# BPIM: A Multi-view Model for Business Process Instances

**Nima Moghadam and Hye-young Paik**

School of Computer Science and Engineering
University of New South Wales
Sydney, NSW, 2052 Australia
Email: {nimam,hpaik}@cse.unsw.edu.au

## Abstract

Business process management has grown into a mature discipline supported by a large number of commercial and open source products, collectively referred to as Business Process Management Systems (BPMS). Because most of the business processes are long-running, any data relating to a running instance of a business process need to persist in a data storage. In an organisation several BPMS products can co-exist and work along side each other. Each one of these BPMS tools has its own definition of process instances, creating a heterogeneous environment. This reduces interoperabilities between business process management systems and increases the effort involved in analysing the data. In this paper, we propose a multi-view business process instance model (BPIM) which provides a holistic view of business process instance. BPIM has three dimensions; process execution path, process instance meta-data and process instance data. This model aims to provide an abstract layer between the data storage and BPM engines, leading to common understanding of what is involved in storing business process instance data.

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

## 1 Introduction

Business process is a collection of related activities performed together to fulfil a goal in an organisation (Aguilar-Saven 2004).

In the last decade, the growing number of business processes and their increasing complexity led to the creation of Business Process Management (BPM) in which standard approaches for design, enactment, analysis and management of business processes are studied (van der Aalst et al. 2003). The area is supported by a range of software frameworks and tools, collectively referred to as Business Process Management Systems (BPMS) (Jeston & Nelis 2014).

One of the main functions of a BPMS 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 process model that spawned the executable and (ii) materialised data (e.g., Customer Name, Order Number) involved. For example, given a business process model describing a car insurance claim process, BPMS would enact concrete instances of the model, each instance representing an actual claim being processed and the details of the data involved (e.g., claim number, customer name, dates, amount claimed).

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 lifecycle 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 for an action or an event, its current state information is written into a permanent storage. BPM system maps process instance information directly to physical storage artefacts (i.e., table, row, xml, text) and stores it. At this stage process instance is still alive but just temporarily suspended. When BPM system needs to resume processing a live process instance, it retrieves the instance information from the repository and continues the process. Besides suspending/resuming live process instances, BPM systems use physical storage to store other information such as process instance execution logs. Organisations can use this data to analyse and improve their business processes (Grigori et al. 2004).

In an organisation several BPM systems can coexist and work along side each other. Some business processes, due to complexity or technology limitations, are implemented by using multiple BPM systems. Each one of them has its own definition of process instance, creating a heterogeneous environment. To build a comprehensive view of a process instance which spans across multiple BPM systems, we need to fully understand each product's physical storages (e.g., log files, database) and the schema they are using to store the process instance information. Diversity of business process instance model in BPM systems introduces a new challenge for creating a holistic view of process instances which are implemented across multiple BPM systems. It also makes sharing information between process instances challenging due to interoperability problems between process instance models.

In this paper:

1. We propose a multi-view business process instance model (BPIM) which provides a holistic view of business process instance. BPIM has three dimensions; process execution path, process instance data and process instance meta-data.

2. We illustrate how BPIM enables different BPM systems to share the same process instance repository and re-use the exiting process instance information which was created by other BPM systems.

3. We discuss the changes in the BPM systems architecture after adopting BPIM.

This paper is organized as following. Section 2 introduces the customer journey process in a tolling system in detail to discuss the motivation and challenges for creating a holistic view as a standard process instance. In section 3 we will discuss the related researches about process instance model. We introduce a high-level overview of our solution in section 4. In section 5 we provide a detailed description about our solution and its components. Also we will demonstrate how BPIM can solve the problem of creating holistic view of a process instance and sharing information. Finally we will discuss the advantages of using BPIM and future work in section 6.

## 2 Motivating Example and Problem Background

We present a customer journey process as a motivating scenario. The scenario depicts a road toll system operation and is divided into two sub-processes: `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 shown in Figure 1, the customer journey process starts with receiving `Journey Message` from a toll gate. If the message includes a transponder ID, the `Get Customer Account` sub-process extracts necessary information using the ID and produces 'Customer Account' and 'Journey Details' as output. If the ID is not available, the process goes through image processing and human reviewing steps to extract necessary information for the output messages. The `Customer Payment` sub-process takes 'Customer Account' and Journey Details' entities as input, calculates the final fare to be paid (e.g., considering any discount that may apply) and processes the payment.

The main implementation concern for each process in the BPM systems is to precisely describe the model using a process description and execution language. This language can be different from product to product. For example, in jBPM the `Get Customer Account` process is described as follows (Figure 2 showing snippets of BPMN Interchangeable Language (Object Management Group 2011) for jBPM).

Figure 3 displays snippets of BPEL execution language[3] which Riftsaw uses to formally describe the activities involved in the `Customer Payment` process.

A BPM execution engine needs a process blueprint (i.e., process models) and data to create process instance. As shown above, BPM products use either interchangeable (such as BPMN) or executable languages (such as BPEL) to formally represent their process blueprints. In this research we will refer to both of these languages as business process execution language, as both eventually lead to business process instances.

[1]Open Source Business Process Management System, www.jbpm.org

[2]RiftSaw Open Source BPEL, riftsaw.jboss.org

[3]Web Services Business Process Execution Language, docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

```
<?xml version="1.0" encoding="UTF-8"?>
        <bpmn2:definitions
         xmlns:bpmn2="http://www.omg.org/spec/BPMN
              /20100524/MODEL"
         ...>
  <bpmn2:process id="defaultPackage.
      GetCustomerAccount"
         tns:version="1" name="GetCustomerAccount"
         isExecutable="true" processType="Private">
         <bpmn2:startEvent id="StartEvent_1"
           name="StartProcess">
         ...
         <bpmn2:exclusiveGateway
           id="Check_Message_Payload_Type_1"
           name="Check Payload Type"
           gatewayDirection="Diverging">
         ...
         <bpmn2:serviceTask id="ServiceTask_2"
           name="Extract Transponder Id">
         ...
         <bpmn2:serviceTask id="ServiceTask_1"
            name="Process Image">
         ...
         <bpmn2:endEvent id="EndEvent_1" name="End
              Process">
         ...
  </bpmn2:process>
```

Figure 2: Get Customer Account in *BPMN Interchangeable Language.*

When required (e.g., a long wait), each process instance is then transformed and stored to a physical storage by its BPM execution engine. For example, an instance of the customer journey process would be stored in jBPM (as part of `Get Customer Account` sub-process) and Riftsaw (as part of `Customer Payment` sub-process). In this case despite the fact that both of these products are using RDBMS database, the underlying data models to represent and store the process instance information are very different. Such differences could be as vast as one using XML (e.g., Riftsaw) and the other using a combination of database records, binary objects and log files (e.g., jBPM).

Both `Get Customer Account` and `Customer Payment` sub-processes are logically part of `Customer Journey` process but the are implemented as individual separated processes. `Customer Payment` process instance depends on the data generated by
tt Customer Journey. In this scenario both BPM systems are sharing the same process instance repository but as a result of differences in the process instance model they can not share the data. jBPM needs to transform and send the 'Customer Account' and 'Journey Details' entities via a different channel (e.g., Web Service call) to `Customer Payment` process instanse.

