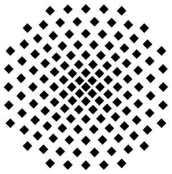

Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

Propagation of States from BPEL Process Instances to Chevron Models

David Schumm, Dimka Karastoyanova,
Frank Leymann, and Sumadi Lie

Technical Report 2011/06
July 11, 2011

Institut für Architektur von Anwendungssystemen

Universitätsstr. 38
70569 Stuttgart
Germany

CR: C.2.4, D.2.2, H.4.1, H.5.2, H.5.3

Abstract

This report describes key aspects of a code library that we developed for the purpose of state propagation for business process monitoring on different levels of abstraction. The library supports the propagation of execution states of process instances based on the Business Process Execution Language (BPEL) to process models specified in the "Chevron" language. The Chevron language is an abstract, non-executable process language that we especially designed for abstract process instance monitoring purposes. The look and feel of this graphical language is similar to value chains. The basic concept of the Chevron language is based on Chevron-shaped charts which can be modeled in Microsoft PowerPoint to describe a process on a high level of abstraction.

We aim at enabling the use of high-level process in order to monitor the instance status of a much more detailed, lower-level model. We describe the overall procedure of performing state projections along a concrete scenario. We describe a format for state propagation rules which define how the status of activities of a BPEL process instance should be projected to the elements of a Chevron model. We present a format to serialize process models in the Chevron language. We present a graphical template based on Scalable Vector Graphics (SVG) which we employ to render a stateful Chevron model graphically. The Chevron language is just one language to be used for abstract representation of process instances. However, the approach for state propagation is generic and can be applied for other languages, too.

Table of Contents

1 Background

In previous work we have developed a framework for web-based monitoring of running or past instances of BPEL process models [2]. This framework, called Business Process Illustrator (BPI), is capable of generating a graphical representation of a process instance that can be displayed in a web browser that is capable of displaying Scalable Vector Graphics (SVG) like Mozilla Firefox. In addition to this monitoring functionality, BPI also supports particular process viewing concepts. Process views, as described for instance in [5, 6, 7], are a means to abstract a process model and thus reduce its complexity in terms of contained activities and control structures. Furthermore, graphical aspects like activity coloring can be used to ease understanding. BPI supports the abstraction of process instances: The omission and aggregation of activities is enabled based on user input, activity types and runtime information. In addition to that, BPI uses graphical highlighting of process structures and graphical analysis of performance information related to the duration of activity executions. With respect to the graphical analysis, we realized heatmap functionality that paints the intensity of activity background colors according to their execution duration.

Although the BPI monitoring framework is capable of process instance monitoring plus presenting abstracted running process instances it has some limitations. In particular, it is bound to monitoring a process instance in the language in which the process model is being executed. However, often process models have been specified initially in different languages, such as Event-driven Process Chains (EPC). Furthermore, they are specified on a higher level of abstraction before they are refined to an executable process model. As a consequence, process instance monitoring needs to deal with process models on different levels of abstraction, specified using different process languages than the ones used for the process modeling. To overcome this limitation, the library we describe in this report implements the novel state propagation approach proposed in [1] to enable process instance monitoring across the boundaries of process models and languages.

The report's structure is the following: In Section 2 a generic architecture for process instance monitoring is discussed. The proposed library is positioned in this architecture. Next, in Section 3 state propagation patterns are summarized. They represent the core concept of the approach. The Chevron language we designed for abstract process monitoring is introduced in Section 4. Section 5 gives an overview on the approach by describing the overall procedure of performing a state projection. Section 6 describes the implementation artifacts of the library consisting of XML schema definitions and illustrating code examples for specifying (i) state propagation rules, (ii) Chevron process models, and (iii) SVG templates for rendering the Chevron process elements. In Section 7 a concrete scenario of an ordering process to be monitored by a customer is presented along with the implementation artifacts for that scenario. Section 8 gives a summary and concludes the report.

2 Generic Architecture for Process Instance Monitoring

In [1] we presented a generic architecture for process instance monitoring on different levels of abstraction. We also provided a technical report on view-based process instance monitoring which breaks down the architecture to the key implementation aspects [2]. In the following, we briefly summarize these works in order to provide the necessary background information for the further discussion. The generic architecture for process instance monitoring is shown in Figure 1. The right hand side of the architecture shows the execution environment of a process. The process engine (we use Apache ODE[1] in our prototypes) performs the execution of the activities specified in the process models which are deployed on it. During execution, events representing status changes are generated. The monitoring framework is located in the center part of the architecture figure. The framework can retrieve execution events using an adapter which connects to the process engine. This adapter also retrieves the process models deployed on the process engine. The framework can calculate the current

[1] The Apache Software Foundation: Apache ODE, http://ode.apache.org/, 2011.

status of a process instance based on the event information retrieved via the adapter. If process views and state propagation are not considered at this stage, the retrieved information is transformed to an internal representation which is forwarded to the monitoring frontend. The monitoring frontend component is generating a graphical representation of the process instance based on SVG which is then sent to the client's web browser.

With respect to the focus of this report, the view generator component is of major interest. It is shown in the bottom middle of the architecture diagram depicted in Figure 1. As we described in detail in [2], this component performs process view transformations (aggregation, omission, highlighting). The library we present in this report, however, significantly extends the functionality of the view generator component. After high-level models and state propagation rules have been deployed on the monitoring framework, the view generator is enabled to present the status of the instance of a BPEL process model in a different model and a different language called "Chevron". The Chevron language is an abstract, non-executable process language that we especially designed for abstract process instance monitoring purposes. The graphical look and feel of this language is similar to value chains, though it does not adopt all the semantics of value chains. The basic concept of the Chevron language is based on Chevron-shaped charts which can be modeled in Microsoft PowerPoint[2] to describe a process on a high level of abstraction. The view designer component shown in the lower left in Figure 1 can be used by an administrator for definition of high-level models and corresponding state propagation rules. The view designer component is out of scope of this report.

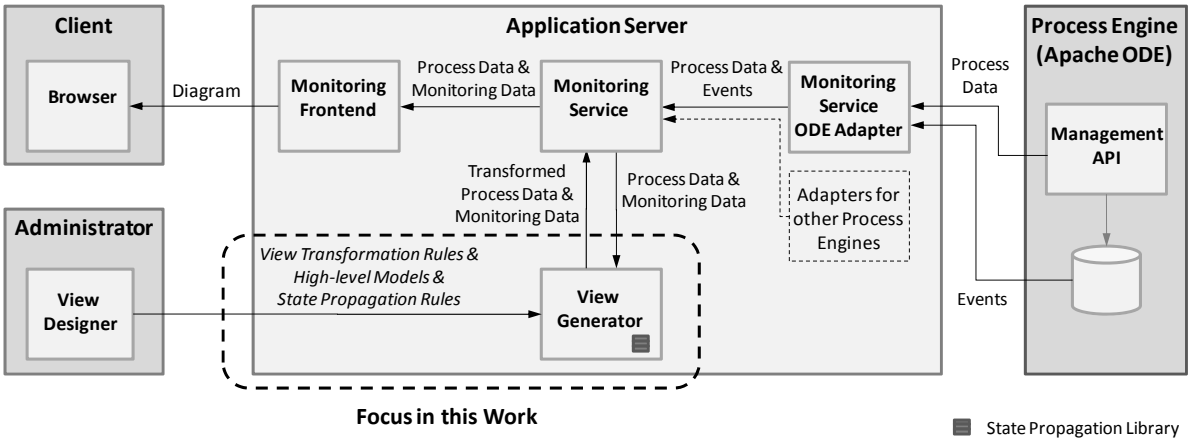

Figure 1 Generic architecture for business process monitoring on different levels of abstraction [1].

With these enhancements the view generator is capable of supporting two different scenarios of process instance monitoring on different levels of abstraction. Figure 2 illustrates these scenarios:

a) Abstraction based on process view transformations: Activities can be omitted, process structures can be aggregated, and the graphical display can be adjusted to ease process instance monitoring.

b) "State projection" performs a concrete mapping of states of activities in a BPEL process instance to states of elements in a Chevron model. In other words, a state projection is the evaluation of the set of state propagation rules that is defined to specify the state mapping.

[2] Microsoft: PowerPoint 2010, http://office.microsoft.com/en-us/powerpoint/, 2011.

