Towards a Common Understanding of Business Process Instance Data

Nima Moghadam and Hye-young Paik
School of Computer Science and Engineering, University of New South Wale, Sydney, NSW 2052, Australia

Keywords: Business Process Management, Process Instances Data, Data Models, Interoperability, BPMS Architectures.

Abstract: In an organisation several Business Process Management System (BPMS) products can co-exist and work alongside each other. Each one of these BPM tools has its own definition of process instances, creating a heterogeneous environment. This reduces interoperability between business process management systems and increases the effort involved in analysing the data. In this paper, we propose a common model for business process instances, named *Business Process Instance Model (BPIM)*, which provides a holistic view of business process instances generated from multiple systems. BPIM consists of visual notations and their metadata schema. It captures three dimensions of process instances: process execution paths, instance data provenance and meta-data. BPIM aims to provide an abstract layer between the process instance repository and BPM engines, leading to common understanding of business process instances.

1 INTRODUCTION

A business process is a collection of related activities performed together to fulfill a goal in an organisation (Aguilar-Saven, 2004). A main function of a BPM system is to turn a business process model into an executable program so that the process described in the model is enacted to assist business operations.

A process instance is a concrete running instance of such a program containing (i) a subset of the activities appearing in the model that spawned the instance and (ii) materialised data (e.g., Customer Name, Order Number). For example, given a process model describing a car insurance claim process, a BPM system would enact concrete instances of the model, each instance representing an actual claim being processed and the details of the data involved.

Although business process instances could be short-lived, many process instances are in fact long running, in that they could take hours and days from start to finish. This is because a typical life-cycle of a business process instance could spend most of its life in wait mode (e.g., waiting for a reply from a previous request). When a running process instance reaches to the point that it needs to wait, the BPM system maps the instance information directly to physical storage artefacts such as relational tables or XML database and stores it. BPM systems also use a physical storage to store other information such as process instance execution logs. Organisations can use this information to analyse and improve their business processes (Grigori et al., 2004).

In modern enterprise environments, multiple BPM systems and applications co-exist and work along side each other. In this paper, we examine issues of business process instance management in such an environment. Figure 1 depicts a scenario where a single application (i.e., single process instance) is supported by two sub-processes, each implemented with different BPM solutions. Each BPM system has its own representation of process instances and a proprietary process instance repository, leading to a heterogeneous environment.

The lack of common understanding about business process instances amongst BPM systems could prompt the following problems:

- Having to analyse multiple/heterogeneous sources (e.g., Log Files, Data Tables) to extract complete process instance information.
- Having not enough information to fully describe a process instance. Some BPM systems do not store important information about process instance (e.g., which user or application started the instance, snapshots of data during the execution) and makes it impossible to understand the process instance fully.
- Tight-coupling of a process instance model to a physical storage model.

We believe the challenges in creating a process instance model to induce the common understanding

Moghadam, N. and Paik, H-y.
Towards a Common Understanding of Business Process Instance Data.
DOI: 10.5220/0005678401930200
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 193-200
ISBN: 978-989-758-168-7

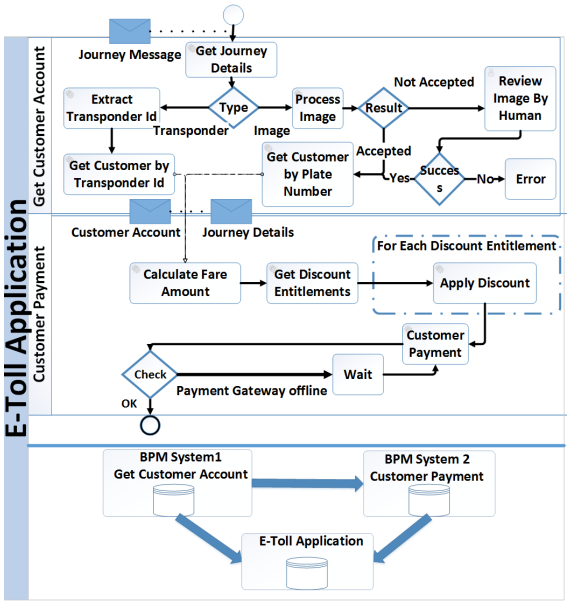


Figure 1: Customer Journey Process Implemented by Two Sub-processes: Get Customer Account, Customer Payment.

