i.e. real people, real systems, and real settings [20]. Different *strategies* can be pursued that are displayed in Fig. 2 as well. Depending on the needs, available resources and circumstances, a strategy is chosen for and possibly changed during evaluation. The fastest strategy with the lowest costs is found in the 'quick\&simple' approach with a very limited number of iterations bears the risk of being not reasonable. A '*Purely Technical*' approach is suitable if naturalistic data and behavior is irrelevant and human users are not focus of the artifact. The other two strategies are used for either facing

'*Human Risk & Effectiveness*' or '*Technical Risk & Efficacy*'. A more detailed description of selecting a suitable strategy depending on specific circumstances can be found in Table 1.

Table 1. Circumstances for selecting a relevant DSR evaluation strategy [17]

DSR evaluation strategy	Circumstances selection criteria
Quick & simple	If small and simple construction of design, with low social and technical risk and uncertainty
Human risk & effectiveness	If the major design risk is social or user oriented *and/or* If it is relatively cheap to evaluate with real users in their real context *and/or* If a critical goal of the evaluation is to rigorously establish that the utility/benet will continue in real situations and over the long run
Technical risk & efficacy	If the major design risk is technically oriented *and/or* If it is prohibitively expensive to evaluate with real users and real systems in the real setting *and/or* If a critical goal of the evaluation is to rigorously establish that the utility/benet is due to the artifact, not something else
Purely technical artefact	If artifact is purely technical (no social aspects) or artifact use will be well in future and not today

The FEDS proposes 4 particular steps during the evaluation process [17]:

1. *Explicate the goals:* 4 goals of the evaluation were distinguished:
   (a) *Rigor* focuses on confirming that the artifact directly produced a certain effect (more likely to be shown with artificial evaluation) or that an instantiation of the artifact works thoroughly in a real situation (more likely to be shown with naturalistic evaluation). A summative evaluation provides the greatest rigor and reliability of the produced knowledge [17].
   (b) *Uncertainty and risk reduction* focuses on reducing either human and social risks or on reducing technical risks, which influences the choice of strategy (see Table 1).
   (c) *Ethics* focuses on reduction of potential risks to animals, people, or the public society. With this, especially potential stakeholders should not be put into risk.
   (d) *Efficiency* focuses on balancing the aforementioned goals in case of resource shortage. Hence, a more formative evaluation is proposed.
2. *Choose a strategy or strategies for the evaluation:* Depending on the aforementioned goals and the described circumstances of Table 1, one or more strategies have to be chosen. This can be done with a 4-step heuristic: (1) evaluate and prioritize design risks (either social/user oriented or technical or both). (2) Estimation of costs for real users, real systems and real settings. If human feedback is

available for a reasonable price, the '*Human Risk & Effectiveness*' strategy is suitable. If the price is too high or serious health concerns exist for users, the '*Technical Risk & Efficacy*' strategy is favorable. (3) If the artifact is purely technical and potential usage lies in remote future, the 'Purely Technical' strategy appears to be suitable or a naturalistic evaluation is just impossible. (4) If the construction that is to be evaluated is of rather small and simple extent, and none of the above-mentioned risks apply, the '*Quick & Simple*' strategy is the best choice.

3. *Determine the properties to evaluate:* the general set of features, goals and requirements of the artifacts that are to be evaluated are chosen. Again, a heuristic with 4 steps is proposed: (1) determine a list of potential evaluands (examples are given in [20–23]), (2) evaluands are to be aligned with the chosen goals, (3) depending on the chosen strategy of step 2, the evaluands should be of rather naturalistic or technical character and (4) determine the final list of evaluands.
4. *Design the individual evaluation episode:* the 3 heuristic sub-steps comprise: (1) derived from the environmental constraints, availability of resources determines their usage. (2) Priority shall be given to essential and more important aspects and resource are to be (re-)allocated. (3) Determination of number and structure of evaluation episodes and the according responsibility.

4 Research Findings

As already mentioned in introduction, we skip both the part of BPMS selection and requirements identification, which have been described in detail in [7]. However, we discuss thoroughly the implementation of those functional requirements, namely Service Selection, Remote Invocation, Process Activity Logging, external Security Provider as well as Cloud Readiness concerning BPMS from the architecture components' perspective. We also focus on additional components we had to embed in order to fulfil the objectives required by project, and which are not a part of the initial project's architecture shown in Fig. 1. The relationships and data exchange format between BPMS and project's components are described in Table 2.