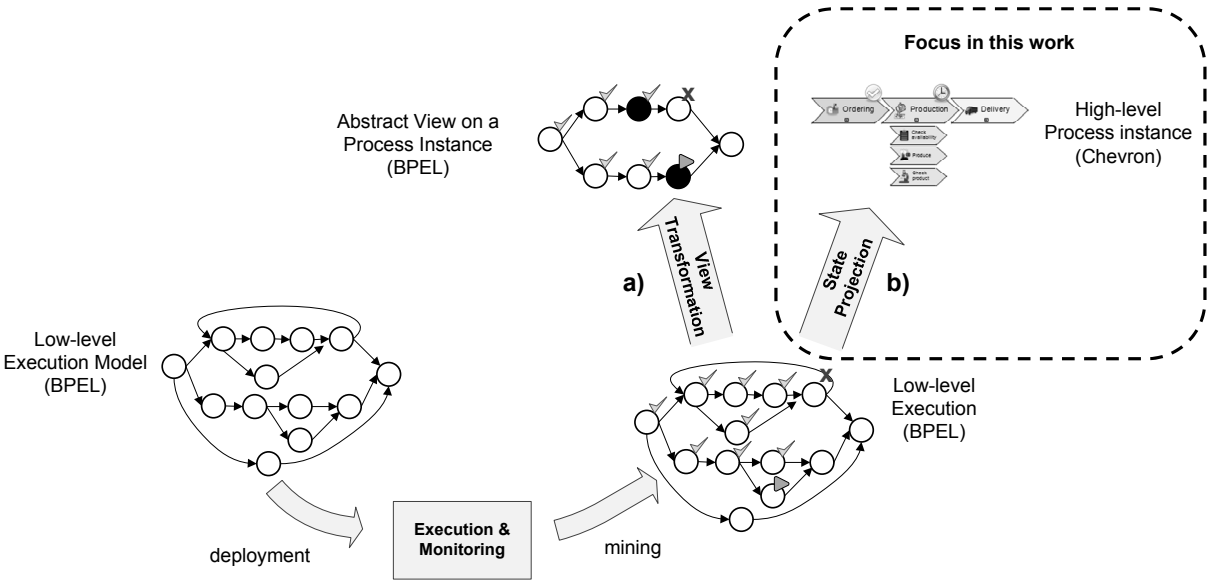

Figure 2 Process instance monitoring scenarios: a) view transformation, b) state projection

3 State Propagation Patterns

In previous work [1] we presented state propagation patterns which are the elementary forms of mapping execution states from activities of one process model instance to another model. In the following we briefly summarize these patterns as they represent the core concept of the overall approach. A detailed description of the patterns including the rationale and examples is provided in [1]. Figure 3 shows a graphical representation of the patterns. Their semantics can be summarized as follows:

1. Direct state propagation: This pattern describes the most common form of state propagation. It is applicable when the source model and the target model are specified in the same language and consequently, have the same state space, i.e., the same set of state types.
2. State alteration: This pattern is applied when there is a change of state types from lower level to higher level due to different semantics of activities, or due to different state spaces in the process languages.
3. State combination: This pattern describes the combination of multiple states to one state which is then propagated to an activity in the higher level model.
4. State deduction: This pattern describes the case when external data sources are used to deduce a state of an element on higher level.
5. Transition state: This pattern describes the propagation of a status to a control link on higher level.
6. State aggregation: This pattern describes performing projections of multiple instance states to one model on higher level.

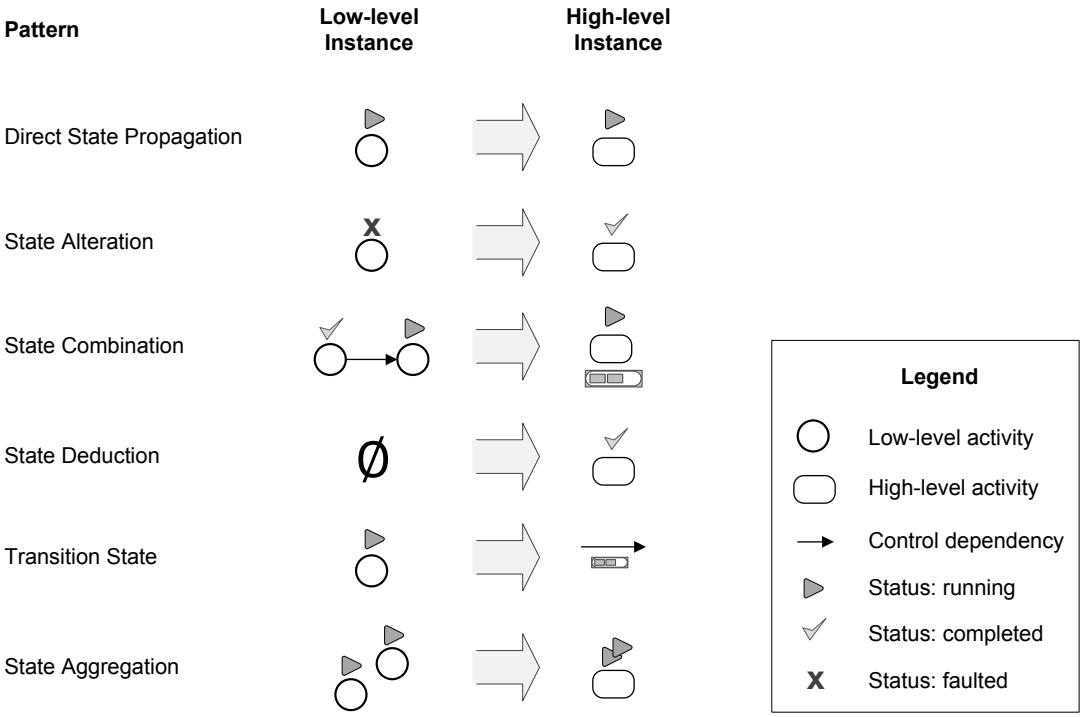

Figure 3 State propagation patterns [1].

4 The Chevron Language

As said before, the Chevron language is an abstract, non-executable process language that we designed especially for abstract process instance monitoring purposes. The language does not contain any complex control structures. It consists of simple and complex activities which are ordered sequentially and displayed from left to right. A simple example of a process model in the Chevron language is shown in Figure 4. This example contains three first-level activities, *Ordering*, *Production*, and *Delivery*. *Ordering* is a complex activity in collapsed mode. *Production* is a complex activity in expanded model, revealing the sub-activities it is made up of. *Delivery* is a simple activity, though on first-level. For simplicity of the language, complex activities are only available on first-level.

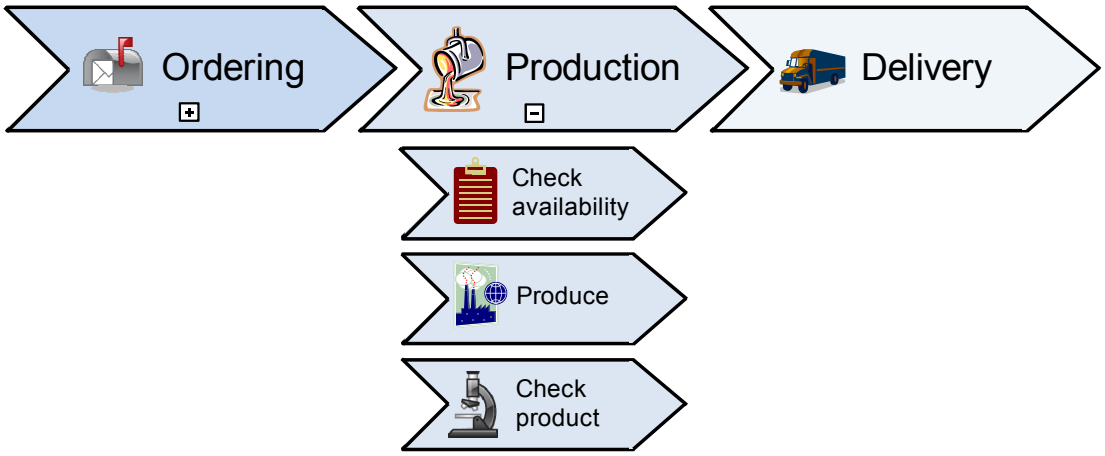

Figure 4 Example of a process model in the Chevron language.

The activities have certain graphical aspects that are depicted in Figure 5. An activity is displayed using a particular background color and font for its caption. An icon visually represents the semantics of the activity. A status decorator is used to graphically indicate the execution status of the activity. A complex activity is equipped with an icon for collapsing.

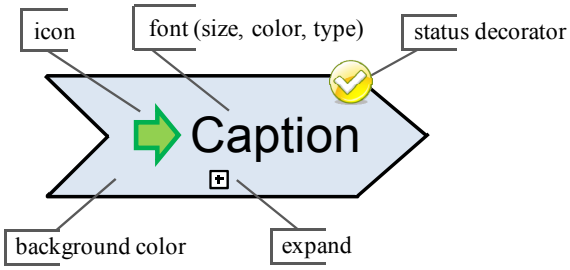

Figure 5 Graphical concept of a Chevron activity.

Compared to the process language BPEL, the Chevron language has a much more simple state space, i.e., the set of state types is significantly smaller. This refers to the status of the overall process status, as well as to the status of an activity. Table 1 compares the state spaces of BPEL and Chevron as we defined it. In this report we assume a BPEL implementation based on the WfMC reference model [8].