are two folds. First, such a model should contain all necessary information that a process execution engine needs to enact, suspend and resume a process instance effectively. Second, the model should be able to aid in creation of an architectural framework that allows decoupling of the business process instance management from an individual BPM system.

Here, we propose BPIM (Business Process Instance Model) which provides a holistic view of a business process instance by considering process execution paths, data provenance and relevant meta-data.

2 PROBLEM BACKGROUND

In this section, we examine the problem area in detail through a motivating scenario, *an E-Toll processing application*. The application is divided into two: `Get Customer Account` sub-process implemented by a third party solution using jBPM[1], and `Customer Payment` sub-process implemented using an in-house solution with Riftsaw BPM[2].

According to the model (Figure 1), the sub-process `Get Customer Account` retrieves a customer account and passes it to `Customer Payment` which calculates the final fare to be paid – considering discounts that may apply, and processes the payment.

When required (e.g., a long wait), an instance of the customer journey process would be stored in its respective BPM engine (i.e., jBPM (as part of `Get Customer Account` sub-process) and Riftsaw (as part of `Customer Payment` sub-process)). Note that BPM products use different data structures (e.g., Data Table, RDF, XML) to model the business process instance (Choi et al., 2007; Grigorova and Kamenarov, 2012; Ma et al., 2007). The E-Toll Application has its own private repository which contains subset of the data from the two BPM systems. The development teams have modified the services/BPM systems to send the data to E-Toll Applications private repository.

Let us explore the following scenarios to highlight the issues in the current BPM systems.

1. **Data Sharing:** Although the two sub-processes are interdependent, directly accessing and sharing the data is difficult because jBPM and Riftsaw are using different schemas.
2. **Failure and Error Diagnosis:** If something goes wrong, to investigate the root cause of the failure, the operation team needs to perform complicated tasks of going through the data in both systems.
3. **Migrating Data:** The stakeholders of E-Toll application request a new business report showing discount entitlements and payment details for each customer journey completed. The development team needs to modify the business processes and the Web services involved in this process to send the new information to the E-Toll application database. However, migrating the information for the existing process instances is hard due to the differences between the process instance models in the two BPM systems.
4. **Rollback/Re-start:** Managing a rollback or re-start of a process instance is difficult as different systems provides different level of supports. For example, jBPM lets you restart the process instance execution from a specific activity, but it does not roll back the changes which might have happened to the data.
5. **Data Inconsistency:** The database system in E-Toll application goes down for a while, but the BPM systems continue to run. This leads to data inconsistency between E-Toll and BPM system databases.
6. **Changing Process Instance Storage Technology:** Because the BPM systems are tightly coupled with its storage mechanism, it is impossible to replace the underlying technology (e.g., from RDBMS to No SQL).

To mitigate these types of problems, we propose a common model for representing business process instances, which BPM systems may adopt and use. The

[1]jBPM, www.jbpm.org
[2]RiftSaw Open Source BPEL,`riftsaw.jboss.org`

model, BPIM (Business Process Instance Model), defines a framework for process instance information; it defines three separate data aspects that can collectively build a *common view* on process instances. In doing so, BPIM aims to makes it easier to obtain a holistic view of the process instance and help build an abstract model that could de-couple the BPMS execution engine from a physical storage.

3 RELATED WORK

Most of the academic research work so far have focused on business process modelling and process model repositories (Yan et al., 2012; Choi et al., 2007; Grigorova and Kamenarov, 2012). In-depth discussions about models for business process instances have been largely neglected. We discuss the most relevant streams of academic work below.