Let us assume that an instance (with `Journey Message ID 11123`) of our customer journey process has not completed after a day, and the Finance department wants to know why payment is not done yet. This is a live process instance which is stretched out across two BPM tools and operation team needs to investigate the root cause of this issue. Such an investigation requires a construction of holistic view of the instance 11123 took as well as the data associated with the instance 11123. This typically involves extracting and coordinating pieces of information from each sub process.

In our scenario, this means investigating first the log files of `Get Customer Account` process in jBPM to check if there was any failure, or extracting the relevant `Customer ID` produced at the end of `Get Customer Account` process. The ID is then used to correlate the activities in the `Customer Payment` pro-

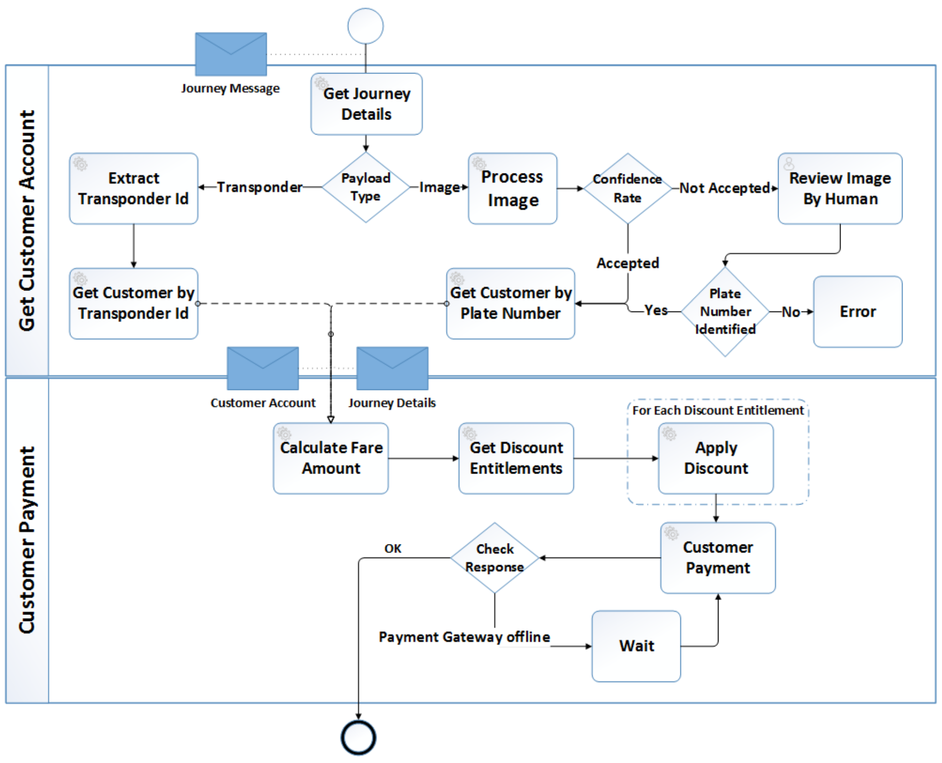


Figure 1: *Customer Journey Process Implemented by Two Sub-processes: Get Customer Account, Customer Payment*

cess from Riftsaw data tables. Only after interrogating the database records, we may discover a reason for failure and where the process is at (e.g., the payment gateway was off-line due to maintenance, so withdrawing money failed. The execution engine placed the process instance into wait state to retry later on).

```
2014-08-18 15:18:23:456 Before Executing Check
     Image Type Node. {JourneyId:11123}
2014-08-18 15:18:23:459 After Executing Check Image
     Type Node. {Result:Image}
2014-08-18 15:18:23:460 After Executing Process
     Image Node. {ConfidenceRate:2}
2014-08-18 15:18:23:461 Before Executing Process
     Image Node. {JourneyId: 11123}
2014-08-18 15:18:25:187 After Executing Process
     Image Node. {ConfidenceRate:2}
2014-08-18 15:18:25:188 Before Executing Check
     Confidence Rate. {ConfidenceRate:2}
2014-08-18 15:18:25:189 After Executing Check
     Confidence Rate. {Result:Not Accepted}
2014-08-18 15:18:25:190 Before Executing Review
     Image. {JourneyId:11123}
2014-08-18 15:18:25:900 After Executing Review
     Image. {PlateNumber:BPM}
2014-08-18 15:18:25:190 Before Executing Plate
     Number Identified. {PlateNumber:BPM}
2014-08-18 15:18:25:900 After Executing Plate
     Number Identified. {Result:Yes}
2014-08-18 15:18:25:190 Before Executing Get
     Customer. {PlateNumber:BPM}
2014-08-18 15:18:25:900 After Executing Get
     Customer. {CustomerId:678989}
```

Figure 4: jBPM Log File.

```
SELECT  BPAF_EVENT.EID, BPAF_EVENT.
    PROCESS_INSTANCE_ID
        , BPAF_EVENT.ACTIVITY_NAME
        , BPAF_EVENT_DATA.NAME
        , BPAF_EVENT_DATA.VALUE
FROM    BPAF_EVENT INNER JOIN
        BPAF_EVENT_DATA ON BPAF_EVENT.EID =
            BPAF_EVENT_DATA.EVENT_ID
WHERE (BPAF_EVENT_DATA.VALUE like 'CustomerId =
    678989')
ORDER BY BPAF_EVENT.EID
```

Figure 5: Sql Query to Retrieve Process Instance Information from Riftsaw Database

This type of investigation takes time due to the sheer amount of data in log files and also storing the process instance data as a string in the database.

To summarise, BPM products use different data structures (e.g., Data Table, RDF, XML) to model the business process instance (Choi et al. 2007), (Grigorova & Kamenarov 2012), (Ma et al. 2007). This has led to an inconsistency in the understanding and analysis of process instances and sharing process instancer repository. A main problem which adds significantly to this inconsistency is each BPM system has different interpretation of process instance information. For example some of the BPM systems automatically store the history of activities which have been performed during process instance execution whereas some others place this responsibility on the developers to log the activities themselves.

We see the following problems in the current state-of-the-art in terms of representing and utilising process instance data:

1. Having different definition of process instance models in each BPM product.

2. Having to analyse multiple sources (e.g., logs, data tables) to extract complete process instance

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process exitOnStandardFault="yes"
    name="CustomerPayment.bpel" ...>
    <bpel:import importType="http://schemas.xmlsoap
        .org/wsdl/"
        location="CustomerPayment.wsdl"
        namespace="http://www.unsw.edu.au/research/
            CustomerPayment/"/>
    ...
    <bpel:sequence name="MainSequence">
     <bpel:receive name="start"
     partnerLink="bpelProcessPartner"
     portType="ns0:CustomerPayment"
     operation="WithdrawMoney"
     variable="withdrawMoneyRequest"
     createInstance="yes"/>
     <bpel:invoke name="Calculate Fare Amount">
     ...
     </bpel:invoke>
     <bpel:invoke name="Get Discount Entitlements">
      ...
     </bpel:invoke>
     ...
     <bpel:reply name="end"
     partnerLink="bpelProcessPartner"
     portType="ns0:CustomerPayment"
     operation="WithdrawMoney"
     variable="withdrawMoneyResponse"/>
     </bpel:sequence>
</bpel:process>
```

Figure 3: Customer Payment in *BPEL Execution Language*

information.

3. 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.

4. Direct coupling of process instance model to physical storage model.

To solve the problems mentioned above, we propose a new business process instance model for live process instances which BPM systems can adopt and use. This model defines a framework for process instance information; it defines the data elements which build a process instance and how we can represent them to demonstrate a holistic view of a process instance. BPIM aggregates all the relevant information to a process instance in one place and makes it easier to display a holistic view of the process instance. Using an abstract model for process instances de-couples the BPMS execution engine from a physical storage.

## 3 Related Works

Most of the academic research work so far have focused on business process modelling and process model repository (Yan et al. 2012). In-depth discussions about models for business process instances have been largely neglected. The most relevant streams of academic work can be classified as follows:

Models and systems to manage business process repositories are proposed. A business process repository stores the artefacts generated by BPM systems. IPM, an XML-based process repository proposed in (Choi et al. 2007), provides support for a full lifecycle of business process. To store the information created in each stage, it divides the repository internally into smaller sections: Process Model Repository, Process Rule Repository, Process Resource Repository, Process instance Repository and Process Knowledge Repository. To the best of our knowledge, IPM does not provide any information about how process instance is modelled in the repository and it stays on the high level requirements. An object relational business process repository is presented in (Grigorova & Kamenarov 2012) where object-based model for storing business process models is proposed. This paper divides the information in a process model into three categories. The first one is business process characteristics (i.e., description, responsible units). The second category is the inter-dependency between business process models (i.e., relationship between process and sub-processes). The third category maintains information about process model's graph structure. This framework transforms different process models into Event-driven Process Chain (EPC) model and stores it in an object-oriented database. In fact EPC acts as an abstract model which sits between process model designer and model repository. (Grigorova & Kamenarov 2012) only focuses on the process model not the process instance model.

There has been some effort that looks into interoperabilities between business process execution languages. Zaplata et al. (Zaplata et al. 2010) discuss the possible ways of process instance execution in a distributed environment. They have identified BPEL and XPDL as the most popular execution languages and conducted a comprehensive study on the elements in these languages. This paper proposes a model for process instance, the proposed model is flexible enough to be used for both XPDL and BPEL. A BPMS execution engine (a source environment) uses this model to transform its native process instance to an interoperable instance and sends the information to another BPMS execution engine (a target environment). The proposed process instance model in this paper only focuses on the migration aspects and does not provide the holistic view a process instance. Hornung et al. (Hornung et al. 2006) investigates merging different business process execution languages and related problems. This paper focuses on merging XPDL and BPEL execution languages and proposes a four-staged BPM meta-model integration process. The final result of this process is a merged schema of XPDL and BPEL. This model can only be used as an interoperable process execution language in design time and it does not support runtime information about the process instance (e.g., when process instance was created). La et al. (La Rosa et al. 2011) suggests Advanced Process Model Repository (APROMORE) which tries to solve the problem of storing business process models by proposing a canonical model which all types of process models (e.g., BPMN, BPEL, YAWL) can be transformed to/from it. The APROMORE's canonical model also only addresses the design aspects of business process (Process Model) not the runtime (Process instance).

Finally, a process mining technique is a possible method for building a holistic view of process instances in a heterogeneous and distributed environment. Process mining is a new brand of data mining which has been developed to extract process information from event logs stored in various sources (e.g., files, database, mail archives) (Van Der Aalst et al. 2012). For example, 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. Process mining techniques analyse these logs and try to build a clear picture of activities in a process instance and eventually shape a process model out of the information. 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 Solution Overview

In this section, we discuss the general idea behind our proposed solution.

As mentioned before, logging the activities and related data during the execution of process instance and extracting information from different sources (e.g., files, database records) are the preferred method by process mining to describe what happened during the process enactment (Grigori et al. 2004).

For the cases where process models are available (and many BPM systems do require a model to be described), we could take the model-first approach (top-down approach). We are proposing a multi-view business process instance model which BPM products can adapt and use it to organise live process instance information. We also discuss the impact of our design on the current BPMS architectures.

### 4.1 Live process instance

In the introduction section we defined live process instance as a process instance which has started and it is not finished yet. BPIM replaces the BPM systems internal process instance model. This means that it would be used by execution engine during the process instance enactment to store and retrieve live process instance information from instance repository. Unlike the process mining methods which:

1. Keep the information gathering about a process instance separate from BPM systems.