	BPEL	**Chevron**
Process Instance	Active: process is execution Completed: process completed successfully Terminated: process completed abnormally	Running: process is executing Completed: process completed
Activity Instance	No Status: activity not started yet Ready: activity is ready for execution Active: activity is executing Completed: activity completed successfully Suspended: activity is temporarily suspended Failed: activity faulted Terminated: activity is terminated	Outstanding: activity not started Running: activity is executing Completed: activity completed

Table 1 Comparison of state spaces of BPEL and Chevron

A simplified conceptual model of the language is shown in Figure 6. A Chevron process carries an identifier and an instance status that can be set for it. The meta-data contains a name and a description which is targeted to the end-user monitoring the process. A Chevron process may contain a large number of activities. However, for ease of use a Chevron process should not be larger than seven to nine activities when considering human perception background in the language design [9].

An activity in a Chevron process carries an identifier which is used for targeting the state propagation rules. The effective status is calculated during evaluation of the state propagation rules. An activity is equipped with meta-data like name, an icon and a description that can be shown to the end-user on mouse-over. An activity may be complex, containing sub-activities.

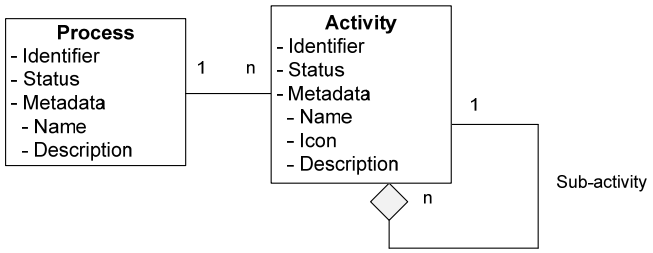

Figure 6 Conceptual model of the Chevron language.

5 State Projection Walkthrough

This section describes the overall procedure of performing a state projection, i.e., how to obtain a graphical rendering of stateful high-level model, whose states are based on rules that describe the propagation of activity states from a lower-level model. We describe the procedure based on a process model defined using the Business Process Model and Notation Standard (BPMN) [10]. Then, we illustrate this procedure with the implementation artifacts created for a concrete scenario.

Figure 7 shows the procedure we implemented in our library to perform a state projection. It is entirely written in Java, using just the packages *java.util*, *java.io*, and *java.lang.StringBuilder* in general and *javax.xml* and *org.w3c.dom* for processing XML in particular. For code development we employed the Eclipse Java EE IDE. The SVG template has been developed using the XML tools available in Eclipse and Mozilla Firefox for testing. The library supports two different ways of invocation. One way is to pass all required artifacts as parameters in the invocation. For performance reasons, another way is to provide file-system paths for the inputs. The process shown in Figure 7 shows the interaction with the file system and corresponding in-memory artifacts.

There are five steps:

1. Read stateful BPEL: In this step a BPEL process model is transformed into an internal representation. This representation needs to be augmented with the execution states of a particular instance of this process model. The library we developed expects a BPEL process model that is already augmented with the execution states. As a result from this step, a set of activities and corresponding states is obtained.
2. Read state projection definition: In the next step, the state projection definition is read into memory. Firstly, the specified mapping sets are extracted. Then, the state propagation rules are extracted and linked to the referenced mapping sets.
3. Calculate propagation states: The propagation rules specify the sources and targets of states and they reference mapping sets that describe how the states are being mapped. Based on this input, a list of activities and states of the high-level model can be calculated by iterating over all rules and evaluating them against the BPEL activities and states.

4. Perform state propagation: To perform the state propagation, the list of propagation states calculated in the prior step is needed. This list is processed to augment an input Chevron model with states. After the augmentation, the status of complex activities contained in the Chevron model can be calculated. As output of this step a stateful Chevron model is produced.
5. Generate SVG representation: A sub-component in our library generates an SVG representation of a Chevron model. This output, a stateful SVG, concludes the procedure.

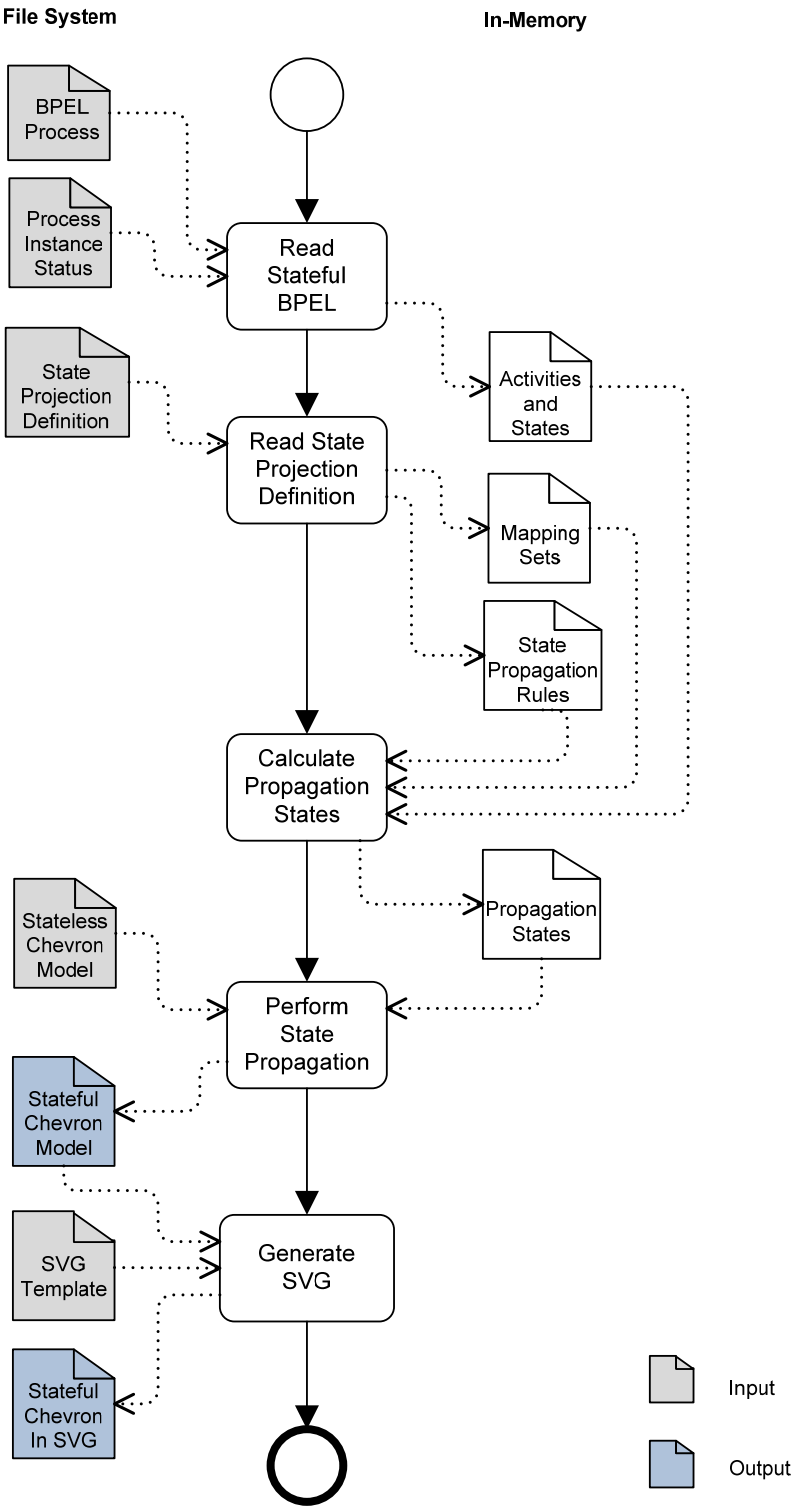

Figure 7 Procedure to obtain a stateful Chevron model and corresponding SVG rendering.

6 Implementation Artifacts

This section describes the implementation artifacts of the library consisting of XML schema definitions and illustrating code examples.