Interoperability: The issue of interoperability between process instances in a distributed environment is discussed in Zaplata et al. (Zaplata et al., 2010). They have identified BPEL and XPDL as the most popular execution languages and conducted a comprehensive study on the elements in these languages to develop a model for process instances that is flexible enough to be used for both languages. Using this model, a BPM execution engine (a source environment) can transform its native process instance to an interoperable instance and sends the information to another BPMS execution engine (a target environment). However, much of the elements in the model focuses on the migration aspects and does not provide the holistic view a process instance.

Artefact Oriented BPM: Recently, a data-oriented business process view has emerged. The approach allows decoupling of the process instance data from its execution engine. Sun et al. (Sun et al., 2014; Sun et al., 2012) propose 'Self-Guided Artefact' as a holistic view of process instance. Each self-guided artefact (sg-artefact) contains process instance data and its process model. A workflow system can understand the sg-artefact and execute the process instance. A data-oriented view of the process through artefacts is certainly relevant to our topic. However, we see major differences in that (i) a sg-artefact incorporates a process model, as we mentioned before BPM systems can use different process modelling languages and coupling the process instance to a specific modelling language makes it less interoperable, (ii) A sg-artefact does not cater for instance specific information such as Execution Path and Meta-Data.

Process Mining: Finally, a process mining technique is a possible method for building a holistic view of process instances in a heterogeneous and distributed environment. The ProM process mining framework (Van Der Aalst et al., 2007) uses an XML format to define a workflow log model. This model contains information about the business process, process instance and related data. This approach is useful when we are dealing with business processes with no formal process model to begin with. We see process mining as a bottom up approach and they have to be customized for each system. In many BPM systems, we already have a model to generate instances with (as described in our scenario in Customer Journey process). Instead of logging the activities and trying to analyse them later (a bottom-up approach), we are proposing to store the instance related information in a format that any BPM system can understand.

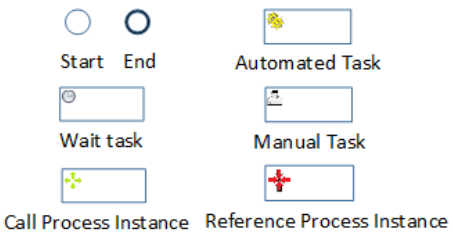
4 BUSINESS PROCESS INSTANCE MODEL

In this section, we introduce the individual components of Business Process Instance Model (BPIM). It consists of three views (i.e., dimensions): Process Instance Execution Path, Process Instance Data, and Process Instance Meta Data.

We will first present the *visual notations* of the model and then the *BPIM Meta Model* that describes the schema of BPIM elements.

4.1 Process Instance Execution Path

The *Process Instance Execution Path*, or *Execution Path* for short, focuses on describing the exact execution path that activities took in an instance. Unlike a process model, a Process Instance Execution Path



(a) *Activities in Execution Paths*

(b) *Transitions in Execution Paths*

Figure 2: BPIM Elements.

contains just the activities that have been performed during the process instance execution.

To be self-contained, we briefly show the main elements of Execution Paths. The design details of this component are in our previous work (Moghadam and Paik, 2015).

BPMN v2.0 (Object Management Group, 2011) comes with an interchange standard. Instead of creating new notations and their semantic from scratch, we use a subset of BPMN elements and added new elements that are relevant to the runtime information.

As shown in Figure 2a, besides the tasks, the *Wait* element indicates that process instance execution is suspended. *Call Process Instance* and *Reference Process Instance* specify that during the current process instance execution, a message or event has been sent to/received from another process instance.

Figure 2b shows transition elements. Transition connects two activities and shows the direction of the process execution flow. Transition also has another responsibility in the Execution Path. It shows how many times the execution engine has passed through it during the execution.

4.2 Process Instance Data

Process instance data contains information relating to the goal the corresponding instance aims to fulfil. Here, we first examine the data structure characteristics of process instance data. Then, we introduce the *Process Instance Data Snapshot*, or *Data Snapshot* for short, *Data Snapshot Graphs* and *Data Snapshot Pools* and their visual notations.