Table 2. Architecture's components in association with BPMS

Component	Relationship/Direction	Format	API
Configurator	Configurator -> BPMS Controller -> BPMS	BPMN/XML	REST
Service-Repository (SR)	BPMS -> SR	JSON	REST
Privacy Management System (PMS)	BPMS -> PMS	XML	REST
Gateway	BPMS <-> Gateway	XML	REST
Cockpit	BPMS -> Log Collector <- Cockpit	JSON	REST
IAMS	BPMS (Security Provider) -> IAMS	XML/XACML	REST

Due to component's loose coupling, the services have a little concern about the process itself, which is entirely encapsulated within BPM engine. As a result, their communication happens either via BPMS' external API such a REST one or via additional middleware, provided complementarily, in order to act between two or more components within the entire system. We refer to such a broker as BPMS-Controller and denote an interface for log data deposition as Log-Collector (both for BPMS and IAMS log data). Additionally, we implement one more external element, namely XML-database (BaseX), to store temporally both BPMN model and I/O mapping data used for process task definition and execution. This concept decouples independent elements of the system, such as Configurator and BPM engine, as well as considerably facilitates process design. The last one implies much clearer process modelling, minimizing of possible errors as well as providing significant speed improvements. All these are possible without the necessity to have some programming background by the process designer.

The tool we have chosen during evaluation phase, which process is thoroughly described in [7], is jBPM BPM system, the open source product of Red Hat, Inc. with comprehensive API possibilities. Thus, its two following interfaces were specified for project objectives' realization: external one to provide the communication with other architecture's components (mainly via REST API) and internal one for implementation of the whole process logic and data processing (mainly via *WorkItem* API). The availability of customizable BPMN activities or tasks, which behavior can be defined by the user self by utilizing corresponding interface, was one of the criteria for choosing BPMS and is referred to as Service Selection, in the sense of reusing process activities. In jBPM such tasks are provided by *WorkItemHandler* (WIH) interface as well as abstract class called *AbstractWorkItemHandler*. The last one implements in turn WIH interface and extends its functionality through *StatefulKnowledgeSession* object being injected during subclass initialization as constructor's parameter. Listing 1 exemplifies the building of custom work item arbitrary defined as *ExecuteGenericTask*.

```
public class ExecuteGenericTask extends AbstractWorkItemHandler {
  public ExecuteGenericTask(StatefulKnowledgeSession ksession) { super(ksession) }
  public void executeWorktItem(WorkItem wi, WorkItemManager wim) { … }
  public void abortWorkItem(WorkItem wi, WorkItemManager wim) {} }
```

Listing 1. Implementation of WorkItem API

According to API documentation [24], *StatefulKnowledgeSession* object provides the most common way to interact with process engine. The integration of custom-built work item into jBPM KIE Server, a standalone execution server, as jar-file has been thoroughly described in [7] as well.

4.1 Configurator

The role of Configurator is to provide a design of business processes as well as their subsequent deployment to BPMS engine. Referring to [25], the process elements such its META-data description, process itself, etc. should be embedded by the project, which will be compiled as *kjar*-file and deployed to KIE-server, in order to make it executable and provide a possibility to administrate it from KIE-workbench. This concern can be applied not only to a new process deployment, but also regarding multi-tenancy problem in the cloud, providing service usage for multiple users at a time. For these purposes, the independent data source, persistence and other requirements should be concerned. The idea is to generate new project (i.e. *kjar*-file) for each new user or organization by the first process start and then subsequently use this project as deployment unit to generate new process instances. In both cases it is not sufficient to use only jBPM REST API but also build management tool's command interface should be involved. For our project, Apache Maven was utilized since this building tool is used by jBPM itself. The provided programmatical approach consists of three following steps: temporal storing of either new or modified process in XML-database (since it should be designed from outside of jBPM workbench), project preparation as well as subsequent project's deployment to KIE execution server. Though the first step is optional and can be eliminated in case no any new or modified process model has been provided, the already existing process may be used.

(1) Storing BPMN model in XML-databank

The storage process utilizes BaseX REST API to save the process model as BPMN/XML file into the database. Simultaneously, the name of deployment unit provided by Configurator is to be sent to BPMS-Controller in order to be utilized for subsequent steps. They are used for the preparation of project artifacts such as *pom.xml* with appropriate variables for name definition, triggering the project's undeployment command since it is important to guarantee that the process with the same name has not been already deployed to jBPM execution server, as well as project's deployment itself. Due to asynchronous calls nature, according to [24], - the request may be always accepted by KIE Server, also in the case the undeployment process itself has been failed – it is important to check twice, namely before and after deployment, whether it has been indeed succeeded. The list of all deployed units is possible to obtain over REST with suffix "rest/deployment" added to URL with jBPM container. Listing 2 provides how the project saves in BaseX utilizing its REST API.

```
(1)  public void saveNewBPMNFile(String fileContent) {
(2)    String deploymentUnit = this.groupId + ":" + this.artifactId + ":" + this.version;
(3)    ByteArrayInputStream content = new ByteArrayInputStream(fileContent.getBytes());
(4)    InputStreamEntity is = new InputStreamEntity(content);
(5)    HttpPut createDb = new HttpPut(host + "/basex/rest/" + this.dbName);
(6)    HttpPut put = new HttpPut(URL + this.dbName + "/" + deploymentUnit + ".xml");
(7)    Trigger.send(this);}
```