6.1 XML Schema for Chevron Models

The XML schema shown in Listing 1 implements the conceptual Chevron language basics discussed in Section 4. The schema supports:

- Complex and simple activities
- Activity states: *Outstanding*, *Running*, *Completed*, *No status* (represented by an empty String)
- Process attributes: *ProcessID*, *ProcessInstanceID*, *ProcessModelVersion*, and *ProcessModel-Documentation* (for additional metadata)
- Activity attributes: *ActivityName*, *ActivityID*, *ActivityCaption*, *ActivityDocumentation* (for additional metadata), *ActivityIcon*, and *ActivityColor* (referencing an SVG definition)
- Using this schema both stateful and stateless Chevron models can be serialized

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
      <!-- definition of simpleActivity status -->
      <xs:element name="activityStatus">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Outstanding"/>
                        <xs:enumeration value="Running"/>
                        <xs:enumeration value="Completed"/>
                        <xs:enumeration value=""/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <!-- definition of complexActivity status -->
      <xs:element name="complexActivityStatus">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Outstanding"/>
                        <xs:enumeration value="Running"/>
                        <xs:enumeration value="Completed"/>
                        <xs:enumeration value=""/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <!-- definition of attributes of chevron process model -->
      <xs:attribute name="processModelName" type="xs:string"/>
      <xs:attribute name="pid" type="xs:string"/>
      <xs:attribute name="piid" type="xs:string"/>
      <xs:attribute name="processModelVersion" type="xs:long"/>
      <xs:attribute name="processModelDocumentation" type="xs:string"/>
      <!-- definition of attributes of activities -->
      <xs:attribute name="activityName" type="xs:string"/>
      <xs:attribute name="activityID" type="xs:string"/>
      <xs:attribute name="activityCaption" type="xs:string"/>
      <xs:attribute name="activityDocumentation" type="xs:string"/>
      <xs:attribute name="activityIconForCaption" type="xs:anyURI"/>
      <xs:attribute name="activityIconForComplexActivity" type="xs:anyURI"/>
      <xs:attribute name="activityColor" type="xs:string"/>
      <!-- definition of chevron process model -->
      <xs:element name="chevronProcessModel" type="cPM"/>
      <xs:complexType name="cPM">
```

```
            <xs:choice maxOccurs="unbounded">
                  <xs:element ref="Activity" maxOccurs="unbounded"/>
                  <xs:element ref="complexActivity" minOccurs="0"
                              maxOccurs="unbounded"/>
            </xs:choice>
            <xs:attribute ref="processModelName" use="required"/>
            <xs:attribute ref="pid" use="required"/>
            <xs:attribute ref="piid" use="required"/>
            <xs:attribute ref="processModelVersion" use="required"/>
            <xs:attribute ref="processModelDocumentation" use="required"/>
      </xs:complexType>
      <!-- definition of activityDepth / complex activity -->
      <xs:element name="complexActivity">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="complexActivityStatus"/>
                        <xs:element ref="Activity" maxOccurs="unbounded"/>
                  </xs:sequence>
                  <xs:attribute ref="activityName"/>
                  <xs:attribute ref="activityID" use="required"/>
                  <xs:attribute ref="activityCaption" use="required"/>
                  <xs:attribute ref="activityDocumentation" use="required"/>
                  <xs:attribute ref="activityIconForCaption" use="required"/>
                  <xs:attribute ref="activityIconForComplexActivity"
                                use="required"/>
                  <xs:attribute ref="activityColor" use="required"/>
            </xs:complexType>
      </xs:element>
      <!-- definition of simple activity -->
      <xs:element name="Activity">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="activityStatus"/>
                  </xs:sequence>
                  <xs:attribute ref="activityName"/>
                  <xs:attribute ref="activityID" use="required"/>
                  <xs:attribute ref="activityCaption" use="required"/>
                  <xs:attribute ref="activityDocumentation" use="required"/>
                  <xs:attribute ref="activityIconForCaption" use="required"/>
                  <xs:attribute ref="activityColor" use="required"/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Listing 1 XML schema for Chevron models [4].

6.2 XML Example of a Stateless Chevron Model

In Listing 2 a simple example of a stateless Chevron model is shown. It contains two first-level activities. The first one is simple ("Ordering") and the second one is complex ("Production"), containing two sub-activities ("Produce", "Check product"). We assume a Chevron activity can be uniquely identified by its attribute *activityID*.

```
<?xml version="1.0" encoding="UTF-8"?>
<chevronProcessModel processModelVersion="1" piid="5" pid="185"
                     processModelDocumentation="documentation"
                     processModelName="example"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:noNamespaceSchemaLocation="chevron.xsd">
   <Activity activityColor="orange" activityIconForCaption="order.png"
             activityDocumentation="order acknowledgement"
             activityCaption="Ordering" activityID="1">
      <activityStatus></activityStatus>
```

```
   </Activity>
   <complexActivity activityColor="orange" activityIconForCaption="product.png"
                    activityDocumentation="production" activityCaption="Production"
                    activityID="2" activityIconForComplexActivity="plus.png">
      <complexActivityStatus></complexActivityStatus>
         <Activity activityColor="light-orange"
                   activityIconForCaption="produce.png"
                   activityDocumentation="producing" activityCaption="Produce"
                   activityID="2.1">
            <activityStatus></activityStatus>
         </Activity>
         <Activity activityColor="light-orange"
                   activityIconForCaption="product.png"
                   activityDocumentation="checking product"
                   activityCaption="Check product" activityID="2.2">
            <activityStatus></activityStatus>
         </Activity>
   </complexActivity>
</chevronProcessModel>
```

Listing 2 XML excerpt of a Chevron model.

6.3 XML Schema for State Projection Definitions

The XML schema that defines the structure of a state projection definition is shown in Listing 3. The schema defines two main sections, *statePropagationSets* and *statePropagationRules*. The first defines different forms of mapping of sets of states (based on the BPEL state space) to one target state (based on the Chevron state space). A *statePropagationSet* contains particular condition-action rules to define the state mapping. An *else* construct specifies the default mapping which is performed when none of the conditions is met.

The second section, *statePropagationRules,* defines concrete rules for propagating states. A *statePropagationRule* consists of triple of sub-elements. The *fromActivities* sub-element references one or more activities in a BPEL model. It also contains a *toActivity* element which references an activity in the Chevron model. The *stateSet* sub-element references a *statePropagationSet.* The segregation of the mappings of states and concrete rules provides the ability to reuse mapping definitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
      <!-- attributes definiton -->
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="identifier" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
      <xs:attribute name="pattern" type="xs:string"/>
      <!-- low level activity status definition-->
      <xs:simpleType name="low-levelActivityStatus">
            <xs:list itemType="xs:string"/>
      </xs:simpleType>
      <!-- containState element definition -->
      <xs:element name="containState" type="stateList"/>
      <!-- low level activity states as a list -->
      <xs:simpleType name="stateList">
            <xs:restriction base="low-levelActivityStatus"/>
      </xs:simpleType>
      <xs:element name="targetState">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Outstanding"/>
```

```
                        <xs:enumeration value="Running"/>
                        <xs:enumeration value="Completed"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <!-- allStateEqual element definition -->
      <xs:element name="allStatesEqual" type="xs:string"/>
      <!--  element definition -->
      <xs:element name="else">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:enumeration value="Outstanding"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <!-- definiton of rule element -->
      <xs:element name="condition">
            <xs:complexType>
                  <xs:sequence>
                        <xs:choice>
                              <xs:element ref="containState"/>
                              <xs:element ref="allStatesEqual"/>
                        </xs:choice>
                        <xs:element ref="targetState"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- definiton of the statePropagationSet -->
      <xs:element name="statePropagationSet">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="condition" maxOccurs="unbounded"/>
                        <xs:element ref="else" minOccurs="1" maxOccurs="1"/>
                  </xs:sequence>
                  <xs:attribute ref="name" use="required"/>
            </xs:complexType>
      </xs:element>
      <!-- definiton of the statePropagationSets -->
      <xs:element name="statePropagationSets">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="statePropagationSet"
                                    maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- definiton of the activities -->
      <xs:element name="activity">
            <xs:complexType>
                  <xs:attribute ref="identifier" use="required"/>
                  <xs:attribute ref="value" use="required"/>
            </xs:complexType>
      </xs:element>
      <!-- definiton of the fromActivities -->
      <xs:element name="fromActivities">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="activity" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- definiton of the toActivity -->
      <xs:element name="toActivity">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="activity"/>
                  </xs:sequence>
            </xs:complexType>
```

```
        </xs:element>
        <!-- definiton of the stateSet -->
        <xs:element name="stateSet">
              <xs:complexType>
                    <xs:attribute ref="pattern" use="required"/>
              </xs:complexType>
        </xs:element>
        <!-- definiton of the statePropagationRule -->
        <xs:element name="statePropagationRule">
              <xs:complexType>
                    <xs:sequence>
                          <xs:element ref="fromActivities"/>
                          <xs:element ref="toActivity"/>
                          <xs:element ref="stateSet"/>
                    </xs:sequence>
                    <xs:attribute ref="name" use="required"/>
              </xs:complexType>
        </xs:element>
        <!-- definiton of the statePropagationRules -->
        <xs:element name="statePropagationRules">
              <xs:complexType>
                    <xs:sequence>
                          <xs:element ref="statePropagationRule"
                                      maxOccurs="unbounded"/>
                    </xs:sequence>
              </xs:complexType>
        </xs:element>
        <!-- definition of state Projection -->
        <xs:element name="stateProjection" type="sP"/>
        <xs:complexType name="sP">
              <xs:sequence>
                    <xs:element ref="statePropagationSets"/>
                    <xs:element ref="statePropagationRules"/>
              </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Listing 3 XML schema for state projection definitions [4].