4.2.1 Data Elements in a Process Instance

Different types of data exist in a process instance. These data types can be grouped into the following categories:

- Basic Data Types: Each BPM system comes with built in data types (e.g., byte, integer, character) for defining process instance variables (Qin and Fahringer, 2012).
- Complex Data types: A complex data structure is composed of basic data types as attributes and builds a new data type (e.g., business entities, documents).
- Arrays: An array contains a collection of data with the same or different data types (e.g., basic or complex data item or another array). For example, in the `Customer Payment` process 'Discount Entitlements' is an array.

During the process instance enactment, activities in the Execution Path can introduce new data or modify the existing instance data. Each activity in the Execution Path may define input or output data items.

Table 1 defines the input and output for the activities in the Execution Path. None of the transitions in the Execution Path modify the data items, so they do not have data input/output and they are not listed here. Also, note that 'End' and 'Wait' activities do not change the instance data, these activities have no data input or output.

Table 1: Input/output for activities in the Execution Path.

Activity Name	Input	Output
Start	✗	✓
End	✗	✗
Automated Task	✓	✓
Manual Task	✓	✓
Wait	✗	✗
Call Process Instance	✓	✗
Reference Process Instance	✗	✓

Keeping track of the changes in the process instance data before and after the execution of an activity can be valuable. Chebotko et al. (Chebotko et al., 2010) states that data provenance management is an essential component for interpreting the result, diagnosing errors and reproducing the same result in scientific workflows. Although most of researches have focused on the data provenance in the scientific workflows, recently the same concepts are being applied to industrial systems. Shamdasani et al. (Shamdasani et al., 2014) discusses the usefulness of data provenance in the BPM systems and proposes a workflow system which can store provenance data.

BPIM, similar to scientific workflows, stores the data provenance as *Data Snapshots and Graphs* and *Data Snapshot Pools*. Each Data Snapshot of a process instance represents the state of data items at a specific point of execution (i.e., before or after execution of an activity in the Execution Path). *Data Snapshot Graphs* capture the transition of Data Snapshots by the activities.

A Data Snapshot Pool is a repository of all Data Snapshots and each snapshot is identified by a unique id. Each node in a Data Snapshot Graph contains this unique id which points to the actual instance of the Data Snapshot. The Data Snapshot Pool helps the Data Snapshot Graphs to share the same Data Snapshots across different nodes in the graph.

4.2.2 Data Snapshots and Graphs

We present the details of Data Snapshots and Graphs along with their the visual notations. The Data Snapshot Graph is a directed graph and it shows:

1. The state of data before and after the execution of each activity in the Execution Path
2. The flow of data items between activities during the process instance execution
3. Any errors and faults that occurred before or after the execution of an activity

Figure 3 shows the visual notations. To explain:

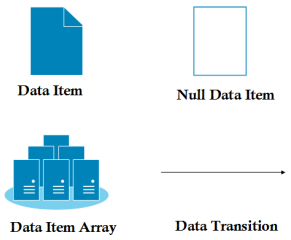


Figure 3: Data Snapshot and Graphs.

Data Item: The basic and complex data types are represented by data item notation. Data item displays a text and a number. The text is data item's name and number is data item's version.

Data Item Array: This notation represents an array of basic or complex data types. Data item array similar to data item displays the data item array's name and version number.

Data Transition: Each activity in the Execution Path maps to a transition in the data snapshot graph. When an action changes the data during the execution, it creates a brand new version of data item and connects the older version to the new version with a directed edge in the graph.

Null Data Item: Null data item is used in cases where the activity does not produce any output data nor require input.

To create a snapshot, we use the following rules:

- Each data item has an identifier.
- Each data item has a version number.
- When an activity in the Execution Path modifies the existing data item, it will create a brand new instance of that object with the same identifier and different version.
- When an activity in the Execution Path creates a new data item, it assigns an unique identifier and version to it.
- Initial version for all data items starts from 1.
- Data item identifier is a unique id which can be used to retrieved that object but there might be more than one instance of that object with different version number.
- Combination of item identifier and version makes an item unique.