Listing 2. API for Configurator for storing BPMN model in BaseX-DB

The *send()*-method (line 7, Listing 2) provide concurrently with process model storage operation the values for project name definition to BPMS-Controller (Listing 3). With those variables BPMS-Controller is able to execute project artifacts' preparations in order to trigger process undeployment command as well as provide *kjar* deployment. Both operations are described in detail below.

```
public void send(BaseXmlService basex) {
HttpPost post = new HttpPost(hostname + port + "/project-deployer/" + basex.groupId + "/" +
    basex.artifactId + "/" + basex.version);}
```

Listing 3. Transferring name definitions to BPMS-Controller

(2) Project Preparation

The full name of the deployment unit consists of project and process names as well as version, which in turn, from Maven perspective, denoted as *groupId*, *artifactId* and *version* accordingly. At least one of them should be changed to make it possible to deploy the project to execution server without any name conflict. Regarding the multi-tenancy problem, it makes sense to change only group id according to user or organization name, if the process remains unchanged in the project. The designation of variables to be changed in *pom.xml* file is provided by Listing 4 and should be saved as external file outside the project to be deployed to execution server.

```
<groupId>#{groupId}</groupId>
<artifactId>#{artifactId}</artifactId>
<version>#{version}</version>
```

Listing 4. Denoting variables to be changed in *pom.xml*

Listing 5 provides programmatical approach to substitution of those variables with the proper values. The values of variables to be set in new *pom.xml* file are to be derived either from BPMN model prepared by designer in Configurator or from the filename saved into XML-databank. Otherwise, full name of user started the process can be used. Since we interact with BPMS-Controller for further operations, these variables are to be retained from URL as parameters sent to it from Configurator or Privacy Management System accordingly.

```
Path path = Paths.get(System.getProperty("user.home") +
"/generic-project/pomToGenerate.xml");
String content = new String(Files.readAllBytes(path));
String replaced = content.replace("#{artifactId}", this.artifactId).replace("#{groupId}",
    this.groupId).replace("#{version}", this.version);
Files.write(Paths.get(System.getProperty("user.home") + "/generic-project/project-to-
    deploy/pom.xml"), replaced.getBytes());
```

Listing 5. Exchanging previously denoted variables in *pom.xml* file

The second important step during project preparation is the exchanging of BPMN-file with new or modified process stored in BaseX-database (this step can be eliminated in case the process model remains unchanged). Listing 6 provides code snippet for such a task.

```
(1) DeploymentUnit dUnit = new DeploymentUnit(this.groupId, this.artifactId, this.version);
    //variables coming from configurator
//retrieving process model from xml-db
(2) String content = baseXService.getXMLDocContent(dUnit.toString());
//saving those data as bpmn.file in project to be deployed
(3) Files.write(Paths.get(System.getProperty("user.home") + "/generic-project/project-to-
    deploy/prestige/prestige-generic-standalone/src/main/resources/" + "prestige-generic-
    standalone.bpmn2"), content.getBytes());
```

Listing 6. Exchanging content of BPMN-file with actual project from BaseX-DB

(3) Project's Deployment

Having both project artifacts been prepared and undeployment unit been verified, project deploying can be initialized as the next step (Listing 7). This time both Maven and jBPM APIs are to be implemented.

```
//execute 'mvn clean install deploy' command
(1) Runtime.getRuntime().exec("mvn -f " + System.getProperty("user.home") + "/generic-
    project/project-to-deploy/prestige/prestige-generic-standalone/" + " clean install deploy");
(2) //Runtime.getRuntime().exec("mvn -f " + System.getProperty("user.home") +
    "/Downloads/prestige-bpms-project-deployer/" + " clean install deploy");
//waiting for jar compilation
(3) Thread.sleep(30000);
//execute deployment of the process
(4) HttpPost request = new HttpPost(hostname + port + "/jbpm-console/rest/deployment/" + groupId
    + ":" + artifactId + ":" + version + "/deploy");
```