6.4 XML Excerpt of a State Projection Definition

In Listing 4, an excerpt of an exemplary state projection definition is shown. It contains one *statePropagationSet* element ("combination") and one *statePropagationRule* ("production"). The exemplary state propagation set shown in Listing 4 defines two condition-action rules and an else-statement if none of the conditions evaluates to true:

- Condition 1: If one of the states *Active*, *Failed*, *Suspended*, or *Terminated* is included in the states set, then the Chevron activity state is set to *Running*
- Condition 2: If all states of the BPEL activities equal *Completed*, then the Chevron activity state is set to *Completed*
- Else: If none of these conditions evaluates to true, then the Chevron activity state is set to *Outstanding*

The *statePropagationRule* in this example contains three sub-elements in the *fromActivities* part. This means that the status of all these activities must be considered in the evaluation of the conditions. When the propagation of states is performed, the mapping that is referenced in the *statePropagationRule* is evaluated and the target state is calculated. Due to the flexible schema, referencing of activities can be based on any available activity attribute. The *identifier* attribute in a

fromActivities sub-element identifies the attribute that is used for identification while *value* denotes the value of the attribute. In the example shown in Listing 4 a BPEL activity is identified by *name* – and a Chevron activity is identified by *id*.

```
<?xml version="1.0" encoding="UTF-8"?>
<stateProjection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:noNamespaceSchemaLocation="rules_schema.xsd">
      <statePropagationSets>
            <statePropagationSet name="combination">
               <condition>
                  <containState>Active;Failed;Suspended;Terminated</containState>
                  <targetState>Running</targetState>
               </condition>
               <condition>
                  <allStatesEqual>Completed</allStatesEqual>
                  <targetState>Completed</targetState>
               </condition>
               <else>Outstanding</else>
            </statePropagationSet>
            ...
      </statePropagationSets>

      <statePropagationRules>
            <statePropagationRule name="production">
                  <fromActivities>
                        <activity identifier="name"
                                  value="Prepare Production"/>
                        <activity identifier="name"
                                  value="Invoke Production Sub-Process"/>
                        <activity identifier="name"
                                  value="Await completion of Production"/>
                  </fromActivities>
                  <toActivity>
                        <activity identifier="id" value="2.2"/>
                  </toActivity>
                  <stateSet pattern="combination"/>
            </statePropagationRule>
            ...
      </statePropagationRules>
</stateProjection>
```

Listing 4 XML excerpt of a state projection definition.

6.5 XML Excerpt of a Stateful BPEL Process

To be able to perform the propagation of states from a BPEL process instance, the instance state needs to be known. We can distinguish two basic approaches to make these states available for further processing. One possibility is to provide a list of activities and corresponding states, separated from the process model. The other possibility is to provide a BPEL process model that is already augmented with the execution states. For instance, on the one hand it is an advantage in having an already augmented model, as it can be easily queried using XPath expressions. On the other hand, a list of activities and states can be processed very fast. The reason to implement the approach with an augmented process model lies in the flexible activity referencing mechanism we designed in the state propagation rules. In the rules any available attribute/value pair can be used for identification of a *fromActivity*. A simple list of activities and corresponding states cannot support this mechanism.

Listing 5 shows an excerpt of a BPEL process model that is augmented with instance information. The states extension is properly declared so that an augmented process can still be opened in a standard-conform BPEL process design tool. The state of an activity is indicated by a sub-element

(*state:activityStatus*) that uses a different namespace than the standard BPEL constructs. If no such sub-element exists for an activity, it is assumed that this activity is in the state *No Status*.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process exitOnStandardFault="yes" name="Ordering-Process"
              suppressJoinFailure="yes"
              targetNamespace="http://iaas.orderingProcess.com"
              xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
              xmlns:tns="http://iaas.orderingProcess.com"
              xmlns:state="http://iaas.orderingProcess.com/states">
    <!-- definition of the states extension-->
    <extensions>
       <extension namespace="http://iaas.orderingProcess.com/states"
                  mustUnderstand="no"/>
    </extensions>
    ...
    <bpws:sequence name="main">
            <state:activityStatus>Active</state:activityStatus>
        <bpws:receive createInstance="yes" name="receive Order" ...>
           <state:activityStatus>Completed</state:activityStatus>
        </bpws:receive>
        <bpws:invoke name="prepare Order Processing" ...>
           <state:activityStatus>Active</state:activityStatus>
        </bpws:invoke>
        ...
    </bpws:sequence>
</bpws:process>
```

Listing 5 XML excerpt of a stateful BPEL process, adapted from [4].

6.6 SVG Template for a Chevron Activity

In Listing 6, the SVG template for a simple Chevron activity is shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
     xmlns:xlink="http://www.w3.org/1999/xlink">

<!-- definition of Chevron Activity using Polygon -->
<polygon points="X,Y+185 X,Y+185 X,Y+215 X,Y+245 X,Y+245 X,Y+215"
         style="fill:url(#COLOR#);stroke:#STROKE_COLOR#;stroke-width:2"/>

  <!-- definition of activity caption -->
  <text x="#X#" y="#Y#+215" style="fill:black"
           font-family="#FONT#"
           font-size="#FONT-SIZE#">#CAPTION#</text>

  <!-- definition of activity status and icon -->
  <image x="#X#" y="#Y#+165" width="40" height="40"
           xlink:href="#STATUS#"/>
  <image x="X" y="Y+195" width="50" height="50"
           xlink:href="#ICON#"/>
</svg>
```

Listing 6 SVG template for a Chevron activity, adapted from [4].

The template is parameterizable with the following parameters:

- #X#: This is a relative x-coordinate to position an activity in the overall Chevron process.
- #Y#: This is a relative y-coordinate to position an activity in the overall Chevron process.
- #COLOR#: This is a reference to a color definition (not shown in Listing 6).
- #STROKE-COLOR#: The reference for the color that strokes the activity shape.
- #CAPTION#: The caption of the Chevron activity.
- #FONT#: A reference to the font that is used for the caption.
- #FONT-SIZE#: Size of the caption font.
- #STATUS#: The icon that graphically indicated the current status.
- #ICON#: The icon that graphically indicates the semantics of the Chevron activity.

6.7 Graphical Example of a Chevron Activity

In Figure 8, a graphical example of a simple Chevron activity is shown. The rendering of the SVG was performed in the Mozilla Firefox browser[3].

Figure 8 Graphical example of an SVG-based Chevron activity.

6.8 State Propagation Pattern Support

The presented implementation artifacts enable the use of the patterns *Direct State Propagation*, *State Alteration* and *State Combination*. The *Transition State* pattern is not applicable as there are no control link elements in the Chevron language. The *State Deduction* pattern is applicable, but not considered in the current version. Furthermore, defining state projections with multiple source models is not yet considered.

The current version of the presented library does not support the pattern *Instance State Aggregation*, but it can be easily extended to support this pattern: The XML schema of the Chevron language contains a restriction which imposes that a Chevron activity may only have one of the pre-defined Chevron activity states, namely "Outstanding", "Running", "Completed", or an empty string to indicate that no information is available. To consider the instance state aggregation patterns only minor changes in the schema are required. One change is to remove the state type restriction and allow a String. This way an array of states can be serialized as a comma-separated list. The other change is to add a Boolean attribute "append" to the process element to indicate if the states should be appended to the elements to the Chevron model that is target of the projection. The actual appending of the state would be performed during state propagation, see Task "Perform state propagation" in Section 5. As illustrated in Figure 9, instance state aggregation can be enabled that way.

[3] Mozilla Foundation: Mozilla Firefox, http://www.mozilla.com/en-US/firefox/new/, 2011.

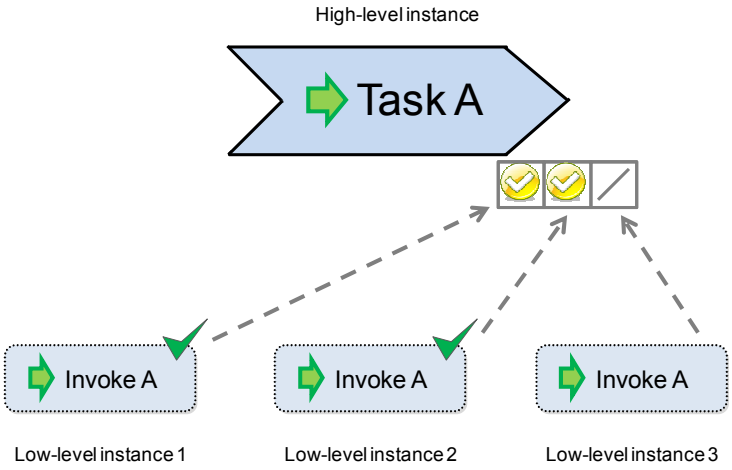

Figure 9 Concept of instance state aggregation for the Chevron language.

7 Example Scenario

In Figure 10 an example scenario for the state projection procedure is depicted. On the left hand side, a BPEL process model is shown (a). The graphical representation is taken from the graphical process design tool Eclipse BPEL Designer[4]. The BPEL process implements order processing functionality:

- After receiving a request message the order is prepared
- Then, product availability is checked
- If the product is not available, it is produced on-demand
- Then, shipping of the ordered product is prepared and performed
- Finally, the order process is finished
- Fault handling functionality is included to notify an operator in case of unexpected problems.

