Decomposed processes in Cloud BPM: techniques for monitoring and the use of OLC

José Martinez Garro¹, Patricia Bazán², and Javier Díaz²
¹Facultad de Informática, UNLP (Universidad Nacional de La Plata), La Plata, Buenos Aires, Argentina
²LINTI, Facultad de Informática, UNLP (Universidad Nacional de La Plata), La Plata, Buenos Aires, Argentina

Abstract - BPM (Business Process Management) in the Cloud has triggered revisions over several concepts like process decomposition, execution and monitoring. Decomposition allows processes to be executed in cloud oriented and embedded environments. This situation takes advantage of both approaches considering topics like sensitive data, high computing performance and system portability. Decomposed processes need to be monitored conserving the original business model’s perspective. It can be considered also for monitoring purposes some data generated during process status changes. These data units are useful due to they contain information associated with the process execution. In this paper we present a model for decomposed process monitoring which also considers OLC (Object Life Cycle) data objects in order to provide a wider set of information for measuring and improving purposes. We also make a comparison about features like process execution and monitoring, considering hybrid environments versus embedded ones, including the use of OLC data objects.

Keywords: OLC, decomposition, monitoring, cloud, BPM.

1 Introduction

The problem of monitoring a business process in a cloud oriented collaborative environment, conserving the process model’s original perspective is deepened in the present work. It is made by facing the possibility of saving and analyzing OLC data objects generated through process transitions, in order to embrace as much relevant information as possible for process measuring and improvement. This work begins with a current bibliography analysis considering concepts associated with process decomposition in a hybrid scheme, process monitoring in the cloud and the necessity of binding processes with data objects, in order to obtain relevant information for process measuring and improvement. In second place it is presented a proposal with the architecture of a process monitoring application which considers at the same time OLC data objects related to process transitions. It is also included a comparison about the mentioned trends in process execution and monitoring, considering both hybrid and embedded environments. Finalizing the document we present some conclusions about the current state of the art and future work proposals in these research lines.

1.1 Related work

There are different trends when it comes to BPM in the cloud. A major topic in the research field is process decomposition and the different concepts associated with it: expressions for cost calculation, equations to obtain the best distribution scheme according to the infrastructure and the involved applications, and also methods to improve process performance. There are several articles providing different perspectives on these subjects [2] [3] [6] [11]. Regarding to process monitoring, there are some research lines introducing concepts about process measurement, process performance, business activity monitoring and OLC data objects, generally in embedded environments exclusively [7] [8] [10]. Considering cloud environments in order to cover this gap, we have presented in [17] an architectural proposal for a decomposed process monitoring application in a hybrid environment. In the present paper we go further deep into this architecture, introducing some new features by applying the concept of OLC data objects in order to obtain more information for process monitoring and measuring.

2 Hybrid architectures

Privacy protection is a major concern when the purpose is to put BPM in the cloud. Once private data are outside the organization, there is a certain lack of control over them. Besides, it is necessary to observe product’s portability and its versions, and their availability in a cloud system. Other two problems also often considered are performance and efficiency.

The scalability and high availability of computing force are highly exploited by the intensive computational activities inside processes. The non intensive computational tasks, on the other hand, not always take advantage of this context. The performance of one activity running in an embedded environment should be better compared to the cloud, because of the amount of data that is transferred in order to execute the activity [1] [6].

• Possible architectures: in most BPM solutions, the process engine, the activities and data are located in the same side, even in an embedded or cloud
There are some papers introducing the PAD model (Process - Activity - Data) of Fig. 1 as a distribution possibility for BPM in the cloud. In this approach the process model, the involved activities and the process data are separately distributed. The PAD model defines four possibilities of distribution:

1) The first pattern is the traditional alternative where all elements are distributed over the final user side.

2) The second pattern is useful when the user already has a BPMS, but the high computing activities are located in the cloud in order to increment their performance.

3) The third pattern is useful for users who still do not have a BPMS, so they can use the cloud system in a “pay per use” way. In this approach the activities with low computing intensity or the ones with sensitive data management can be located on the final user side.

4) The fourth pattern is the cloud based model, where all the elements are located in the cloud.

Optimal distribution: the cloud system costs have been studied several times previously. There are formulas that allow calculating the optimal distribution of activities, since they can be located in the cloud or in an embedded system. The calculation takes in consideration different items, like time, money and privacy risks. [2] [3] [4] [6].