Listing 7. Project compilation as jar-file and its deployment to KIE server

During project's deployment phase, KIE Server searches, first of all, in local Maven repository installed on the virtual machine it is deployed to. Secondly, it looks in an external repository, namely Maven Central. However, it is also possible to provide an URL to own repository being used and utilized privately by organization for storing their private artifacts. We believe it is a good practice to have an external deposition place for artifacts additionally to application's one since the project migration may be concerned. For such a purpose, Apache Archiva was installed and utilized in the production environment. To grant an access to it, the credentials are to be provided in settings.xml file in the default Maven folder called ".m2" (Listing 8). In order to make it possible to execute Maven "deploy" phase after "install" one (line 2, Listing 7), the URL and ID for external repository, the artifact has to be deployed to, are to be provided in *pom.xml* file (Listing 9).

```
(1) <server>
(2)   <id>external </id>
(3)   <username>admin</username>
(4)   <password>admin</password>
(5) </server>
```

Listing 8. Providing credentials to external repository in .m2/settings.xml file

```
<distributionManagement>
  <repository>
  <id>external </id>
  <url><hostname>:8580/repository/external /</url>
…
</distributionManagement>
```

Listing 9. Providing URL and ID for external repository in *pom.xml* file of project folder

Approach described in this section was implemented for automatic deployment once new or modified process is available in the database. However, we also consider this approach, as it has been already mentioned initially, regarding new user registered on the system, namely during its first process start. This can provide segregation of resources concerning various users in order to grant independent process execution for each of them. This has a particular meaning, due to impossibility to generate a new process instance during work item task execution.

4.2 Cockpit

Cockpit should provide graphical representation of operational process data. Those should be gained from Log-Collector, they were pushed in by each work item and BPMS-Controller during process execution. The second concern regarding this component is to provide a link for process continuation from a particular node it was suspended due to privacy rules violation. The objective of both requirements is to provide logging of each process step during process execution and it has an exceptional meaning for process automatization. The process log data should provide information about who has triggered the process, at what time and what service (activity) is being executed at the time among others. The challenge of such a task is that process engine persistence module commits no any transaction until the process is either finished or stopped (atomicity property). However, there are some elements in BPM notation, namely signal events and human-acting tasks, which provide intermediate state persistence. They are also known as Safe Points in terms of workflow engine. The other possibility associated with modifying isolation layer to the low state is not to be considered due to possible incorrect state output accordingly to [26]. However, after having paused the process, we need to trigger its execution so it can be processed with the next subsequent activity and so on. Mostly, this requires a human interaction in order to send a signal to the process in case of signal events or the execution of a particular task in case of human tasks implementation. Moreover, the problem becomes more severe if the process execution needs to be continued only if some particular requirements have been fulfilled - for instance, if the process was suspended by some service due to a privacy violation. In the last case, the process can either be triggered by the service user after providing additional information or automatically once privacy properties have been changed. The decision regarding process suspension depends on the Boolean variable called “suspended”. It has to be sent both to the BPMS-Controller along with other relevant process information for logging and to the XOR-Gateway within the process to decide which direction the process should follow after the trigger signal was sent from the BPMS-Controller (Listing 11). A snippet of the process model is provided in Fig. 3 and demonstrates the implementation. Listing 10 provides insights for Boolean programming of edges.

```
(1)  return KieFunctions.isTrue(suspended_var)
(2)  return KieFunctions.isFalse(suspended_var)
```

Listing 10. Programming of node edges between XOR-Gateway and WorkItems

```
(1)  HttpPost post = new HttpPost(URL + "/" + this.deploymentUnit +"/process/instance/" +
     this.processInstanceId + "/signal" + "?signal=" + String.valueOf(this.nodeInstanceId()+1) +
     "&event=" + this.suspended ());
```

Listing 11. Triggering Signal Event via REST API with variable for XOR-Gateway

When the BPMS-Controller gets a request from within a task node, the value of the node instance id and not of the signal event is sent. Thus, as it can be seen in Listing 11 after "signal"-prefix, this value has to be increased by one in order to denote the subsequent process node id.

Since ensued edges can be programmed logically, we specify the process flow as follows: the last executed task should be accomplished repeatedly in case "suspended" set to "true". This can be triggered by the users after changing their privacy rules. This step requires reverification of privacy settings newly provided or modified by user. Otherwise, further process execution continues automatically (in case of negative value of "suspended"). In both cases, the process generates log data due to "suspender" (signal event), which in turn is to be read off by BPMS-Controller from a jBPM database via REST API and to be sent further to Log-Collector.