2. Normally start gathering the information after process instance finished.

BPIM is designed be part of BPM system and starts gathering information as ssoon as process instance starts. Also it can be used later on to provide detail information about process instance.

### 4.2 A Common Model for Process Instances

Business Process Instance Model aims to be a model for interoperable process instance data. In proposing this model, we aim to suggest that we should have a standardised understanding of a business process instance data. Using a standard ontology increases the interoperability between different BPM products and makes it possible to build business intelligence tools which can seamlessly integrate with different BPM systems.

BPIM presents a holistic view of a business process instance by integrating different 'views' of a process instance. It contains all the process instance information in a format which should be understandable by all BPM products. We have identified three views (i.e., dimensions) in this model, which are:

1. **Process instance execution path:** This is a visual dimension of the model and it describes information about the activities which have been executed during the process instance enactment.
2. **Process instance meta-data:** Provides extra information about the elements in the execution path and process instance data dimensions. For example it keeps when process instance was created, started and finished or which user has performed a manual activity in the execution path.
3. **Process instance data:** Process instance contains some related data (e.g., order no, customer details). Each activity in the execution path can modify this information. This dimension is focused on the data flow and in combination with the execution path dimension, it should keep snapshots of an instance during any stage of execution.

### 4.3 Impact on BPMS Architecture

BPIM changes the way BPM systems would interact with the underlying physical storage. BPIM adds an abstract layer between the process runtime engine and physical storage. This abstract layer gives BPM systems the flexibility of changing their physical storage technology without requiring any changes on their side. Another affect of this model is that BPM systems does not need to rely on a particular query language (e.g., SQL) to analyse the process instance data. We envisage in the future that BPIM, as a complete and mature solution, would provide an implementation-agnostic language that would interact with the BPIM model. Section 5.5 in this paper provides more detailed discussion about architectural effects of the proposed model.

## 5 Business Process Instance Model

In this section, we introduce the individual components of BPIM.

### 5.1 Process Instance Execution Path

The process instance execution path dimension of BPIM focuses on describing the exact execution path that activities took in an instance. Unlike a process model, which contains all possible actions and scenarios in a business process, a process instance execution path contains just the activities that have been performed during the process instance execution. In fact activities in the execution path are subset of activities in a process model.

The following items summarize the differences between these two:

1. A process instance execution path is designed to accommodate the runtime information, so some of the elements we normally see in the process model are removed or replaced by different types of elements. For example, BPMN elements such as 'sub-process' and 'pool' have been removed because in runtime only actions which have been executed are important and how we logically group them does not have any affect on the execution of a process instance.
2. A process model can use block and graph structures (Ko et al. 2009) but process instance execution path just uses graph structure to provide a clear view of the actions have been run during the execution. There is a growing number of tools that allow effective and efficient analysis of graph data and we believe choosing the graph data structure for accessing and analysing process instance data purposes would be a reasonable choice.

3. A process instance execution path is made of generic and simple elements. There are different execution languages which formally describes business process model (e,g,. BPEL, XPDL). Using a generic and simple model makes it possible to transform different types of execution languages to our process instance execution path.

BPMN v2.0 (Object Management Group 2011) comes with an interchange standard to formally describe the process model. This standard makes a BPMN process model interoperable and BPMS runtime engine can use it to build and execute the process instance.

In this research, instead of creating new notations and their semantic from scratch, we have used a subset of elements in the BPMN. However, we also have added new elements that are relevant to runtime information. Each element in the execution path has a visual representation. A visual model makes it easy to track the activities during the process execution.

### 5.1.1 Execution Path Elements: Activities.

All activities in this model are directly or indirectly inherited from the flow node class in the BPMN v2.0. Each activity in a process execution path represents an action which has been performed during the process execution.

There are different types of activities identified within a process execution path to separate different types of action that activities can represent. The rest of this section explains each type of activity and its functionality.

**Start:** A process instance execution path always starts with a start activity. Unlike the process model, each process instance execution path can only have one start element. This is because start element represents the creation point of a process instance and each instance creation happens only once.

**End:** End activity indicates that a process instance has reached the termination point. Each process instance execution path can have only one end element in it.

Figure 6: *Start and End Activities*

**Automated Task:** Represents an activity which has been performed by an application. For example, 'Process Image' activity in `Get Customer Account` process is an automated task.

**Mannual Task:** Represents an activity which has been performed by a human. For example, 'Review Image' activity in `Get Customer Account` process is a manual task.

**Wait:** Process execution might be suspended due to various reasons (e.g., waiting for external events or messages). The Wait element indicates that process instance execution was suspended. For example, in the `Customer Payment` process, if payment fails due to unavailability of the payment gateway, process execution goes to wait to retry later on.

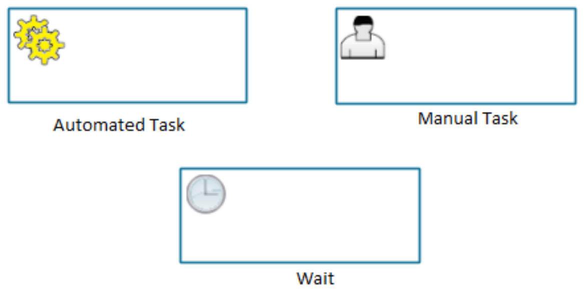


Figure 7: *Automated, Manual and Wait Tasks*

**Call Process Instance:** This element specifies that during the current process instance execution a message or event has been sent to another process instance. This call can be to an existing process instance or it could lead to creating a new instance. It is also a possible to call a process instance which is hosted by another BPM system. In this case it will contain information about the BPM system which is executing this process instance. we will discuss about this information in more detail in Section 5.3.

**Reference Process Instance:** A process instance during the execution can receive a message or event from another instance. The source instance in the execution path will be represented by this reference process instance activity.

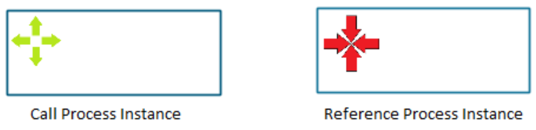