4.3 Process Instance Metadata

BPIM also represents the information about the lifecycle of a process instance (i.e., creation, enactment and termination). Some of these information are merely informational (e.g., creation date time) and some of them are important for the execution engine (e.g., process instance state) to enact the instance.

We describe the Process Instance Metadata as a relation with the following tuple $\{id, name, modelId, creationDateTime, endDateTime, creator, server, state\}$, where *modelId* is the process model Id, *creator* is the name of the application or user created the process instance, *server* is the host which created the instance, *state* is the current state of the instance[3].

4.4 BPIM Meta Model

Along with the visual notations, BPIM defines a meta model using UML to formally describe the elements of the model as well as the schema for the elements. That is, the BPIM meta model is a model which describes all the elements in the Process Instance Execution Path and Data Snapshots and Pools.

4.4.1 Execution Path Meta Model

Figure 4 provides the meta model for the Execution Path elements.

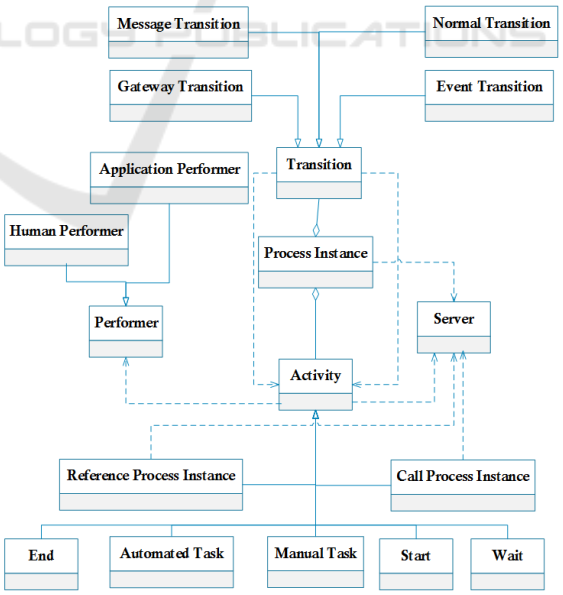


Figure 4: Process Instance Execution Path Meta Model.

In the following, we will closely look at the schema information for the elements in the Execution Path.

[3]From here on, we will skip descriptions of the attributes whenever the names themselves are descriptive enough.

Activities: All the activities in the BPIM Execution Path share some common attributes. We define them as a relation with a tuple {*id, name, startDateTime, endDateTime, performer, server, state, mappingCorrelationId*}, where *performer* is the name of the person/application executed the activity, *server* is the host which executed the activity, *mappingCorrelationId* is the correlationId to identify the target element in the execution language during the mapping process. The following tuple describe the additional attributes for each activity type in the Execution Path:

- **Automated Task:** {*serviceName, serviceURL, serviceGroup, applicationName, applicationId*}, where *serviceName* is the name of the service which BPM system calls. A service refers to any object which can process the instance data and provide a response (e.g., Java Object, Web service), *applicationName* and *applicationId* are the details of the application which hosts the service,
- **Manual Task:** {*userId, userName, role, comments, description, organisation, department*}, where *userId, userName, role* are the details of the user who performs the task, extra comments made and task descriptions are captured in *comments* and *descriptions* respectively.
- **Wait:** {*duration, expiryDateTime, interrupted*}, where *duration* is the period of time which process execution was suspended (ExpiryDateTime should be empty), *expiryDateTime* specifies the date and time which process instance execution can resume (Duration should be empty), *interrupted* specifies if Wait was interrupted
- **Call Process Instance:** {*targetInstanceId, targetActivityId, targetServer*}, where the details of the target process instance and activity are stored
- **Reference Process Instance:** {*sourceInstanceId, sourceActivityId, sourceServer*}, where the details of the source instance and activity are stored