2.1 Criteria for process subdivision (Decomposition)

It is possible to generalize the distribution and identify a fifth pattern where the process engine, the activities and data are deployed in the cloud and in the final user simultaneously. This solution presents two potential benefits:

1) The process engine manages control and data flows. Once the activity receives data from the process engine and concludes its execution, the outcomes are passed again to the process engine. Considering now a sequence of activities located in the cloud, and a process engine deployed in the final user side: each activity uses data produced by the previous activity as an income. Data are not passed directly from one activity to the other but they are sent to the process engine first. Since data transference is one of the billing factors in this model, this kind of situations could become more expensive when large amounts of data are transmitted between activities. To avoid this problem a process engine can be added to the cloud, in order to regulate the control and data flows between the activities located in it.

2) When the cloud is not accessible, users can execute business processes in a complete way in the embedded system until the former one is available again [5] [11] [12].

In order to run a single business process between two separated engines, it should be divided into two individual processes, being convenient in this case for users to take a distribution list of the process and its activities. The process can be automatically transformed into two units: one in the cloud and the other in the embedded system. The communication between both systems can be described using a choreography language, like BPEL. Business process monitoring is more complicated now, since the process has been divided into two or more parts. As a solution, a monitoring tool can be developed for the original process, through the combination of the individual process monitoring details. This point will be analyzed further down [1] [16] [17]. As we presented previously in [17], there are several approaches implementing process decomposition in a hybrid architecture considering different aspects like data sensibility, application portability and high computing. Once the decomposition criterion was established, it is necessary to synchronize the different parts in runtime, which can be made following different lines. In our case, it was made by using Bonita BPMS and its process connectors, in order to initiate a new piece of the process in a new server once the previous one is finished. The result of this implementation is a set of cloud nodes joined by process connectors in runtime.
accomplishing in this way the original process model execution [5] [6] [16] [17].

2.2 Hybrid environments and process monitoring

As it has previously been seen, major problems about using a partitioned process model are, besides the proper execution, the gathering and monitoring of the different distributed instances (either in an embedded system or in the cloud), and at the same time the accomplishment of viewing them under the optic of the “original process” to which they belong. To face this inconvenience we have designed a solution considering distributed and intercommunicated components forming an architecture, which is described as follows. On the one hand, it is necessary to associate the different process instances initiated by using a chain, with the purpose of gathering information about them by accessing the different involved servers. The execution model based on decomposed processes consists of linking each instance flow to the corresponding partitioned processes. Thus, when an instance finishes in a server, it initiates automatically a new instance corresponding to the next process partition, depending on the distribution model. For this purpose, each node in the architecture should be capable of establishing communication with the next node in order to initiate new instances, and gather in this way information about them. Namely, given a new instance which was initiated in a node of the architecture, we should be able to obtain, not only its data but every instance generated by it in another server.

2.2.1 Bonita Open Solution: API and connectors

There are several ways of implementing instance flow linking. In our case we have selected Bonita Open Solution [17] as the BPMS. So, once the original process was partitioned over the servers following criteria like sensitive data storing, data transferring and application portability, we have used the API and the connectors provided by the BPMS in order to create instances and recover their information using Java classes. These classes use the API as libraries, including functions like server authentication, instance launching, instance information gathering and process variable setting. These classes are invoked from the process definition using connectors.

It was also included on each process definition the information needed for the communication with other Bonita servers present in the architecture. Taking advantage of this link, it is possible to launch new instances in those servers by using connectors. Thus, every instance when is finished will execute the connector, which allows initiating a new instance by using the API, linking in this way automatically the process execution flow [6] [16] [17].

2.2.2 Centralized front-end

As it was previously described, a monitoring application must be developed in order to show integrated data related to distributed instances. Facing the execution trace, it is very important for each instance to be able of storing, not only its own information but the one associated with the instances created by it over other servers. In this way, by accessing the initial instance of the process, it is possible to recover the information associated with the next instance, and so on in order to obtain the complete execution trace.

Once recovered the information from the different servers, it must be provided an application in charge of the gathering process and showing the obtained data seamlessly. The most important thing in this aspect is to accomplish monitoring transparency for the users: they should not be forced to distinguish the server where the activity was executed, but they should visualize the different instances and observe them as a unique entity, according to the original process model. The implementation of this feature was made through a cloud based web application. This application was placed there in order to access each involved server, being them cloud or embedded, assuring in this way access to users from every point. For this purpose it is important for the application to have a catalog with the existing servers considering their location information updated. Each of these servers has a copy of a web service (getInstanceService), which receives a process definition id and returns information of each instance existing in the server that is associated with the definition sent as a parameter. The information returned includes instance id, current status (executing, completed, suspended), current activity if the instance is not finalized, start and end date. In this way, the application located in the cloud sends to each server a web service invocation with the selected process definition as a parameter, and receives the information about the associated instances. Then, this information is visualized in a web interface, where the user can select a particular instance and observe its details. In order to make this, the application has another web service (getInstanceActivityService) used to get from the engines the details of each activity associated to the instance. The returned information includes activity id, participant, start date, current status and end date. Once ended this collection phase, we need to remember that each instance contains also some information about the different instances initiated by it over the servers in the architecture. In this way, the web application will have to concatenate the received information and allow the users to observe the monitoring details in a transparent and integrated way [8] [9] [12].