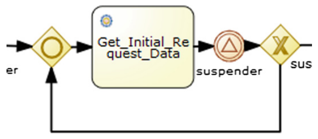

Fig. 3. Process flow accordingly to XOR-Gateway and variable named "suspended"

As already mentioned, this approach of manual trigger operation can be substituted with fully automatic process continuation direct after changing privacy rules, providing that the Privacy Management System takes over the handling of the signal event by itself. However, this was not considered by the project requirements so that the users can decide themselves, when some particular process instance should be continued.

4.3 Log-Collector

Log-Collector provides REST interface for storing both IAMS-data about the user, who started the process, and process META-data into relational database so that cockpit can gain those from it. The data, which collector has to be fed with, are to gain from a process within with the help of the following methods provided by *StatefulKnowledgeSession* object:

- *getProcessIntanceId(workItem)* method, its value is a number of the particular process instance. This value is particular useful for process handling via REST API. It has also been used to denote particular container for deposition of XML-files within BaseX with both process META-data and operational one during process execution.
- *getNodeInstance(workItem).getId()* or *getNodeInstance.*
- *(workItem).getNode().getId()* are both identical methods that return the node instance id as a sequence number of each node provided in the process and is strongly important in order to properly utilize signal events in jBPM. Unfortunately, a bug was detected in version 6.4 that generates new ids automatically after each task repetition (see section "Cockpit") and provides no correct response to them. Since we have utilized this particular version, we were not allowed to use this method but we had to provide task ids manually through I/O-parameters. This is redundant and impairs the idea of process design facilitation. We believe that this bug will be fixed in coming software versions.
- *getNodeInstance(workItem).getNodeName()* method provides information about the node name as a String type, its value has to be used for process monitoring dashboard – in the architecture known as Cockpit – in order to define on which point the error or privacy inconsistency had occurred. Since the node name reflects service description which the task executes, it facilitates further debugging, problem detection or provides user with more correct information about privacy compliance.
- *getNodeInstance(workItem).getNode().getMetaData().get("Lane").toString()* method helps to retrieve the name of the lane. However, since our process consists of a single swimlane, this method can be neglected.
- *KieContext.getVariable(String name)* is the method of *KieContext* interface, its implementation is injected in the KIE server during process execution and can be accessed via *kcontext* from within jBPM script task provided by the process modeler in the design palette. Since the process start has to be triggered from within the Privacy Management System and along with the user email, also other process relevant information have to be transferred, there is no other possibility to eliminate such a script task in order to facilitate the process model. Otherwise, Keycloak API or other libraries for token parsing can be utilized. It's also possible to gain this value from the KIE context, if the process would be started from the KIE workbench, given that the email is set up as a principal attribute of the Keycloak subdomain in the configuration file of the application server (see section "IAMS").

However, there is also another approach in jBPM to read full process log data off via REST API after persistence transaction has been proceeded (either at the end of process or at safe points). In contrast, this method does not allow direct access of the data during the task execution as *AbstractWorkItemHandler* does, but provides comprehensive information about the whole process instead. Listing 12 demonstrates snippets for log data querying via jBPM REST API from within BPMS-Controller in order to get process, node and variable logs of every particular process.

```
Process process = get(host + "jbpm-console/rest/history/instance/” + id);
List<Process> processes = get(host + “jbpm-console/rest/history/instances”);
List<Node> nodes = get(host + "jbpm-console/rest/history/instance/” + processInstanceId + “/node”);
List<Variable> variables = get(host + "jbpm-console/rest/history/instance/” + processInstanceId + “/variable”);
```

Listing 12. jBPM REST API for Log Queries

4.4 Service Repository

Process design can be very time-consuming and elaborative for its designer, especially by providing I/O-mapping between work item tasks. In order to facilitate this process, both service object description retrieved from service repository and Java XML-builder library were utilized within each work item. Thus, these require only one variable to be provided manually by each work item during the process design, namely service id. This way, XML-nodes and their values gained from gateway controller may be created automatically during process execution. For traversing through the document or building a new one there, both *w3c.dom* and *javax.xml* libraries were used. As BPMS and Gateway Controller communicate with each other on the basis of REST API and XML-payloads, it makes sense to allocate some sort of temporary storage during process execution, namely XML-database and grant the rights both for writing in and reading from for each work item. With such an approach, the I/O-exchange has been fully automated and needs no any human-involvement. Having implemented BaseX for such a role, a very concise API for managing database via REST is provided. Listing 13 demonstrates the simplicity with which both may be created. As container name, we have utilized a distinguished process instance id, while each XML-file overtakes the name of particular service (service id) provided by repository.

```
HttpPut putDb = new HttpPut(hostname + “/basex/rest/” + this.processInstanceId);
HttpPut putDocument = new HttpPut(hostname + “/basex/rest/” + this.processInstanceId + “/” + this.serviceId + “.xml”);
putDocument.setEntity(inputStreamEntity);
```