The Chevron model in Figure 10 (b) is meant for customers who are interested in the current status of their ordering. The Chevron model has three main steps: order receiving, production, and delivery. No technical details are included and activities have different labels and icons compared to the BPEL process. Besides, the process contains much less activities, and it even contains an activity that is not reflected in the BPEL model at all, “Check product”. In this scenario, this activity has been included to increase customer satisfaction with the product and the overall process. Of course, the product is being checked during and after production, but there is no explicit activity for that in the BPEL process.

The arrows in Figure 10 define the sources and targets of state propagation rules. The caption on the arrows refers to a particular state mapping set. Three different mapping sets have been defined for this scenario, “Alteration A”, “Alteration B”, and “Combine”:

- Alteration A
  - Active, Suspended, Failed → Running
  - Completed, Terminated → Completed
  - No status → Outstanding

[4] The Eclipse Foundation: Eclipse BPEL Designer, http://www.eclipse.org/bpel/, 2011.

- Alteration B
    - Completed → Completed
    - Other → Outstanding
- Combine
    - All status are No status → Outstanding
    - All status are Completed → Completed
    - Other → Running

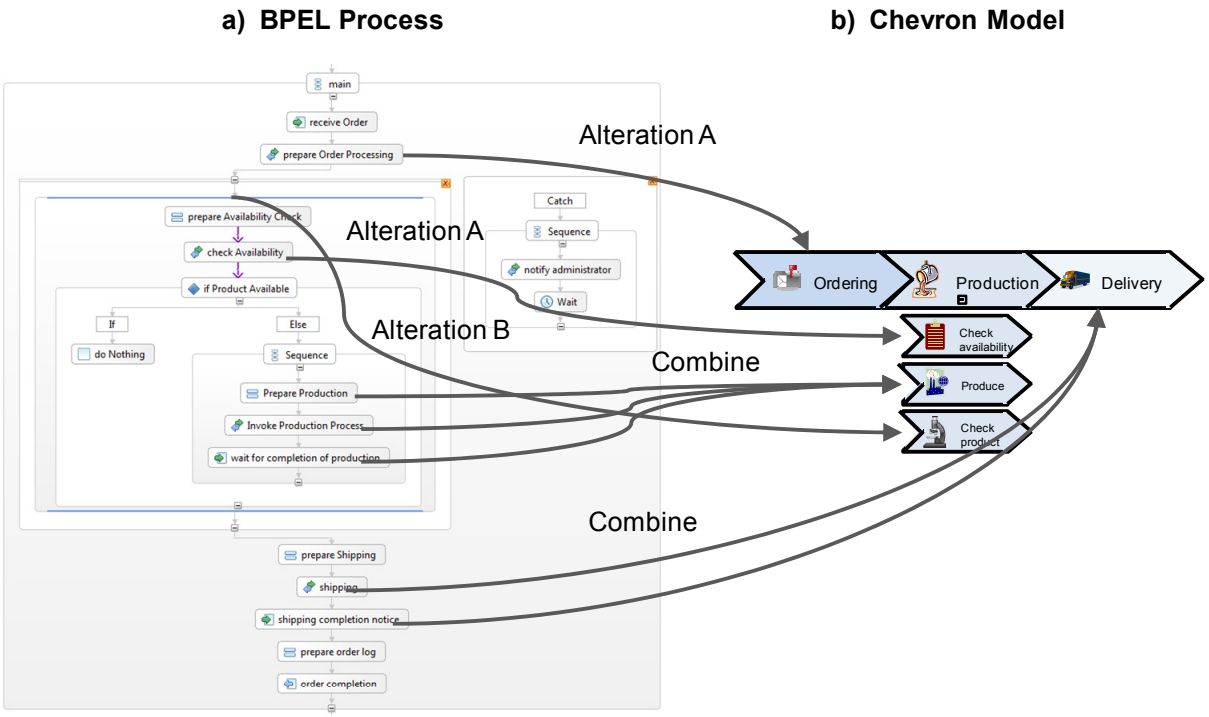

Figure 10 Concept of a state projection from an ordering process in BPEL (a) to a Chevron model (b).

The following listings show the different artifacts to implement this scenario:

- Figure 11 shows a detailed graphical view of the BPEL process (exported from Eclipse BPEL Designer)
- Listing 7 shows the code of the stateful BPEL process (modeled in Eclipse BPEL Designer)
- Listing 8 shows the code of the Chevron model (modeled using Eclipse XML tools)
- Listing 9 shows the state projection definition (modeled using Eclipse XML tools)
- Listing 10 shows the code of the Chevron model after augmentation with propagated states (augmented through invocation of the code library)
- Listing 11 shows the SVG code generated out of the augmented Chevron model (generated by the code library)
- Figure 12 shows the graphical rendering of the Chevron model in SVG (viewed in the Mozilla Firefox browser)

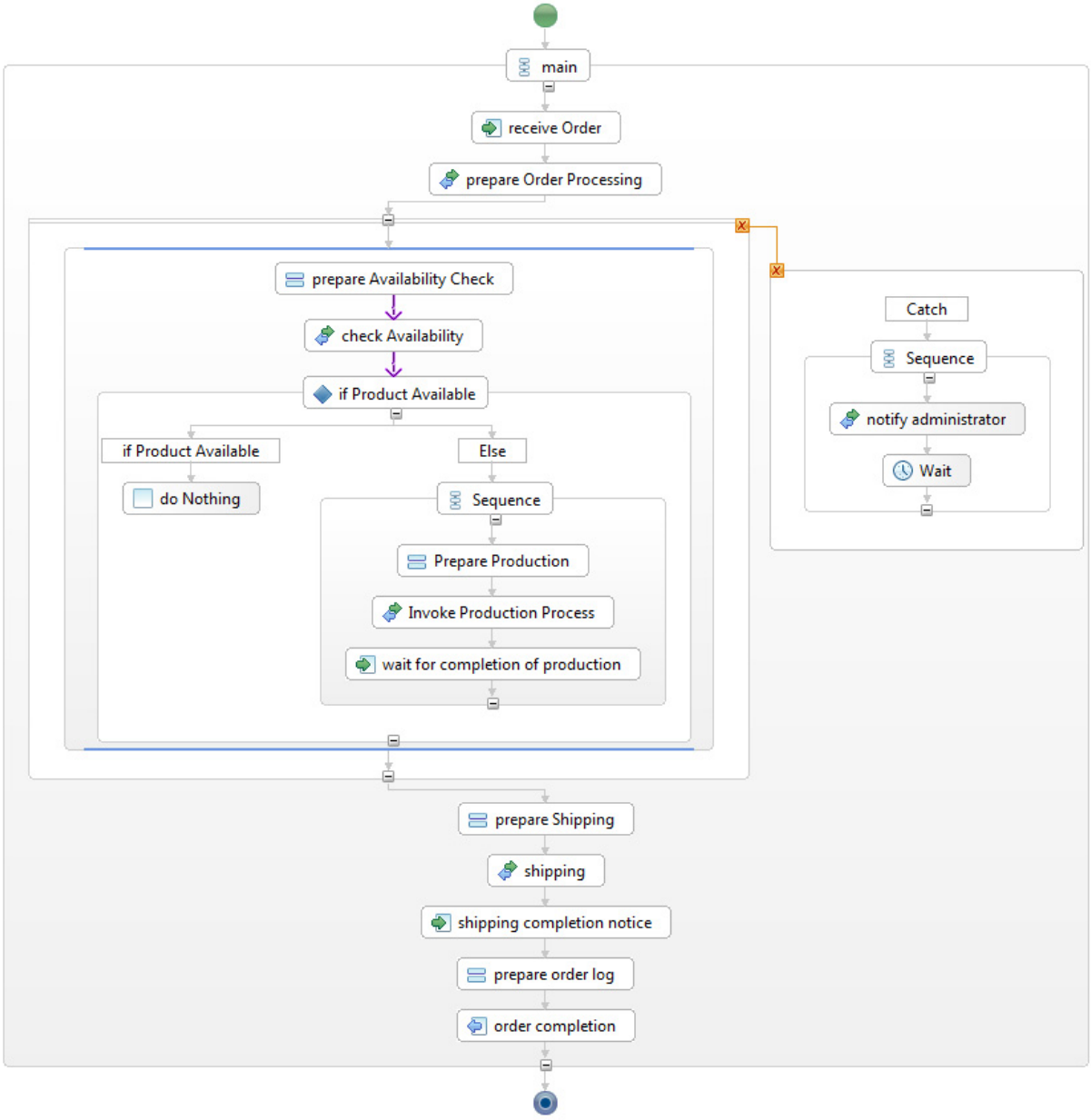

Figure 11 Graphical rendering of the BPEL process model in Eclipse BPEL Designer.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process exitOnStandardFault="yes" name="Ordering-Process"
    suppressJoinFailure="yes"
    targetNamespace="http://iaas.orderingProcess.com"
    xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:tns="http://iaas.orderingProcess.com"
    xmlns:state="http://iaas.orderingProcess.com/states">
    <bpws:import importType="http://schemas.xmlsoap.org/wsdl/"
                 location="Ordering-Process.wsdl"
                 namespace="http://iaas.orderingProcess.com"/>
    <bpws:partnerLinks>
        <bpws:partnerLink myRole="Ordering-ProcessProvider"
                          name="client" partnerLinkType="tns:Ordering-Process"
                          partnerRole="Ordering-ProcessRequester"/>
```