Figure 8: *Call and Reference Process Instance Activities*

### 5.1.2 Execution Path Elements: Transitions.

Transitions link the activities in a process instance execution path. Each transition connects two activities and shows the direction of the process execution flow. Transitions also have another responsibility, they show how many times execution engine has passed through them during the execution. This is because execution path can contain loops which are made of repetitive activities and transitions. To display the loops in the execution path model, all transitions have a number associated to them which shows the number of times they have been traversed.

**Normal Transition:** Connects two activities in the process execution path.

**Message Transition:** As we discussed in the activity types section, wait activity suspends the process instance execution until it receives a message or event signal. Message transition indicates that wait activity has received a message and moved on. This message could have been sent from inside the process instance or from another instance. Message Transition connects the node which generates that message to the wait node. If the wait activity is located in another process instance, message transition connects to call process instance activity.

**Event Transition:** Event transition connects the node which has generated this event to the first activity in the event handler chain. Event handler chain has been defined the process model and it contains at least one activity. The target node for event transition can exist inside the same process instance or in another instance. If target activity is in another process instance, event transition connects the source activity to call process instance activity.

**Gateway Transition:** Connects two activities in the execution path model. This transition represents that a decision has been made during the process instance enactment and the path which was chosen as a result of that decision. We map the decision node in the process model plus it's input and output transitions to one 'Gateway Transition'.

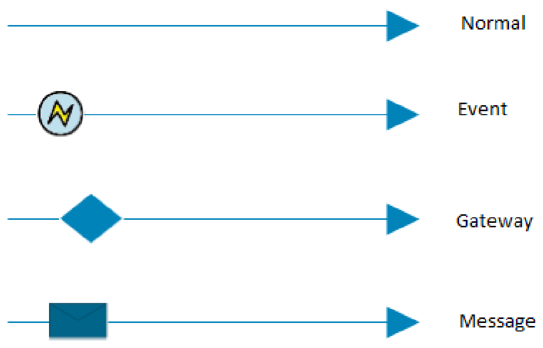


Figure 9: *Process Execution Model Transitions*

## 5.2 Process Instance Data Dimension

Process instance data contains information relating to the goal which this instance is trying to fulfil. In this section, we first examine the data structure characteristics of process instance data. In the next step we define the process instance data snapshot pool and graph. And finally we propose a model for process instance data snapshot graph and show how different data types will be handled by this model.

### 5.2.1 Data elements in Process Instance

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

1. Basic Data Types: Each BPM system comes with built in data types for defining process instance variables (Qin & Fahringer 2012). Table 1 displays some of the basic data types.

| | | |
|---|---|---|
| byte | boolean | date |
| integer | float | time |
| character | string | object |

Table 1: Basic data types in a process instance

2. Complex Data types: Some information in a process instance have more complex data structure (e.g., Business entity, document). Complex data structure is composed of basic data types as attributes and builds a new data type. Table 2 lists customer entity's attributes in the `Get Customer Account` process.

| Attribute Name | Data Type |
|---|---|
| id | string |
| firstName | string |
| lastName | string |
| isActive | boolean |

Table 2: Customer Entity

3. Arrays: An array contains collection of data with the same or different data types (e.g., Basic or complex Data Item or another Array ). In the `Customer Payment` process discount entitlements is an array.

### 5.2.2 Process Instance Data Snapshot Pool

During the process instance enactment, activities in the execution path can modify the instance data. Keeping track of process instance data changes before and after execution of each activity can be very 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 the scientific workflows. Although most of researches have focused on the data provenance in the scientific workflows, but recently researchers are investigating applying the same concepts 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. BPIM keeps snapshots of data during the process instance execution. Each snapshot of data represents the state of data items in a process instance at a specific point of execution(i.e., before or after execution an activity in the execution path). Process Instance Data Snapshot Pool (PIDSP) contains all the data snapshots. We will use the PIDSP in the next section to create a graph model for the data provenance in a process instance. PIDSP acts as a repository for the data snapshots and each snapshot is identified by an unique id. Each node in the data provenance graph contains this unique id which points to the actual instance of that snapshot. PIDSP helps the data provenance graph to share the same snapshot across different nodes in the graph.

To create a data snapshot before and after execution of an activity we need to understand the behaviour of that activity in the execution path. Each activity in the execution path might define input or output data items. Table 5.2.2 defines the input and output for the elements in the execution path. None of the transitions in the execution path has any effect on the data items so they don't support data input/output and they are not listed in the table 5.2.2.

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

Table 3: Data input/output for activities in the execution path

As we can see in the table 5.2.2 Start, End and Wait do not change the instance data, these activities have no data input or output.

BPIM after executing an activity and before running the next one stores the new data items in the PIDSP.

#### 5.2.3 Process Instance Data Snapshot Model

BPIM data dimension provides visual notations to display the changes on the data items in the PIDSP during the process instance execution. The Data snapshot notations build a directed graph which is called PIDS graph. PIDS graph displays:

1. State of data before and after execution of each activity in the execution path
2. Flow of data items between activities during the process instance execution
3. Errors and faults which occurred before or after execution of an activity

Below we will describe the visual notations in the PIDS graph:

**Data Item:** The basic and complex data types are represented by data item notation. Displays a text and a number. Text is the activity's name and number is the 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 activity's name and data object's 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 process instance 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 PIDS graph. Similar to transitions in the execution path, the data transition is also have number associated to it. This number specifies how many times that data transition (i.e., activity) is traversed during the execution.

**Null Data Item:** Some activities in the execution path don't have input or output data. Null Data Item is used in this cases to show that the activity doesn't produce any output data or doesn't require input data.

To be able to create snapshot of data we are using the following rules in the BPIM framework:

1. Each data item has an identifier
2. Each data item has a version number.

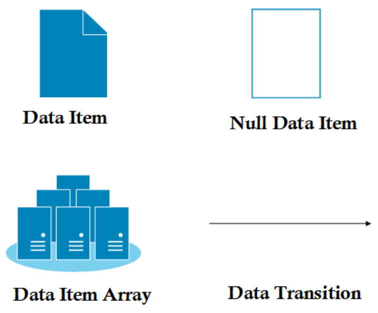