Listing 13. Creating both database and XML-document with BaseX REST API

Furthermore, XQuery can be used over REST against BaseX as a query language. It provides a possibility to search for particular values by providing either XML-node name or its attribute. The syntax for both methods has been provided in Listing 14.

```
HttpGet getByAttribute = new HttpGet(hostname + processInstanceId + “?query=//” + element + “[@” + attribute + “=” + value + “]”);
HttpGet getByNode = new HttpGet(hostname + processInstanceId + “?query=//” + rootNode + “/” + nodeName);
```

Listing 14. Retrieving values from XML-Document via BaseX API

With all methods described in this section, integration of XML-databank as additional component of BPMS-architecture can thoroughly facilitate data exchange between work items and thus provide high automatization level of process execution.

4.5 Gateway

Gateway consists of multiple various modules (Fig. 1) in order to provide data privacy handling during service invocations. This requires a high level of security and can be extremely time-consuming due to the amount of services arising with time in the service repository. Hence, process optimizing is required. Although a cloud architecture can provide a solution to the problem with its performance on-demand, the process itself should implement an asynchronous approach. Since Gateway is implemented with modern architecture and parallel processing in mind, multiple requests can be transferred to it from BPMS concurrently, without idly awaiting each REST response one after another, but rather acquiring them once they are proceeded. Listing 15 shows how this task can be achieved with Java default concurrency library.

```
(1)    List<Callable<Map<String, String>>> callsToGWC = new HashMap<>();
try { PostService postService = new PostService();
// get guid according to your request
(2)     callsToGWC.add( () ->
            postService.post(EncodingService.getBytes(payload), serviced));
} catch (Exception ex) { e.printStackTrace();}
(3)    List<Future<Map<String, String>>> gwcGuids = null
(4)    try {gwcGuids = executorPool.invokeAll(callsToGWC, 10000, TimeUnit.SECONDS); } catch
       (InterruptedException e) { e.printStackTrace();}
(5)     executorPool.shutdown();
```

Listing 15. Utilizing Java Concurrency Library for parallel requests

According to [27], the formula for calculation of threads quantity results from the number of CPUs and I/O intensity coefficient. For-intensive tasks, this value is almost 0, while it approximates to 1 the more I/O intensive task is. Since our process is I/O intensive, we calculate the number of threads as Listing 16 provides.

```
(1)    ExecutorService executorPool = Executors.newFixedThreadPool(
         (int) (coresNumber/(1-0.9) ));
```

Listing 16. Threads' number calculation

The strong performance improvements were detected after parallel processing implementation. However, the process (not the task execution) cannot be continued before all the requests are proceeded by Gateway. That is, further task execution can succeed gradually as responses are coming back but it has to wait up to the last one in order to finish the current one. Since the execution time differs thoroughly depending on the service, the transaction module of application server is to be set up accordingly. In Wildfly 8.x, such a setting can be provided in a standalone-full.xml file (Listing 17), to avoid the interruption of the task execution after five minutes' expiration set up by default (the value is to be provided in seconds).

```
<subsystem xmlns="urn:jboss:domain:transactions:2.0">
…
  <coordinator-environment default-timeout="43200"/>
</subsystem>
```

Listing 17. Changing default timeout of transaction module in standalone-full.xml

4.6 Privacy Management

With the security measures integrated into the cloud environment, the process has been deployed to, the transfer of operational parameters to jBPM from other components, such as Privacy Management System, can be run via URL utilizing jBPM REST API. This has been used to start or stop the process as well as to transfer some user relevant data, such as email, ip address, etc. All transferred variables are to be denoted with “map_”-prefix as provided in line 1, Listing 18 and can be retrieved with the help of KIE context via script task from within a process as it is provided by Listing 20. Immediately after process start, the response provided in payload comes with the relevant information about the started process. The actual process instance id can be retrieved in order to utilize it for further process handling such as process interruption (Listing 19).

```
(1) HttpPost post = new HttpPost(https://” + this.hostname + “:” + this.port + “/jbpm-
    console/rest/runtime/Prestige:prestige-generic-standalone:1.0/process/prestige-prestige-generic-
    standalone/start?map_ip= + this.ip + “&map_email=” + this.email”);
(2) post.setHeader(“Authorization”, “Basic “ + this.encoded);
(3) HttpClient client = HttpCli-
    ents.custom().setSSLSocketFactory(SecurityConfig.getSSLConnectionSocketFactory()).build();
```

Listing 18. Utilizing jBPM RESTAPI for remote process start

```
HttpPost post = new HttpPost(https://” + this.hostname + “:” + this.port + “/jbpm-
console/rest/runtime/Prestige:prestige-generic-standalone:1.0/process/instance” + this.instance +
“/abort”);
```