```
    </bpws:partnerLinks>
    <bpws:variables>
        <bpws:variable messageType="tns:Ordering-ProcessRequestMessage"
                       name="input"/>
        <bpws:variable messageType="tns:Ordering-ProcessResponseMessage"
                       name="output"/>
    </bpws:variables>
    <extensions>
        <extension namespace="http://iaas.orderingProcess.com/states"
                   mustUnderstand="no"/>
    </extensions>

    <bpws:sequence name="main">
       <state:activityStatus>Active</state:activityStatus>
        <bpws:receive createInstance="yes" name="receive Order"
            operation="initiate" partnerLink="client"
            portType="tns:Ordering-Process" variable="input">
            <state:activityStatus>Completed</state:activityStatus>
        </bpws:receive>
        <bpws:invoke name="prepare Order Processing">
              <state:activityStatus>Completed</state:activityStatus>
        </bpws:invoke>
        <bpws:scope name="Scope">
            <state:activityStatus>Active</state:activityStatus>
            <bpws:faultHandlers>
                <bpws:catch>
                    <bpws:sequence>
                        <bpws:invoke name="notify administrator"/>
                        <bpws:wait name="Wait"/>
                    </bpws:sequence>
                </bpws:catch>
            </bpws:faultHandlers>
            <bpws:flow name="Flow">
             <state:activityStatus>Active</state:activityStatus>
                <bpws:links>
                    <bpws:link name="link1"/>
                    <bpws:link name="link2"/>
                </bpws:links>
                <bpws:invoke name="check Availability">
                    <state:activityStatus>Completed</state:activityStatus>
                    <bpws:targets>
                        <bpws:target linkName="link1"/>
                    </bpws:targets>
                    <bpws:sources>
                        <bpws:source linkName="link2"/>
                    </bpws:sources>
                </bpws:invoke>
                <bpws:assign name="prepare Availability Check" validate="no">
                    <state:activityStatus>Completed</state:activityStatus>
                    <bpws:sources>
                        <bpws:source linkName="link1"/>
                    </bpws:sources>
                </bpws:assign>
                <bpws:if name="if Product Available">
                    <bpws:targets>
                        <bpws:target linkName="link2"/>
                    </bpws:targets>
                    <bpws:empty name="do Nothing"/>
                    <bpws:else>
                        <bpws:sequence name="Sequence">
                            <bpws:assign name="Prepare Production" validate="no">
                              <state:activityStatus>
                                 Completed
                              </state:activityStatus>
                            </bpws:assign>
                            <bpws:invoke name="Invoke Production Process">
                                <state:activityStatus>Ready</state:activityStatus>
                            </bpws:invoke>
```

```
                        <bpws:receive name="wait for completion of
                                                    production"/>
                    </bpws:sequence>
                </bpws:else>
            </bpws:if>
        </bpws:flow>
    </bpws:scope>
    <bpws:assign name="prepare Shipping" validate="no"/>
    <bpws:invoke name="shipping"/>
    <bpws:receive name="shipping completion notice"/>
    <bpws:assign name="prepare order log" validate="no"/>
    <bpws:reply name="order completion"/>
  </bpws:sequence>
</bpws:process>
```

Listing 7 Stateful BPEL process.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<chevronProcessModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      pid="15.1" piid="15" processModelDocumentation="documentation"
      processModelName="example" processModelVersion="1"
      xsi:noNamespaceSchemaLocation="chevron.xsd">
      <Activity activityCaption="Ordering" activityColor="blue"
            activityDocumentation="order receipt" activityID="1"
            activityIconForCaption="order.png">
            <activityStatus></activityStatus>
      </Activity>
      <complexActivity activityCaption="Production" activityColor="blue"
                  activityDocumentation="production of ordered product"
                  activityID="2" activityIconForCaption="production.png"
                  activityIconForComplexActivity="plus.png">
            <complexActivityStatus></complexActivityStatus>
            <Activity activityCaption="Check availability" activityColor="blue"
                  activityDocumentation="availability check" activityID="2.1"
                  activityIconForCaption="check-availability.png">
                  <activityStatus></activityStatus>
            </Activity>
            <Activity activityCaption="Produce" activityColor="blue"
                  activityDocumentation="production" activityID="2.2"
                  activityIconForCaption="production-2.png">
                  <activityStatus></activityStatus>
            </Activity>
            <Activity activityCaption="Check product" activityColor="blue"
                  activityDocumentation="product quality assurance"
                  activityID="2.3" activityIconForCaption="check-product.png">
                  <activityStatus></activityStatus>
            </Activity>
      </complexActivity>
      <Activity activityCaption="Check product" activityColor="blue"
            activityDocumentation="checking product" activityID="3"
            activityIconForCaption="delivery.png">
            <activityStatus></activityStatus>
      </Activity>
</chevronProcessModel>
```

Listing 8 Stateless Chevron model.

```
<?xml version="1.0" encoding="UTF-8"?>
<stateProjection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:noNamespaceSchemaLocation="rules_schema.xsd">

	<!-- State Mappings -->
	<statePropagationSets>
		<statePropagationSet name="Alteration-A">
			<condition>
				<containState>Active;Suspended;Failed</containState>
				<targetState>Running</targetState>
			</condition>
			<condition>
				<containState>Completed;Terminated</containState>
				<targetState>Completed</targetState>
			</condition>
			<condition>
				<containState>No Status</containState>
				<targetState>Outstanding</targetState>
			</condition>
			<else>Outstanding</else>
		</statePropagationSet>
		<statePropagationSet name="Alteration-B">
			<condition>
				<containState>Completed</containState>
				<targetState>Completed</targetState>
			</condition>
			<else>Outstanding</else>
		</statePropagationSet>
		<statePropagationSet name="Combination">
			<condition>
				<allStatesEqual>Completed</allStatesEqual>
				<targetState>Completed</targetState>
			</condition>
			<condition>
				<allStatesEqual>No Status</allStatesEqual>
				<targetState>Completed</targetState>
			</condition>
			<else>Running</else>
		</statePropagationSet>
	</statePropagationSets>

	<!-- State Propagation Rules -->
	<statePropagationRules>
		<statePropagationRule name="Ordering">
			<fromActivities>
				<activity identifier="name"
					value="prepare Order Processing" />
			</fromActivities>
			<toActivity>
				<activity identifier="id" value="1" />
			</toActivity>
			<stateSet pattern="Alteration-A" />
		</statePropagationRule>
		<statePropagationRule name="Check availability">
			<fromActivities>
				<activity identifier="name" value="check Availability" />
			</fromActivities>
			<toActivity>
				<activity identifier="id" value="2.1" />
			</toActivity>
			<stateSet pattern="Alteration-A" />
		</statePropagationRule>
		<statePropagationRule name="Produce">
			<fromActivities>
				<activity identifier="name" value="Prepare Production" />
				<activity identifier="name"
					value="Invoke Production Process" />
```

```
                        <activity identifier="name"
                                value="wait for completion of production" />
                </fromActivities>
                <toActivity>
                        <activity identifier="id" value="2.2" />
                </toActivity>
                <stateSet pattern="Combination" />
        </statePropagationRule>
        <statePropagationRule name="Check Product">
                <fromActivities>
                        <activity identifier="name" value="Flow" />
                </fromActivities>
                <toActivity>
                        <activity identifier="id" value="2.3" />
                </toActivity>
                <stateSet pattern="Alteration-B" />
        </statePropagationRule>
        <statePropagationRule name="Delivery">
                <fromActivities>
                        <activity identifier="name" value="shipping" />
                        <activity identifier="name"
                                value="shipping completion notice" />
                </fromActivities>
                <toActivity>
                        <activity identifier="id" value="3" />
                </toActivity>
                <stateSet pattern="Combine" />
        </statePropagationRule>
    </statePropagationRules>
</stateProjection>
```

