How to integrate the BPMConverter

BP2014w1-Team

July 9, 2015

This How-To describes possibilities to integrate the BPMConverter into a project. We will motivate why conversion is useful and afterwards introduce the Converter module in general. We will describe the structure and then the integration of the BPMConverter will be explained with an example.

1 Introduction

Process models can be expressed using different representations. Each representation serves a special purpose. Activity Centric Process Models such as BPMN models can be used to express the order of tasks, Object Life Cycles provide us with the possibility of conformance checking and configuring state transitions. Hence, it might be helpful to have models using different notations and approaches of one process. The challenge is to assert that the models are compliant. This goal can be reached by generating one representation from another. For example we can extract an Object Life Cycle from a BPMN model.

1.1 How to receive the Code

The code is hosted on Github[1]. You have the possibility to download the sources in a ZIP archive or to clone the repository using git[2]. We recommend to use git. Figure 1 shows how you can clone the repository. Afterwards the sources will be saved on your local machine.

```
git clone https://github.com/BP2014W1/BPMConverter
```

Figure 1: Cloning the repository from Github using Git

[1] `[http://github.com]`
[2] `http://git-scm.com`

1.2 How to build the converter

The converter module can be compiled and build using Maven[3]. Maven will compile the sources and run all the tests. Run the command 2 from the command line inside your projects root folder.

```
mvn clean install
```

Figure 2: Maven command to compile the sources and build a JAR archive

2 Project Structure

The BPMConverter is build with two purposes. On the one hand it provides interfaces and classes to describe Process Models using different representations. There are elements to create an so called Activity Centric Process model as well as elements to create an Object Life Cycle. These classes can be found in the *activity_centric* package.

On the other hand their are classes implementing different algorithms to generate additional process models. These classes can be found inside the *converter* package.

2.1 How to create a process model

The *conversion* package contains classes to build process models. In general a model consists of multiple nodes connected by edges. All classes representing a node must implement the *INode* interface. Edges must implement the *IEdge* interface. Each node can have multiple incoming and outgoing edges, but their might stricter restrictions for the concrete implementations.

Currently their are two types of models implemented:

- **Activity centric process models**. These models can be compared to BPMN models.
- **Object Life Cycles**. Instances of this model represent a state transition chart for one data class.

2.1.1 Activity Centric Process Models

Activity centric process model express processes by adding constraints to activities. Those constraints can be either data dependencies or control flow dependencies. Data dependencies express data objects which will be read and written by the activity; whereas control flow allows us to create an order between activities of one model.

The Table 1 shows the elements of Activity Centric Process Models the interface they implement and the restrictions they add to the default behavior.

[3] https://maven.apache.org/

Additionally, the scenario package contains a class (Table 2) to aggregate multiple activity centric process models into one scenario.

2.1.2 Object Life Cycle

Object Life Cycles are state transitions systems for one data class. They express all possible states and transitions between them. They can used in various ways for example for conformance checking.

The structure of an Object life cycle is simple. Every model has exactly one start node and should have at least one final node. There is no limit for other nodes. All these models will be connected using state transitions. In BPM a state transition normally represents an action which alters the data object. Table 3 describes the possible elements used inside a object life cycle.

The Table 1 shows the elements of Activity Centric Process Models the interface they implement and the restrictions they add to the default behavior.

In addition there is another package *synchronize*. This package contains the synchronized Object Life Cycle (Table 4).

2.2 How to convert a Process Model

Once you have created process models with the classes described above (or with any subclasses) you can use the elements of the *converter* package to convert these models. The *subpackages* indicate the source of the generation. *OLC* subpackage transforms object life cycles, *activity_centric* Activity Centric Process Models and *pcm* Production Case Management scenarios (A collection of Activity Centric Process Models). Figure 3 shows which transformations can be made. Table 2.2 shows which class provides methods for the transformations.