Listing 19. Providing URL for process instance termination

```
kcontext.setVariable(“email_var”, kcontext.getVariable(“email”));
kcontext.setVariable(“ip_var”, kcontext.getVariable(“ip”));
```

Listing 20. Retrieval of user data from within the process context and its assignment to process variables

4.7 IAMS

It is necessary to provide a consistent and protected environment for the service users during its utilization. However, not only the customers can benefit from such a uniform environment but also platform developers. The security unit of the entire architecture can facilitate the system tests and provide homogeneous methods. However, the integration of single sign-on processes can be very complex and challenging. Thus, one of the criteria for BPMS selection was the availability of security tools (in [7] known as Security Provider), which can be easily integrated within the platform. Keycloak is such a tool for jBPM, that extends its security functionality. It can be installed on the same application server through the appropriate adapter. Its installation provides extensions in *standalone-full.xml* file by adding an authentication subsystem with a flag set up to “required”. Having server been acquainted about Keycloak module, further security settings can be provided. The most significant ones are presented in Listing 21.

```
<subsystem xmlns=”urn:jboss:domain:keycloak:1.1”>
  <secure-deployment name=”kie-wb-6.4.0-Final.war”>
  <real>prestige</real>
  <realm-public-key>…</realm-public-key>
  <auth-server-url>https://...</auth-server-url>
  <ssl-required>external</ssl-required>
  <resource>jbpm</resource>
  <enable -basic-auth>true</enable-basic-auth>
  <credential name=”secret”>…</credential>
  <principal-attribute>email </principal-attribute>
  </secure-deployment>
</subsystem>
```

Listing 21. Providing security settings for jBPM platform

The settings in Listing 21 provide flexible options accordingly to the needs of developers. For instance, basic authorization can be activated along with security tokens. It can facilitate, for instance, integration tests. Also, principal attributes can be chosen from multiple options (name, email, etc.). This eases the gain of user identification without needs to retrieve this information from the token programmatically (in case, the process has been started from within a KIE workbench).

4.8 Discussion of Findings

Although BPM systems along with workflow ones have been introduced to provide fully automated process execution, some project requirements may impede this idea through human interaction involvement at those places, it could be eliminated. Furthermore, such complex systems, which provide full control over the process lifecycle, consist of multiple complex components, which in turn are hard to control and most of the time are not bug-free. These point to the necessity of finding a solution, that at least is not less efficient than a default one.

The whole process implementation regarding project requirements is shown in Fig. 4. Since both the process logic and the components’ communication are encapsulated by work items, it was possible to provide a clear, not overloaded design. Consequently, process modelling can be done faster while avoiding common process design mistakes. The final solution provided by Fig. 4, optimized in terms of performance and design, had met all project’s specific requirements and had successfully performed during evaluation phase.

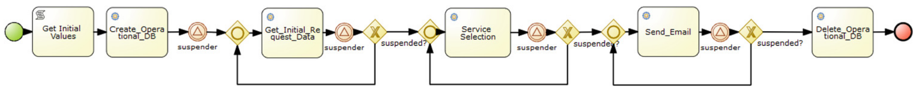

Fig. 4. Final process design and implementation

5 Conclusion

In this paper, the implementation of a fully automated BPMS for privacy-preserving management and execution of collaborative business processes has been discussed. This was the last piece that was needed to implement and evaluate the platform based on the architecture. The implementation and the architecture as a whole has been evaluated in multiple workshops with companies and researchers. The evaluation was positive, all the requirements that have been defined were met and the resulting platform was easy to use and was able to ensure privacy of the business data while executing the business processes.

Hence, the architecture and the fully automated BPMS enable the privacy preservation without the need for human interaction in every situation. Of course, the architecture is not perfect right now. The crucial part for the application of the architecture is the number of services available to the users. The task of implementing service adapters for additional services has to be carried out by the actual platform provider. Since the services adapters are simple, this should not raise any issues.

In the future, we will focus on improving the performance of the platform and optimization of the architecture.