Listing 9 State projection definition for a particular BPEL process model and a particular Chevron model. It is independent of a particular process instance or point in time.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<chevronProcessModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    pid="15.1" piid="15" processModelDocumentation="documentation"
    processModelName="example" processModelVersion="1"
    xsi:noNamespaceSchemaLocation="chevron.xsd">
    <Activity activityCaption="Ordering" activityColor="blue"
        activityDocumentation="order receipt" activityID="1"
        activityIconForCaption="order.png">
        <activityStatus>Completed</activityStatus>
    </Activity>
    <complexActivity activityCaption="Production" activityColor="blue"
                activityDocumentation="production of ordered product"
                activityID="2" activityIconForCaption="production.png"
                activityIconForComplexActivity="plus.png">
        <complexActivityStatus>Running</complexActivityStatus>
        <Activity activityCaption="Check availability" activityColor="blue"
            activityDocumentation="availability check" activityID="2.1"
            activityIconForCaption="check-availability.png">
            <activityStatus>Completed</activityStatus>
        </Activity>
        <Activity activityCaption="Produce" activityColor="blue"
            activityDocumentation="production" activityID="2.2"
            activityIconForCaption="production-2.png">
            <activityStatus>Running</activityStatus>
        </Activity>
        <Activity activityCaption="Check product" activityColor="blue"
            activityDocumentation="product quality assurance"
            activityID="2.3" activityIconForCaption="check-product.png">
            <activityStatus>Outstanding</activityStatus>
        </Activity>
    </complexActivity>
```

```
      <Activity activityCaption="Check product" activityColor="blue"
            activityDocumentation="checking product" activityID="3"
            activityIconForCaption="delivery.png">
            <activityStatus>Outstanding</activityStatus>
      </Activity>
</chevronProcessModel>
```

Listing 10 Stateful Chevron model.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     height="200%" version="1.1" width="200%">
      <defs>
            <linearGradient id="blue_aqua" x1="0%" x2="0%" y1="0%"
                  y2="120%">
                  <stop offset="0%" style="stop-color:#00ffff;stop-opacity:1" />
                  <stop offset="100%" style="stop-color:#0000ff;stop-opacity:1" />
            </linearGradient>
      </defs>
      <polygon points="40,80 240,80 280,120 240,160 40,160 80,120 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <text font-family="Franklin Gothic Heavy" font-size="20" style="fill:black"
            x="150" y="125">Ordering</text>
      <image height="65" width="65" x="80" xlink:href="\img\ordering.png"
            y="90" />
      <image height="50" width="50" x="210" xlink:href="Completed.png"
            y="55" />
      <polygon points="260,80 460,80 500,120 460,160 260,160 300,120 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <text font-family="Franklin Gothic Heavy" font-size="20" style="fill:black"
            x="370" y="125">Production</text>
      <image height="65" width="65" x="300" xlink:href="\img\production.png"
            y="90" />
      <image height="50" width="50" x="430" xlink:href="Running.png" y="55" />
      <polygon points="290,185 440,185 470,215 440,245 290,245 320,215 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <image height="50" width="50" x="320" xlink:href="\img\check.png"
            y="195" />
      <image height="40" width="40" x="415" xlink:href="Completed.png"
            y="165" />
      <polygon points="290,270 440,270 470,300 440,330 290,330 320,300 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <image height="50" width="50" x="320" xlink:href="\img\produce.png"
            y="280" />
      <image height="40" width="40" x="415" xlink:href="Running.png" y="250" />
      <polygon points="290,355 440,355 470,385 440,415 290,415 320,385 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <text font-family="Franklin Gothic Heavy" font-size="16" style="fill:black"
            x="375" y="215">Check</text>
      <text font-family="Franklin Gothic Heavy" font-size="14" style="fill:black"
            x="375" y="230">availability</text>
      <text font-family="Franklin Gothic Heavy" font-size="16" style="fill:black"
            x="375" y="300">Produce</text>
      <text font-family="Franklin Gothic Heavy" font-size="16" style="fill:black"
            x="375" y="385">Check</text>
      <text font-family="Franklin Gothic Heavy" font-size="16" style="fill:black"
            x="375" y="400">product</text>
      <image height="50" width="50" x="320" xlink:href="\img\check-product.png"
            y="365" />
      <image height="40" width="40" x="415" xlink:href="Outstanding.png"
            y="335" />
      <polygon points="480,80 680,80 720,120 680,160 480,160 520,120 "
            style="fill:url(#blue_aqua);stroke:#0A0080;stroke-width:2" />
      <text font-family="Franklin Gothic Heavy" font-size="20" style="fill:black"
```

```
            x="590" y="125">Delivery</text>
      <image height="65" width="65" x="520" xlink:href="\img\deliver.png"
            y="90" />
      <image height="50" width="50" x="650" xlink:href="Outstanding.png"
            y="55" />
</svg>
```

Listing 11 Stateful Chevron model in SVG.

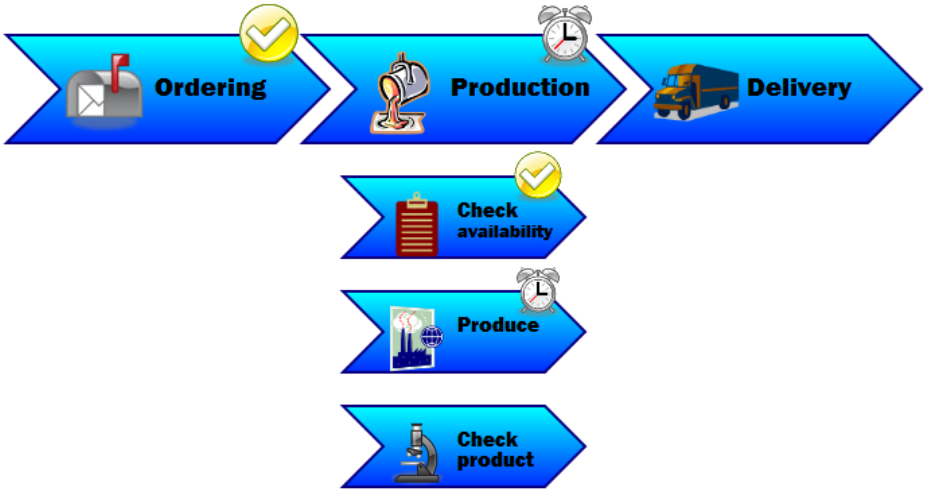

Figure 12 Graphical rendering of the stateful Chevron model in SVG

8 Discussion

In this report we presented a means to monitor a BPEL process instance in a different process model, specified in a different language. We demonstrated this approach with the Chevron language. However, the approach for state propagation is generic and can be applied for other languages, on the lower level as well as on the higher level. For business stakeholders further process language are common on higher level of abstraction, such as EPC or BPMN. On the other side, there are various further languages used for workflow execution, such as YAWL. State propagation is also applicable in these scenarios, whereas this report create a basis for further (applied) research.

One aspect we did not address in this report is the semantic relationship between the low-level process that is being executed and the higher-level process that is being monitored. The high-level models which are used for monitoring can be modeled from scratch, detached from the actual execution model. Using the presented approach, state projections make it possible to significantly change the visual impression and semantics of what is actually being executed. Process structures which are executed in the lower-level model might not be contained in the higher-level model at all. Activities shown in a higher-level model are not necessarily contained in the execution model. Change of activity labels and icons contributes to this further. In some cases this relaxation of semantics may be beneficial, while in other scenarios this freedom needs to be limited. Possibly, in some business scenarios a proof must be made that the high-level model “in principle” conforms to the low-level model. Further research is required in this field.

Acknowledgements

The authors David Schumm and Dimka Karastoyanova would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

References

[1] D. Schumm, G. Latuske, F. Leymann, R. Mietzner, T. Scheibler. State Propagation for Business Process Monitoring on Different Levels of Abstraction. Proceedings of the 19th European Conference on Information Systems (ECIS 2011), 2011.

[2] D. Schumm, G. Latuske, F. Leymann. A Prototype for View-based Monitoring of BPEL Processes. Technical Report No. 2011/04, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2011.

[3] Organization for the Advancement of Structured Information Standards (OASIS). Business Process Execution Language 2.0 (BPEL). OASIS Standard, 2007.

[4] S. Lie. Abstract Business Process Monitoring. Student Thesis No. 2315, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2011.

[5] D. Schumm, F. Leymann, A. Streule. Process Viewing Patterns. Proceedings of the 14th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2010), IEEE Computer Society Press, 2010.

[6] R. Eshuis and P. Grefen. Constructing Customized Process Views. In: Data & Knowledge Engineering, 64(2):419–438, Elsevier, 2008.

[7] R. Bobrik, M. Reichert, T. Bauer. View-Based Process Visualization. Proceedings of the 5th International Conference on Business Process Management (BPM'07), Springer, 2007.

[8] Workflow Management Coalition (WfMC): Workflow Management Application Programming Interface (Interface 2 & 3) Specification Document Number WFMC-TC-1009. Version 2.0e, 1998.

[9] D. L. Moody. The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, 35:756–779, IEEE, 2009.

[10] Object Management Group (OMG): Business Process Model and Notation (BPMN), Version 2.0, OMG Document Number formal/2011-01-03, January 2011.