Figure 10: *PIDS Graph Notations*

3. 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.
4. When an activity in the execution path creates a new data item, it assigns an identifier and version to it.
5. Initial version for all data items start from 1
6. 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.
7. Combination of item identifier and version makes an item unique
8. Data item array is made of references to data items and version number. Each time an activity changes a referenced data item, a new version of data item array gets created which point to the new version of data item.

### 5.3 Business Process instance Metamodel Dimension

The meta model dimension formally describes the BPIM model. The BPIM metamodel is a model which describes all the elements in the process instance execution path and instance data dimensions. It also contains metadata information for the artefacts in the process instance model (e.g., creation date and time for process instance, or size of a document, the actor of an activity).

In the following, we will show what kind of metadata is included in the process instance metamodel dimensions to compliment the instnace execution path and data dimensions.

#### 5.3.1 Execution Path Metamodel

In this section we will closely look at the metadata information for the elements in the execution path model and propose a meta model which formally describes this model.

**Process Instance Metadata:** It contains information about the process instance life-cycle (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 execution engine (e.g., process instance state) to enact a process instance. Table 5.3.1 lists all the process instance attributes.

| Attribute Name | Description/Usage |
|---|---|
| id | Process instance Id |
| name | Process instance name |
| modelId | Process model Id |
| creationDateTime | Date and time which process instance created |
| endDateTime | Date and time which process instance ended |
| creator | Task instance creator |
| server | Server which created the process instance |
| state | Current state of process instance |

Table 4: Process Instance Attributes

**Activities Metadata:** All the activities in BPIM have some attributes in common. Table 5.3.1 displays these attributes.

| Attribute Name | Description/Usage |
|---|---|
| id | Activity Id |
| name | Activity name |
| startDateTime | Date and time which activity started |
| endDateTime | Date and time which activity ended |
| performer | Person or application which executed the activity |
| server | Server which executed the activity |
| state | Current state of activity |

Table 5: Activities Common Attributes

Some of the activities in the execution path have extra attributes which only belongs to them. The following tables show the individual attributes for each activity.

| Attribute Name | Description/Usage |
|---|---|
| serviceName | Name of the service which BPM system calls. Service refers to any object which can process the instance data and provide a response back(e.g., Java Object, Web service). |
| serviceURL | Address of the service which BPM system calls |
| serviceGroup | Service group name |
| applicationName | Application name which is hosting the service |
| applicationId | Application Id which is hosting the service |

Table 6: Extended Attributes for Automated Task

| Attribute Name | Description/Usage |
|---|---|
| userName | User name |
| userId | User Id |
| role | Role of user |
| comments | Extra comments |
| description | Task description |
| organisation | Organisation name |
| department | Department name |

Table 7: Extended Attributes for Manual Task

| Attribute Name | Description/Usage |
|---|---|
| duration | The period of time which process execution was suspended. ExpiryDateTime attribute should be empty |
| expiryDateTime | Specifies the date and time which process instance execution can resume. Duration attribute should be empty |
| interrupted | Wait was interrupted |

Table 8: Extended Attributes for Wait

| Attribute Name | Description/Usage |
|---|---|
| targetInstanceId | Target process instance Id |
| targetActivityId | Activity Id in the target process instance |
| targetServer | Server address which hosts the target process instance |

Table 9: Extended Attributes for Call Process Instance

| Attribute Name | Description/Usage |
|---|---|
| sourceInstanceId | Source process instance Id |
| sourceActivityId | Activity Id in the source process instance |
| sourceServer | Server address which hosts the source process instance |

Table 10: Extended Attributes for Reference Process Instance

**Transitions Metadata:** All the transitions in BPIM have some attributes in common, Table 5.3.1 displays these attributes.

| Attribute Name | Description/Usage |
|---|---|
| id | Transition Id |
| name | Transition name |
| from | Source activity |
| to | Destination activity |

Table 11: Transitions Common Attributes

Some of the transitions in the execution path have extra attributes which only belongs to them. The following tables show the individual attributes for each transition.

| Attribute Name | Description/Usage |
|---|---|
| eventType | Type of event(e.g., Message, Timer) |
| eventName | Event name |
| eventId | Event Id |

Table 12: Extended attributes for event transition

| Attribute Name | Description/Usage |
|---|---|
| messageId | Message Id |
| messageName | Message name |
| messageData | Message data |

Table 13: Extended Attributes for Message Transitions

| Attribute Name | Description/Usage |
|---|---|
| gatewayId | Gateway Id |
| gatewayName | Gateway name |

Table 14: Extended Attributes for Gateway Transitions

Figure 11 provides the metamodel for the elements in the execution path model.

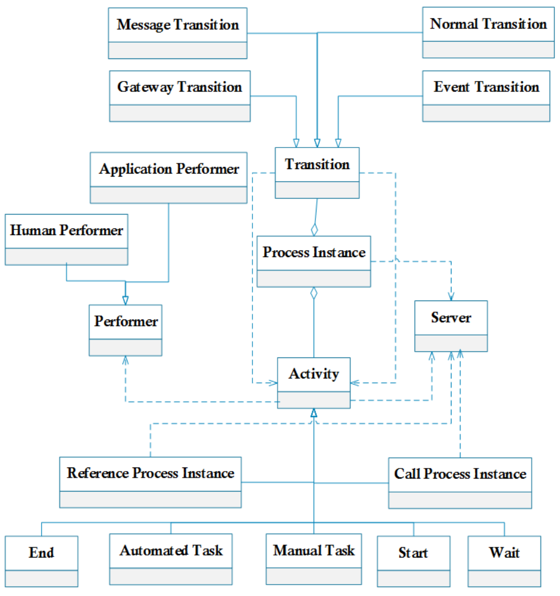


Figure 11: *Process Execution Path Metamodel.*

### 5.3.2 Process Instance Data Metamodel

Process instance data metamodel formalises the PIDS model and also provides metadata information for the process instance data elements.

**Data Item Metadata:** As we discussed in section 5.2.3, the data item can be a basic or complex data type. Depends on the complexity of data type for each data item it can have a different metadata (e.g., Document can have author attribute or size as metadata information). Table 15 lists the common metadata attributes for the data items.

| Attribute Name | Description/Usage |
|---|---|
| dataItemId | Data Item Id |
| dataItemObject | Data Item object |
| version | Data Item version |
| creationDateTime | Specifies the date and time this which data item was created |
| type | Data Item Type |

Table 15: Common Metadata Attributes for the Data Items

**Data Item Array Metadata:** The data item array shares similar metadata attributes with data item but there are some minor differences. The type attribute does not exist in the item array because it can contain different type of data. Item array also has one additional attribute which is size. Size attribute specifies the number of items in this array.

| Attribute Name | Description/Usage |
|---|---|
| dataItemArrayId | Data Item Array Id |
| dataItemArrayObjects | Refers to the data objects in the array object |
| version | Data Item version |
| creationDateTime | Specifies the date and time this which object was created |
| size | Data Item size |

Table 16: Common Metadata Attributes for the Data Item Arrays

**Data Transition:** Links a data input and output to an activity.

| Attribute Name | Description/Usage |
|---|---|
| dataTransitionId | Data Transition Id |
| activityId | Activity Id in the execution path |
| dataInput | Input data |