References

1. Schwarzbach, B., Franczyk, B., Petrich, L., Schier, A., Hompel, M.T.: Cloud based privacy preserving collaborative business process management. In: 2016 IEEE International Conference on Computer and Information Technology (CIT), pp. 716–723. IEEE (2016)
2. Schier, A., Petrich, L., ten Hompel, M., Schwarzbach, B., Franczyk, B.: Cloud-Architektur für Privacy-Management in kollaborativen Logistikprozessen. Logist. J. **2192**, 1 (2016)
3. Schmelzer, H.J., Sesselmann, W.: Geschäftsprozessmanagement in der Praxis. Kunden zufriedenstellen, Produktivität steigern, Wert erhöhen: [das Standardwerk]. Hanser, München (2013)
4. Li, Z., Keung, J.: Software cost estimation framework for service-oriented architecture systems using divide-and-conquer approach. In: Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE), Proceedings, 4–5 June 2010, Nanjing, China, pp. 47–54. IEEE, Piscataway (2010)
5. Becker, J., Kugeler, M., Rosemann, M. (eds.): Prozessmanagement. Ein Leitfaden zur prozessorientierten Organisationsgestaltung. Springer Gabler, Berlin (2012). https://doi.org/10.1007/b138686
6. Hugos, M.H., Hulitzky, D.: Business in the Cloud. What Every Business Needs to Know About Cloud Computing. Wiley, Hoboken (2011)
7. Schwarzbach, B., Glöckner, M., Makarov, S., Franczyk, B., Ludwig, A.: Privacy preserving BPMS for collaborative BPaaS, pp. 925–934. IEEE (2017)

8. Mertens, P., Bodendorf, F., König, W., Picot, A., Schumann, M., Hess, T.: Grundzüge der Wirtschaftsinformatik. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-30515-3
9. Euting, S., Janiesch, C., Fischer, R., Tai, S., Weber, I.: Scalable business process execution in the cloud. In: 2014 International Conference on Communications and Networking (ComNet), 19–22 March 2014, Hammamet, Tunisia, pp. 175–184. IEEE, Piscataway (2014)
10. Megersa, B.T., Zhu, W.: Cloud-enabled business process management. Int. J. Comput. Theory Eng. **4**, 690 (2012)
11. Woitsch, R., Utz, W.: Business process as a service. Model based business and IT cloud alignment as a cloud offering. In: 2015 International Conference on Enterprise Systems (ES), pp. 121–130. IEEE (2015)
12. Chen, D., Zhao, H.: Data security and privacy protection issues in cloud computing. In: International Conference on Computer Science and Electronics Engineering (ICCSEE), Hangzhou, Zhejiang, China, 23–25 March 2012, pp. 647–651. IEEE, Piscataway (2012)
13. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: A BPMN extension for the modeling of security requirements in business processes. IEICE Trans. Inf. Syst. **E90-D**, 745–752 (2007)
14. Backes, M., Pfitzmann, B., Waidner, M.: Security in business process engineering. In: van der Aalst, W.M.P., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 168–183. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44895-0_12
15. Herrmann, G., Pernul, G.: Viewing business-process security from different perspectives. Int. J. Electron. Commer. **3**, 89–103 (2015)
16. Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Gener. Comput. Syst. **28**, 583–592 (2012)
17. Venable, J., Pries-Heje, J., Baskerville, R.: FEDS: a framework for evaluation in design science research. Eur. J. Inf. Syst., 1–13 (2014)
18. Hevner, A., March, S., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**, 75–105 (2004)
19. Wiliam, D., Black, P.: Meanings and consequences: a basis for distinguishing formative and summative functions of assessment? Br. Educ. Res. J. **22**, 537–548 (1996)
20. Sun, Y., Kantor, P.B.: Cross-evaluation: a new model for information system evaluation. J. Am. Soc. Inf. Sci. Technol. **57**, 614–628 (2006)
21. Mathiassen, L., Munk-Madsen, A., Nielsen, P.A., Stage, J., Jacksen, M.: Object-Oriented Analysis & Design. Marko, Aalborg (2000)
22. Smithson, S., Hirschheim, R.: Analysing information systems evaluation: another look at an old problem. Eur. J. Inf. Syst. **7**, 158–174 (1998)
23. Stufflebeam, D.L.: The CIPP model for evaluation. In: Kellaghan, T., Stufflebeam, D.L. (eds.) International Handbook of Educational Evaluation, vol. 9, pp. 31–62. Springer, Dordrecht (2003). https://doi.org/10.1007/978-94-010-0309-4_4
24. jboss.org. Community Documentation: jBPM Documentation. https://docs.jboss.org/jbpm/release/6.4.0.Final/jbpm-docs/html_single/
25. Fiorini, S.: Mastering jBPM6. Design, Build, and Deploy Business Process-Centric Applications Using the Cutting-Edge jBPM Technology Stack. Packt Publishing, Birmingham (2015)
26. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ANSI SQL isolation levels. SIGMOD Rec. **24**, 1–10 (1995)
27. Subramaniam, V., Hogan, B.P. (eds.): Programming Concurrency on the JVM. Mastering Synchronization, STM, and Actors. Pragmatic Bookshelf, Raleigh (2011)