Transitions: All the transitions in the Execution Path have some common attributes. We define them as a relation {*id, name, from, to, mappingCorrelationId, traverseCounter*}, where *mappingCorrelationId* is the correlationId to identify the target element in the execution language during the mapping process, *traverseCounter* shows how many times execution engine passed through this transition

In the following, the tuples describe the extra attributes for each transition type in the Execution Path:

- **Event Transition:** {*eventId, eventType, eventName*}, where *eventType* refers to the type of the event (e.g., Message, Timer)
- **Message Transition:** {*messageId, messageName*}, where the message Id and Name are message details
- **Gateway Transition:** {*gatewayId, gatewayType, gatewayName*}, where the gateway Id, Type and Name are gateway details

4.4.2 Process Instance Data Meta Model

The process instance data meta model describes the structure and schema of Data Snapshots and Graphs. Figure 5 provides the meta model for the elements.

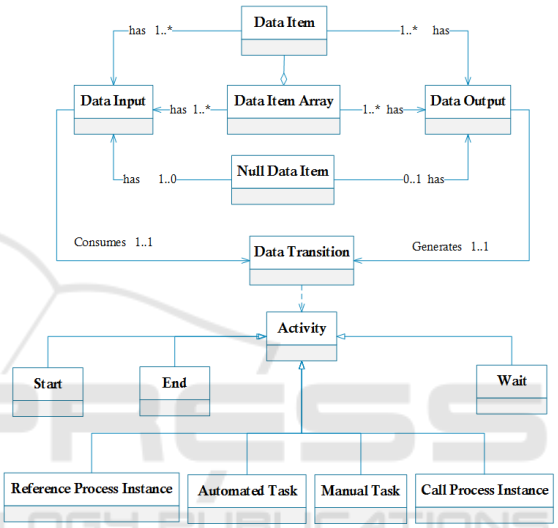


Figure 5: Process Instance Data Snapshots and Graphs Meta Model.

As discussed in Section 4.2.1, the process instance data elements can have basic or complex data types. Depending on the complexity of the data type, it can have different metadata (e.g., a document can have *author* and *size* attribute, a Customer entity can have *name* and *address*). In this section, we only list the common attributes for these data types.

The following tuples describe the attributes for each element in the Snapshots and Graphs:

- **Data Item:** {*id, dataItemObject, version, creationDateTime, type*}, where *dataItemObject* is a reference to a data object.
- **Data Item Array:** {*id, dataItemArrayObjects, version, creationDateTime, size*}, where *dataItemArrayObjects* is a reference to an array of data objects, *size* is the number of items it.
- **Data Transition:** {*id, activityId, dataInput, dataOutput*}, where *activityId* refers to an activity in the Execution Path.
- **Data Input:** {*id, dataElementIds*}, where *dataElementIds* specifies the transition inputs.

- **Data Output:** {*id, dataElementIds*}, where *dataElementIds* specifies the transition outputs.

5 APPLICATION OF BPIM

In this section, we present how BPIM is applied to the customer journey process scenario.

5.1 The Customer Journey Process

We use the customer journey process presented in Section 2 and create a process Execution Path, Data Snapshots and Data Snapshot Pools. In doing so, we assume that: (i) the BPM systems involved have adopted the BPIM framework and implemented it, (ii) they have added new functionality to support calling a process instance which is hosted by the other tools. We also assume that BPM systems share the same process instance repository.

5.1.1 The Execution Paths

Figure 6 shows an Execution Path of a customer journey process instance, made up of two Execution Paths from the sub processes: one from `Get Customer Account` process instance in jBPM, the other from `Customer Payment` process instance in Riftsaw.