| dataOutput | Output data |

Table 17: Metadata Attributes for the Data Transition

**Data Input/Output:** Data input/output associates one or many data elements (e.g., Data Item, Data Item Array and Null Data Item) to a data transition.

| Attribute Name | Description/Usage |
|---|---|
| dataInputId | Data input Id |
| dataElementIds | List of the Data Item or Item Array Ids associated to this data input. In case input is empty this list contains only one Null Data Item Id |

Table 18: Metadata Attributes for the Data Input

| Attribute Name | Description/Usage |
|---|---|
| dataOutputId | Data output Id |
| dataElementIds | List of the Data Item or Item Array Ids associated to this data output. In case output is empty this list contains only one Null Data Item Id |

Table 19: Metadata Attributes for the Data Output

Figure 12 provides the metamodel for the elements in the PIDS model.

## 5.4 Customer Journey Process instance

In section 2 we discussed the customer journey's process model and the challenges which implementing this model by two different BPM systems introduced. In this section we will demonstrate how using BPIM can solve these problems. We use the customer journey which was introduced in section 2 and create a process execution path and data model for it. We assume that BPM systems have adopted the BPIM framework and implemented it. They have also added new functionalities to support calling a process instance which is hosted by the other tools. They also share the same process instance repository.

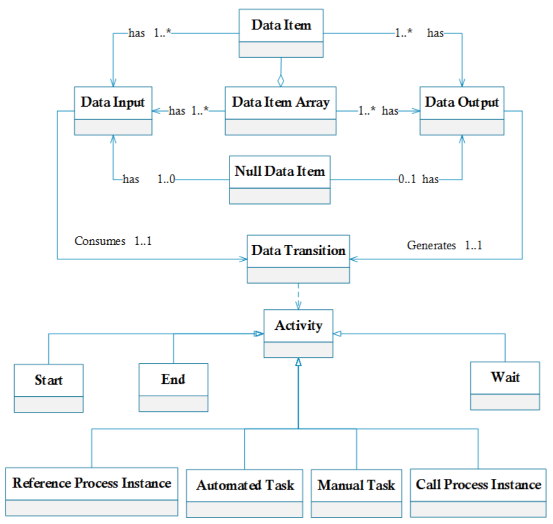


Figure 12: *Process Instance Data Snapshot Metamodel.*

BPIM revolutionises the way we structure process instance information. It provides visual models for process instance activities and data. Also it stores relevant data for each activity and data elements in the metamodel dimension.

### 5.4.1 Execution Path Model

In this section we will focus just on the execution path model and its elements. Figure 13 displays the execution path model for this process instance.

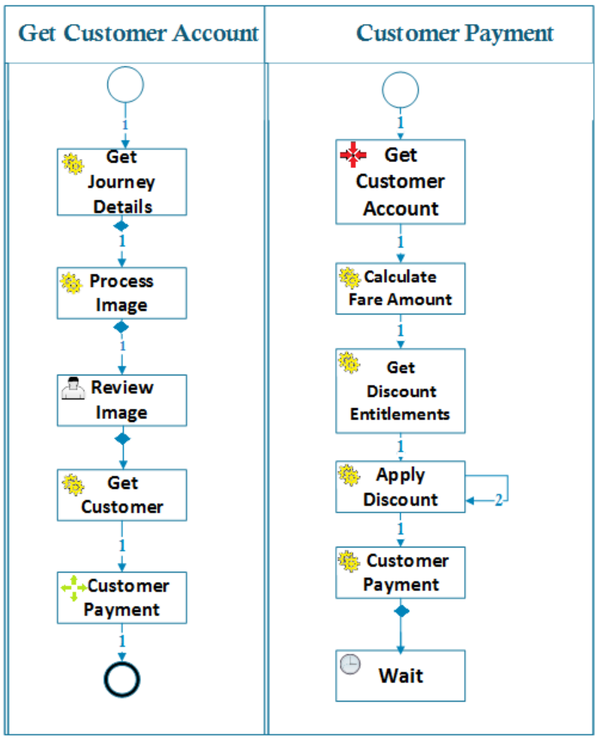


Figure 13: *Customer Journey Process Execution Path*

Figure 13 contains two execution path models, one for `Get Customer Account` process instance in jBPM and the other one for `Customer Payment` process instance in Riftsaw. These are two separate process instances but linked together by call process instance and reference process instance elements.

`Get Customer Account` execution path begins with start activity, a normal transition connects the start node to 'Get Journey Details' automatic task. A gateway indicates that there was a decision element in the process model and transition to 'Process Image' was chosen. Next step is 'Review Image' manual task which is connected to 'Process Image' by another gateway transition. In the next step, executing 'Get Customer' automatic task makes it clear that 'Review Image' was successful. Finally after loading customer account, jBPM execution engine sends a message to Riftsaw to create a new `Customer Payment` process instance and terminates. 'Customer Payment' node is a 'Call Process Instance' activity which links the current process instance to `Customer Payment` process instance.

`Customer Payment` execution model similar to `Get Customer Account` begins with the start element. 'Get Customer' node which is a 'Reference Process Instance' activity points to the `Get Customer Account` process instance. The link between these instances is bidirectional, this means that we can get to the other one if we have just one of them. 'Calculate Fare Amount' and 'Get Discount Entitlements' are the next two activities in the execution path. All the transitions in both execution path models so far are marked with 1, this means the execution engine has traversed that transition just once. But 'Apply Discount' node has two input transitions which they are marked with 1 and 2. We know that there are three discount entitlements for this journey and 'Apply Discount' recalculates the fare amount for each discount entitlement. The transition which marked with 1 indicates that process execution engine has entered this node for the first time and the other transition shows that execution engine has traversed this node two more times in total three times. 'Customer Payment' node has just executed once and after that execution path has stopped at Wait node. There is no end activity for this process instance. This means this process has not finished yet and it is waiting to try to call payment service again.

### 5.4.2 Data Snapshot Model

Similar to the Execution Path Model, we have two Data Snapshot Models `Get Customer Account` and `Customer Payment`. Figure 14 displays these two models side by side.

`Get Customer Account` process instance receives the 'Journey Message' and pass it on to 'Get Journey Details' activity to extracts the journey details from the 'Journey Message'. As we mentioned in section 5.2.2 none of the transitions in the execution path have any effect on the data and they do not appear in the Data Snapshot Model. In this scenario 'Payload Type' gateway transition just checks if the message payload type is image or transponder data and does not modify the process instance data so it does not appear in this model. The number beside the description of each data object in this model is the version number. Having version number helps to distinguish multiple versions of the same object. The result of image processing is stored in the 'Image Processing Result' . Due to low confidence rate the original 'Journey Message' and 'Image Processing Result' is sent to a human for review. Human actor identifies the plate number and send this information back to process instance as 'Car Plate' business entity. In the next step 'Get Customer' activity uses the 'Car Plate' data item and retrieves the customer account. At the end call 'Customer Payment' activity sends the 'Customer Account' and 'Journey Details' to `Customer`

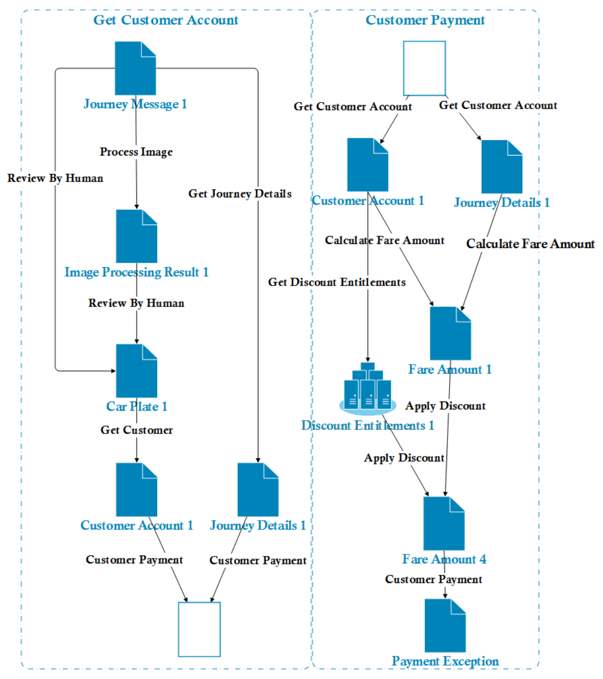


Figure 14: *Customer Journey Data Snapshot Graph*

`Payment` process instance. 'Customer Payment' does not return any response to the caller (i.e., Customer account) as a result of that the last node this model is 'Null Data Item' .

Data Snapshot Model for the `Customer Payment` process instance starts with a 'Null Data Item' because the first activity in the execution path is a 'Reference Process Instance' . As we have already discussed the behaviour of different types of activities in section 5.2.2, a 'Reference Process Instance' does not have any input data and it only generates output (i.e., This type of activity only receives information from outside without any dependency to the data in the current process instance). 'Get Customer Account' transition displays that the current process instance receives 'Customer Account' and 'Journey Details' from `Get Customer Account` process instance. Because both tools are sharing the same process instance repository `Customer Payment` instance needs just retrieve this information from repository. In the next step Calculate Fair Amount' activity uses the 'Customer Account' and 'Journey Details' entities to calculate the cost. This is the cost without discount so the initial version of 'Fair Amount' entity gets created and marked with number one. To calculate the discount, 'Get Discount Entitlements' activity uses the 'Customer Account' entity to fetch the available discounts for this customer. The result is an array of discount entitlements. 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 three versions of 'Fair Amount' . In order to simplify the Data Snapshot Model 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 PIDS Pool, we are just not displaying it. At the end 'Customer Payment' activity uses version three of 'Fair Amount' data item to make the payment but it returns 'Payment Exception' response.