2.2.3 Application’s architecture

We can observe in Fig. 2 the different distributed components identified in the architecture design, as well as the internal relationship between them and the user.

The solution is composed by three main nodes: the cloud, the embedded or traditional systems and the monitoring application. The cloud works as a container for several elements: the BPMS, the monitoring application, the REST
API used by the developers in order to integrate applications with the process engine, and eventually a geolocation service which allows assigning to mobile clients the most convenient version of the service according to their localization [17].

2.2.4 Component Communication

If we consider every component present in the architecture, we have analyzed the communication between each one of them through an application communication diagram. There we can observe the most important involved applications, their main actors and the interaction of the different distributed software components.

On the other side we find the embedded type components, namely traditional BPM applications which belong to the organization, and because of different reasons like data sensibility or application portability, they are not located in the cloud. These nodes, functionally speaking, are equivalent to the cloud ones, even when they have access restrictions and lower computing force compared with the first ones. The third component is related to the monitoring function. It is used by the application, and is in charge of returning information about instances and activities which were executed in every node of the distributed architecture. The web services getInstance and getInstanceActivity were constructed jointly with the monitoring application, and are executed on demand by this one. They are communicated with the process servers through an API (in our case, the one provided by Bonita), and are in charge of returning, in first order, information about the instances initiated on each server, and then, some data about the activities composing these instances.

We can see at the same time the different user profiles involved in the execution of the components represented in the architecture. While the preponderant role in process execution is the activity’s participant, the monitoring site results are important for the business analyst, as well as for the architecture administrators who can optimize services or process components (Fig. 3).

A feature in common between the process execution application and the monitoring one is location transparency. Users should not be necessarily notified about changes in the execution environment, in case we are considering a decomposed process where the activities are located in different servers. This is very useful in order to allow users to have a unified vision of the process, more than a partitioned one, which main existence reason is related to technical issues and not with logical or business aspects generally. We can also visualize in Fig. 3 how both the execution and monitoring components access indistinctly to the cloud or the
embedded nodes, in order to gather information about each instance initiated over the distributed servers.

2.3 Enriching Process Monitoring by using OLC objects

2.3.1 Foundations

In the field of BPM, monitoring is used to observe process behavior, and also probably to react to events and predict future process steps during execution. Processes that are automated using information systems like process engines, can be correctly monitored due to the system often offers logging capabilities and thus, the process can be easily recognized. In contrast, in environments where processes must be executed manually in a big portion, for example in a health care dependency, a high number of events are not captured automatically. An event monitoring point is related to certain events captured by an IT system connected to a specific event source, e.g. a database or a barcode scanner, and informs when certain state transitions (for example enabled, started or finished) occur in a process activity. In this case, probabilistic means that it is possible to provide an indicative index about process progress, but it is just an approximation [7] [8] [9]. In this way, an approach using events extracted from data objects creation or modification is introduced in order to increment the number of observable events used for process monitoring and progress prediction. Those are called state transition data events. After recording a data object with a specific status assigned, we can assume its existence. Additionally, this approach can be used to identify incorrect behavior in a similar way of what is done in the data conformance field [12] [13] [14].

The presented approach allows approximations through process execution based on information about data objects. These objects and their life cycle are the foundation for this approach. An OLC data can be represented through a Petri net, where a node describes a data state and a transition represents a data state change from the predecessor to the successor. An OLC specifies all the allowed data state changes of the corresponding data object. Based on this OLC, the transitions that can be monitored with events are selected. We consider an event as a fact that occurs in a particular point in time, in a certain place and with a specific context which is represented in the IT system. The data state transitions that are observable have been connected to the events which provide information about transition triggering. This connection is made by joining the particular OLC transitions to an implementation which extracts information from an event source, including the correlation to a specific data object in runtime. Nevertheless, not all data state objects in the OLC need to be reflected in the process model. Based on the assignment of data objects to activities, it is possible to provide information about process execution in runtime. We assume that an activity is enabled if it can be executed with the control flow specifications, and the incoming data objects are available in the required data state. We also assume that an activity is done as soon as the output data object reaches a certain data state. In design time, data state transitions can be marked as “observable”, referencing the before mentioned runtime monitoring capabilities. In runtime, process progress can be recognized through data state transition events that occur in the event storage. In order to obtain non observable details in process monitoring through data manipulation, the introduced approach should be combined with other techniques [13] [14] [15].

2.3.2 OLC management in Cloud BPM

In terms of implementation, the monitoring application should be capable of gathering process status information, and at the same time, all the information marked as observable in process transitions.