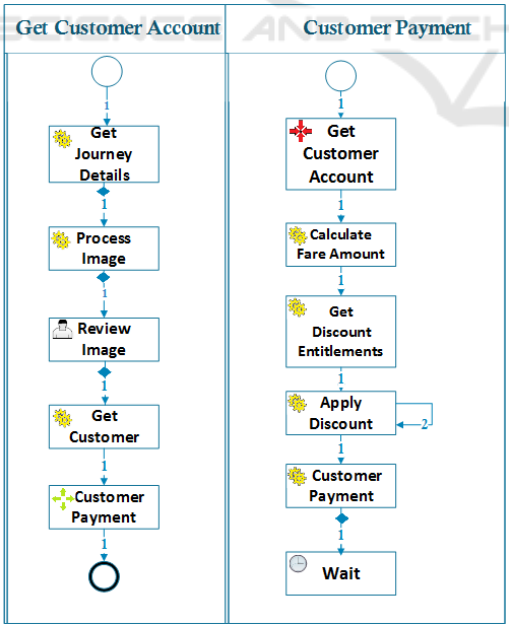


Figure 6: An Execution Path of a customer journey instance.

The `Get Customer Account` Execution Path displays all the steps taken to retrieve a customer account. After loading the customer account, jBPM execution engine sends a message to Riftsaw to create a new `Customer Payment` process instance and terminates. The 'Call Process Instance' activity is a link between these two instances. The `Customer Payment` Execution Path also has a corresponding 'Reference Process Instance' activity which points to the `Get Customer Account` process instance.

All the transitions in the Execution Path are marked with a number. This number shows how many times the execution engine has traversed that transition. In this example 'Apply Discount' node has two input transitions which are marked with 1 and 2. From there, we know that there are three discount entitlements applied for this journey. The example also shows that the `Customer Payment` Execution Path has no 'End' activity for this process instance. This means this process is not finished yet and it is waiting to try to call payment service again.

5.1.2 The Data Snapshots

Similar to the Execution Paths, we have two Data Snapshots from `Get Customer Account` and `Customer Payment`. Figure 7 displays these two side by side.

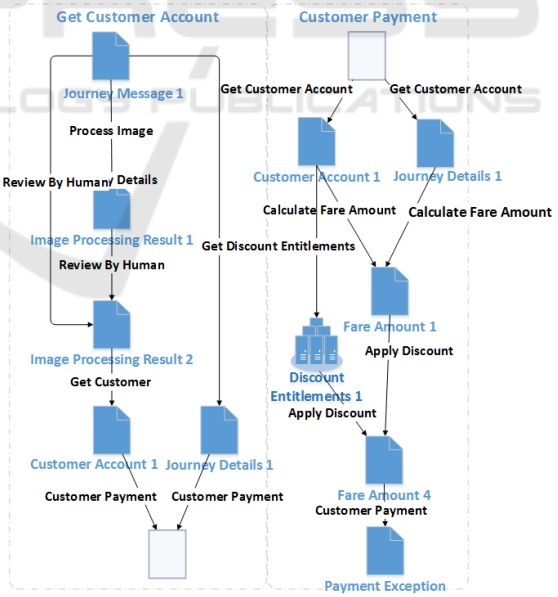


Figure 7: Data Snapshots from a customer journey instance.

The number beside the description of each data object is the version number. Having a version number helps to distinguish multiple versions of the same object. For example there are three discount entitlements for this customer and each one individually applies to the fair amount. As a result of that we end

up with four versions of 'Fair Amount' entity. In order to simplify the notations, if an activity (e.g. for each loop) produces multiple versions of the same data item we just display the latest version of that data item. All the intermediary versions of the data item exist in the Data Snapshot Pool.

Figure 8 illustrates the Data Snapshot Pool for the `Customer Payment` process instance.