Figure **??** and 16 display the data snapshot pool for tt Get Customer Account and `Customer Payment` process instances.

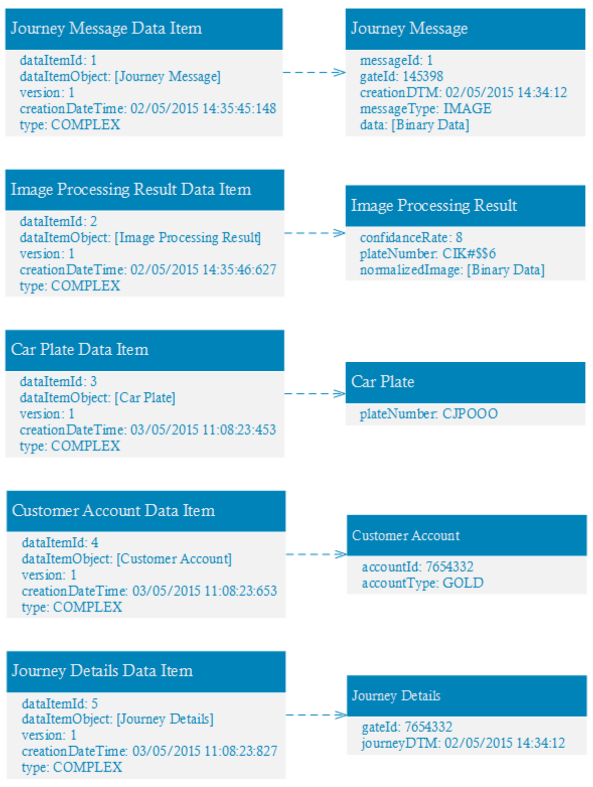

Journey Message Data Item
dataItemId: 1
dataItemObject: [Journey Message]
version: 1
creationDateTime: 02/05/2015 14:35:45:148
type: COMPLEX
Journey Message
messageId: 1
gateId: 145398
creationDTM: 02/05/2015 14:34:12
messageType: IMAGE
data: [Binary Data]
Image Processing Result Data Item
dataItemId: 2
dataItemObject: [Image Processing Result]
version: 1
creationDateTime: 02/05/2015 14:35:46:627
type: COMPLEX
Image Processing Result
confidanceRate: 8
plateNumber: CIK#$$6
normalizedImage: [Binary Data]
Car Plate Data Item
dataItemId: 3
dataItemObject: [Car Plate]
version: 1
creationDateTime: 03/05/2015 11:08:23:453
type: COMPLEX
Car Plate
plateNumber: CJPOOO
Customer Account Data Item
dataItemId: 4
dataItemObject: [Customer Account]
version: 1
creationDateTime: 03/05/2015 11:08:23:653
type: COMPLEX
Customer Account
accountId: 7654332
accountType: GOLD
Journey Details Data Item
dataItemId: 5
dataItemObject: [Journey Details]
version: 1
creationDateTime: 03/05/2015 11:08:23:827
type: COMPLEX
Journey Details
gateId: 7654332
journeyDTM: 02/05/2015 14:34:12

Figure 15: *Get Customer Account Process Instance Data Snapshot Pool*

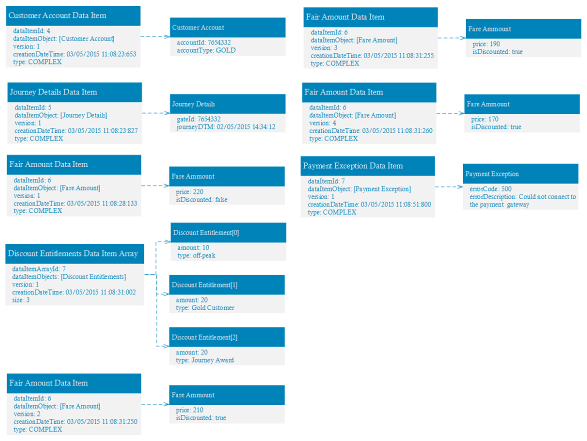


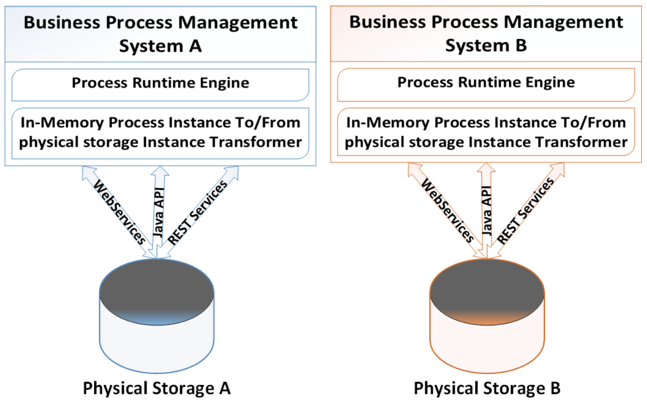
Figure 16: *Customer Payment Process Instance Data Snapshot Pool*

Now we can answer the question finance team asked about this journey. With the help of execution path model, support team can see the process instance activities and where it was stopped. Also Data Snapshot model shows the last response from 'Customer payment' activity. They find out the process instance has recieved an exception response from payment gateway and is waiting to retry to call the payment service.

## 5.5 BPM System Architecture with BPIM

In this section we will look at the effect of having an interoperable process instance model on the existing

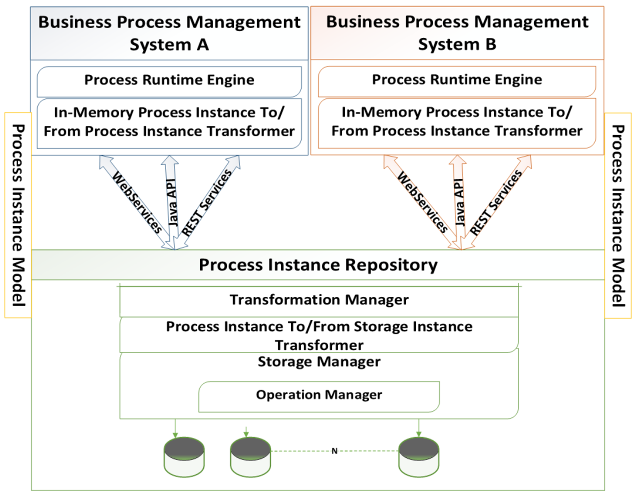
BPM systems architecture. BPM products transform an in-memory process instance object to a storage process instance object and store it. This means, process runtime engine should fully understand the physical storage's model and data structure. Figure 17 shows the BPM system's architecture without using BPIM.



Business Process Management System A
Process Runtime Engine
In-Memory Process Instance To/From physical storage Instance Transformer
WebServices
Java API
REST Services
Physical Storage A
Business Process Management System B
Process Runtime Engine
In-Memory Process Instance To/From physical storage Instance Transformer
WebServices
Java API
REST Services
Physical Storage B

Figure 17: *Current BPM System's Architecture.*

BPIM acts as an abstract layer between process runtime engine and the physical storage.This abstract layer makes the runtime engine completely independent from physical storage. This way runtime engine can focus only on the process instance information and does not need to know about the physical storage's data structure or the commands to insert, delete or update a process instance. Having an abstract layer between runtime engine and physical storage de-couples runtime engine from physical storage. It gives the BPM products ability to replace/modify their physical storage without making any change in the runtime engine. Also we can use the same schema to store different BPM system's process instance information. Figure 18 shows the BPM systems architecture after using proposed process instance model.



Business Process Management System A
Process Runtime Engine
In-Memory Process Instance To/ From Process Instance Transformer
Business Process Management System B
Process Runtime Engine
In-Memory Process Instance To/ From Process Instance Transformer
Process Instance Model
Process Instance Model
WebServices
Java API
REST Services
WebServices
Java API
REST Services
Process Instance Repository
Transformation Manager
Process Instance To/From Storage Instance Transformer
Storage Manager
Operation Manager
N

Figure 18: *BPM System Architecture with BPIM*

BPM system maps the native process instance information to the BPIM instance and sends it to physical storage. Transformation manager maps the changes in the BPIM instance to physical storage's commands(e.g., Insert, delete, update) and executes them.

## 6 Conclusion and Future Work

This paper focuses on creating an interoperable model which provides a holistic view of process instance. This model can be adopted by BPM systems to work as an abstraction layer between execution engine and physical storage. This way all of BPM systems can share their process instances with each other. Also There is no need to create custom tools for each BPM system to extract process instance information because they use the same model to organise the information in process instances.

We are planning to provide the full mapping between the elements in the BPMN interchangeable language and BPEL to/from BPIM execution path elements and realise a transformation algorithm. The next step we are planning to do is to complete the design of the process instance data dimension. A prototype will be developed to show how all these components work together and help 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. & 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. & 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. & Shan, M.-C. (2004), 'Business Process Intelligence', *Computers in Industry* **53**(3), 321–343.

Grigorova, K. & Kamenarov, I. (2012), Object Relational Business Process Repository, *in* 'Proceedings of the 13th International Conference on Computer Systems and Technologies', ACM, pp. 72–78.

Hornung, T., Koschmider, A. & Mendling, J. (2006), Integration of Heterogeneous BPM Schemas: The Case of XPDL and BPEL, *in* 'CAiSE Forum', Vol. 231.

Jeston, J. & Nelis, J. (2014), *Business Process Management*, Routledge.

Ko, R. K., Lee, S. S. & Lee, E. W. (2009), 'Business Process Management (BPM) Standards: A Survey', *Business Process Management Journal* **15**(5), 744–791.

La Rosa, M., Reijers, H. A., Van Der Aalst, W. M., Dijkman, R. M., Mendling, J., Dumas, M. & GarcíA-BañUelos, L. (2011), 'Apromore: An advanced process model repository', *Expert Systems with Applications* **38**(6), 7029–7040.

Ma, Z., Wetzstein, B., Anicic, D., Heymans, S. & 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)'.

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

Qin, J. & Fahringer, T. (2012), Semantic-based scientific workflow composition, *in* ‘Scientific Workflows’, Springer, pp. 115–134.

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

Van Der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., Bose, J. C., van den Brand, P., Brandtjen, R., Buijs, J. et al. (2012), Process Mining Manifesto, *in* ‘Business Process Management Workshops’, Springer, pp. 169–194.

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

van der Aalst, W., ter Hofstede, A. & Weske, M. (2003), Business Process Management: A Survey, *in* ‘Business Process Management’, Vol. 2678 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–12.

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

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