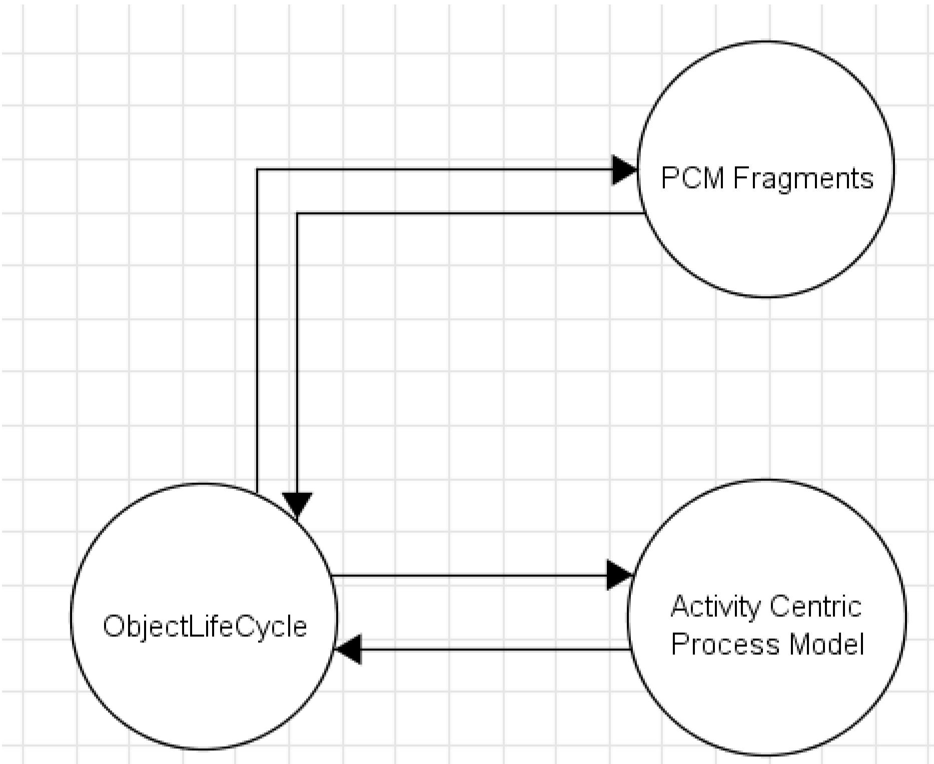

Figure 3: Possible transformations

class name	**interface**	**restrictions**
Activity	INode	• Only Control Flow and Data Flow edges are supported • There must be only one incoming control flow edge • There must be only one outgoing control flow edge
ActivityCentricProcessModel	IModel	• Node must be one of the supported types.
ControlFlow	IEdge	• Source must be either a Gateway, Event or Activity
DataFlow	IEdge	• The source must be either a Activity or DataObject • If the source is an Activity the target must be a DataObject and vise versa
DataObject	INode	• All incoming and outgoing edges must be of type DataFlow
Event	INode	• An event can have only one edge • Only an Start Event can have an outgoing edge • Only an End Event can have an incoming edge
Gateway	INode	• Every Edge must be of type ControlFlow. • An additional type attribute defines if it is an XOR or an AND

Table 1: Elements used by the activity centric process Model

class name	interface	restrictions
Scenario	IModel	• Initialized from multiple ActivityCentricProcessModels • Altering is not possible

Table 2: The Scenario

class name	interface	restrictions
Activity	INode	• Only Control Flow and Data Flow edges are supported • There must be only one incoming control flow edge • There must be only one outgoing control flow edge
ObjectLifeCycle	IModel	• Every node must be of type DataObjectState • There should be exactly one initial state • There should be at least one final state
DataObjectState	INode	• All incoming and outgoing edges must be of type StateTransition
StateTransition	IEdge	• Source and target must be of type DataObjectState

Table 3: Elements used by the object life cycle model

class name	**interface**	**restrictions**
SynchronizedObjectLifeCycle	IModel	• Initialized from multiple ObjectLifeCycles • Altering is not possible • Additional Synchronization edges can be created

Table 4: The SynchronizedObjectLifeCycle

input	**output**	**class name**
ActivityCentricProcessModel	SynchronizedObjectLifeCycle	ActivityCentricToSynchronizedOLC
2x Collection<ObjectLifeCycle>	Collection<ActivityCentricProcessModel>	FragmentsFromOLCVersions
SynchronizedObjectLifeCycle	ActivityCentricProcessModel	SynchrnoizedOLCToActivityCentric
Collection<ActivityCentricProcessModel>	Collection<ObjectLifeCycle>	ScenarioToSynchronizedOLC

Table 5: Class that provide transformation algorithms

3 Integration Strategies

There different approaches on how to use the BPMConverter.

3.1 How to use the provided classes?

A naive way to integrate the converter in your project would be to create the models out of the classes provided by the converter. On the other hand you can write your own converter converting your models to BPMConverter compatible models and back in your representation. The biggest benefit of this approach is, that you don't have to alter your models.

3.2 How to use inheritance?

If you have full power of the implementation of your models, you might subclass the classes provided by the BPMConverter. This allows you to add any behavior to the model elements used by the converter. The benefit is that you can use your models directly as an input for the BPMConverter. But you have to be aware that the result will not be compliant to your models, because the converter uses the superclasses.

Nevertheless, transforming those superclass models into your model representations is easy since you can simply map the classes.

3.3 How to integrate using adapters?

In our Production Case Management Framework we integrated the Converter module inside the ProcessEditor[4] using the adapter pattern. Every adapter class extends an element of the BPMConverter and is initialized with at least one model element used by the ProcessEditor. The adapter wraps these objects and delegates part of calls to the original object. If necessary the return value will be wrapped as well. In order to improve the performance and to allow comparison with the default equals method wrapped objects are cached.

[4] `https://github.com/BP2014W1/processeditor`