![Decomposed Model View in design time](image)

In order to make this, it is necessary to bind the information in the process engine database with the information related to the different OLC data objects identified along the process definition. In Fig. 4 we can see how the different states are traversed by events and the information recording process in the environment. It is mandatory in this case to bind process activities and transitions with the respective OLC objects in order to gather all this information. We are considering that the process is already decomposed in the cloud environment, so the OLC objects should be collected from the corresponding nodes. The web application that we have considered before for process monitoring should be altered in order to gather this new information [7].
In Fig. 5 we can observe how the different transitions insert information in the event storage during runtime. It is important to bind the recorded information with process details, in order to obtain all the events once the monitoring application is gathering the data to be shown [8] [9]. Each service that implements process activities should be responsible for saving with the business data some information about the process execution (instance's identification, timestamps and activity's identification) in the event storage, in order to be able of recovering the OLC objects during the monitoring process. This is the most traditional way to integrate the process logic with business data: using event data persistence as a way of integration in order to recover the associated information [8].

2.3.3 Considering OLC objects in the Monitoring Component

Taking in consideration the previously presented architecture, the monitoring component is formed essentially by two web service definitions: getInstanceService and getInstanceActivityService. The first one is in charge of gathering information about an instance from each node in the hybrid architecture, and the second one acts once the first one was executed, in order to obtain details about each activity associated with the instance. Now these services will keep their behavior, but should change their definition in order to consider besides the OLC data objects associated with the process, which means not only to search over the process engine’s databases for information but to access the different sources in the event storage. All these mechanisms should be duplicated on each distributed node. A really important issue to consider in this context is how to display the OLC data objects in order to show them to the user in a helpful way. Analyze these objects provides an interesting vision about process execution, performance and progress, but they depend severely on the organization’s context, so the monitoring application should consider a standard visual way to show them in order to uncouple the visualization mechanism and the business logic [9] [10] [11].

2.4 Process Monitoring Techniques comparison

Process monitoring, mostly applied as BAM (Business Activity Monitoring), is a discipline deeply developed in embedded schemes, where all the information needed is located in one server, and the process engine has not major issues during gathering it. It was also researched the possibility of taking business data objects in order to enrich the monitoring process with information related to the organization’s logic besides the proper information collected by the engines. As it was previously presented, the inclusion of BPM in the cloud has triggered the consideration of hybrid schemes, where both the execution and monitoring processes are more intricate compared with the embedded version, due to the connectors and the distributed gathering process required. In Table I, we compare hybrid environments with traditional ones in terms of process execution and process monitoring, and several concepts associated with them.

<table>
<thead>
<tr>
<th>TABLE I. COMPARISON OF THE EXECUTION AND MONITORING FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Embedded environments</strong></td>
</tr>
<tr>
<td>Process execution</td>
</tr>
<tr>
<td>Is performed by the individual process engine, which is in charge of guaranteeing the execution flow between the activities and gateways existing in the process model.</td>
</tr>
<tr>
<td>Process monitoring</td>
</tr>
<tr>
<td>Generally, the embedded systems have their own components for process monitoring. They are in charge of gathering the information from the proper engine and perform measurements about the different defined KPIs.</td>
</tr>
<tr>
<td>Gathering process</td>
</tr>
<tr>
<td>This activity is performed by the same unit in charge of displaying the executed instances' progress in the individual engine.</td>
</tr>
<tr>
<td><strong>Hybrid environments</strong></td>
</tr>
<tr>
<td>The execution on each node is provided by the individual engine located inside of it. It is necessary to link the different instances through process connectors or a choreography language like BPEL.</td>
</tr>
<tr>
<td>It is necessary to build a new application in charge of gathering the information from the different nodes in order to accomplish the original view according to the decomposed process model.</td>
</tr>
<tr>
<td>Due data are distributed along the architecture, it is necessary to implement modules in charge of collecting information from the different intervening nodes.</td>
</tr>
</tbody>
</table>
3 Conclusions and future work

Process decomposition is a very useful mechanism in order to take advantage of the different features offered by embedded and cloud environments, but it inserts complications on how to monitor processes with the original model perspective, and also on how to obtain more relevant information related to the business logic and not necessarily to the process engine (like OLC data objects). Our future work is focused on improving the monitoring application in order to develop a standard visualization mechanism which allows showing the OLC information in an uncoupled way, ergo not to depend on the particular organization’s business logic, considering the fact that our processes are distributed over the hybrid architecture. For making this, we have to analyze some patterns about OLC objects and obtain a general BAM based set of indicators, useful for process measuring and progress analysis (Business Activity Monitoring), but now applied on cloud based distributed processes.

4 References