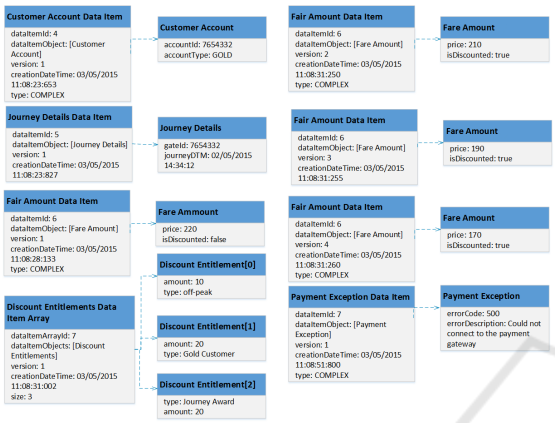


Figure 8: Customer Payment Process Instance Data Snapshot Pool.

BPIM makes it possible for the BPM systems to share the same schema and data. Using a standard model for process instance, removes the need for E-Toll application's private database. It also makes it easy to diagnose an error and because it keeps the data snapshots, it is possible to rollback the changes and restore the process instance to a specific point during the execution.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed an interoperable model which provides a holistic view of process instances. The model is designed to capture process execution paths, instance data provenance and process context metadata. This model may be adopted by BPM systems to work as an abstraction layer between execution engine and physical storage. This helps BPM systems share their process instances with each other.

Currently, we are working to provide the full mapping between the elements in the BPMN and BPEL to/from BPIM elements and realise a full transformation algorithm. A prototype will be developed to show how all these components work together and build a holistic view of process instance information.

REFERENCES

Aguilar-Saven, R. S. (2004). Business Process Modelling: Review and Framework. *International Journal of Production Economics*, 90(2):129–149.

Chebotko, A., Lu, S., Fei, X., and Fotouhi, F. (2010). Rdfprov: A relational rdf store for querying and managing scientific workflow provenance. *Data & Knowledge Engineering*, 69(8):836–865.

Choi, I., Kim, K., and Jang, M. (2007). An XML-based Process Repository and Process Query Language for Integrated Process Management. *Knowledge and Process Management*, 14(4):303–316.

Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., and Shan, M.-C. (2004). Business Process Intelligence. *Computers in Industry*, 53(3):321–343.

Grigorova, K. and Kamenarov, I. (2012). Object Relational Business Process Repository. In *Proceedings of the 13th Int'l Conference on Computer Systems and Technologies*, pages 72–78. ACM.

Ma, Z., Wetzstein, B., Anicic, D., Heymans, S., and Leymann, F. (2007). Semantic Business Process Repository. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007)*.

Moghadam, N. and Paik, H.-y. (2015). Bpim: A multi-view model for business process instances. In *Proceedings of the 11th Asia-Pacific Conference on Conceptual Modelling (APCCM 2015)*, volume 27, page 30.

Object Management Group (2011). Business Process Model And Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0/.

Qin, J. and Fahringer, T. (2012). Semantic-based scientific workflow composition. In *Scientific Workflows*, pages 115–134. Springer.

Shamdasani, J., Branson, A., McClatchey, R., Blanc, C., Martin, F., Bornand, P., Massonnat, S., Gattaz, O., and Emin, P. (2014). Cristal-ise: Provenance applied in industry. *arXiv preprint arXiv:1402.6742*.

Sun, Y., Su, J., and Yang, J. (2014). Separating execution and data management: a key to business-process-as-a-service (bpaas). In *Business Process Management*, pages 374–382. Springer.

Sun, Y., Xu, W., Su, J., and Yang, J. (2012). Sega: A mediator for artifact-centric business processes. In *On the Move to Meaningful Internet Systems: OTM 2012 Workshops*, pages 658–661. Springer.

Van Der Aalst, W. M., Reijers, H. A., Weijters, A. J., van Dongen, B. F., Alves de Medeiros, A., Song, M., and Verbeek, H. (2007). Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732.

Yan, Z., Dijkman, R., and Grefen, P. (2012). Business Process Model Pepositories – Framework and Survey. *Information and Software Technology*, 54(4):380–395.

Zaplata, S., Hamann, K., Kottke, K., and Lamersdorf, W. (2010). Flexible Execution of Distributed Business Processes Based On Process Instance Migration. *Journal of Systems Integration*, 1(3):3–16.