Process Modeling Using Event-Driven Process Chains

Jan Kramer
3233561
TA: Daniel Alami
Department of Information and Computing Sciences
Utrecht University

1 Introduction

Event-Driven Process Chains (EPC) is a business process modeling language developed in a joint effort between SAP and the Institute of Information Systems of Saarbrücken in the context of the Architecture of Integrated Information Systems (ARIS) framework (Keller, Scheer, & Nüttgens, 1992). One of the initial purposes of EPC was to configure workflows in ERP software such as SAP R/3. As the language is aimed at regular users, it is designed to be intuitive and easy to understand, in contrast to more formal languages such as Petri nets which have a stronger focus on correctness (Aalst, 1999).

No standardized procedure exists for producing EPC models, but Scheer, Thomas, and Adam (2005, pp. 136–137) describe a set of steps that help create EPC in a structured manner:

1. Define a unique title of the business process, to make clear which process is modeled.
2. Derive initial and final events from the process title.
3. Look for relevant verbs in the process description, translate them to a function flow, consisting of functions, logical connectors and process interfaces where applicable. The elements should be arranged in the order they are executed in. Where necessary, add logical connectors to split or join the execution.
4. For each transition between two functions, define one or more events in such way that the preceding function produces the event, and the event triggers the next event.
5. Verify the model for structural correctness, and perform validation with key stakeholders in the process.

Please note that one of the original steps described by Scheer et al. requires the use of other aspects of the ARIS framework. Since this paper focuses on EPC as a process modeling language, this step has been omitted here.

As the original paper (Keller et al., 1992) does not include a formal definition of the syntax, multiple variations and extensions of the notation have emerged over time. Mendling (2008) compared these variations in an effort to develop
a unified syntax, which will be used in the remainder of this paper. Note that while other literature also includes additional elements such as organizational units and IT systems (e.g. Becker, Rosemann, & Uthmann, 2000; Nüttgens, Feld, & Zimmermann, 1998), this paper focuses on the elements presented by Mendling.

According to Mendling, an EPC consists of the following elements:

– **Function**: Represents an activity or task.
– **Event**: Describes the situation before and/or after a function.
– **Control flow**: A directed arc which connects two EPC elements.
– **Logical connector**: Connects three or more functions and events. Each connector can either be a split (multiple outbound flows), or a join (multiple inbound flows). The connector type defines the number of flows activated (split) or waited for (join): exactly one for exclusive OR (XOR), one or more for OR, and all flows for ALL connectors.
– **Process interface (optional)**: Represents another EPC that is consecutively executed in relation to the current EPC. Can be placed at either the start or the end of the EPC.

Please refer to Figure 1 for the graphical notation of these elements.

![Figure 1. Syntax elements of Event-Driven Process Chains](image)

One of the key people in developing EPC, Professor August-Wilhelm Scheer, has both a background in business and academia. He was chair of the Institute for
Information Systems at Saarland University in Germany from 1975 to 2005, and founded the software and consulting company IDS Scheer AG in 1984. Currently he resides as a consulting professor at the German Research Centre for Artificial Intelligence (DFKI) (Institut für Wirtschaftsinformatik, n.d.).

2 Example

In the following section, the procedure of process modeling using EPC is explained by applying the steps as described by Scheer et al. (2005, pp. 136–137) to a fictional process description. Consider a product software development company that has defined the following process for handling bug reports:

"All bugs that are encountered by both customers and internal stakeholders should be filed as an issue in our issue tracker. When a bug report is filed, the product manager (PM) checks whether the issue is reproducible and valid. If it is not reproducible or in fact not a bug, the PM discards the issue and notifies the user that the issue was discarded. Otherwise, the bug is assigned to the development team. As soon as the development team has resolved the issue, the PM updates the release notes of the upcoming product release and informs the user that the bug will be resolved in the next version."

Applying the aforementioned steps to this description results in the EPC as depicted in Figure 2. The remainder of this section discusses how this model has been established.

From the description, it becomes clear that the process is concerned with the workflow to handle bugs. Therefore, the name of this process is defined as "Bug Handling Workflow."

The process appears to be triggered when a user or developer encounters a bug, so the initial event of the process is "Bug encountered". Furthermore, the process ends after an issue is handled, so this process has one final event named "Issue handled".

With the initial and final events in place, the process description is scanned for relevant verbs which are translated to functions. From the example, the following functions are derived: "File issue", "Check issue", "Discard issue", "Notify user", "Assign issue", "Resolve issue", "Update release notes" and "Inform user".

Additionally, in several places logical connectors are added to specify the control flow. After "Check issue", it is either valid, invalid or not reproducible. Because only a single flow is possible, this is represented by a XOR connector. Note that a second XOR connector is added before the final 'Issue handled' event, in order to synchronize the multiple flows. Also, the 'Issue resolved' event has multiple outbound flows, but in this case all functions should be triggered and therefore an AND connector is appropriate.

After the functions and connectors have been added, only the events remain to be added. Between each two consecutive functions, at least one event is added.
Figure 2. Event-Driven Process Chain of the process "Bug Handling Workflow"
Often the event is straightforward, such as the function 'Discard issue' resulting in the event 'Issue discarded' which in turn triggers 'Notify user of discarded issue'. While this one-to-one mapping is prevalent in simple models, events can in fact trigger multiple functions, as seen after 'Issue resolved', which triggers both 'Inform user on upcoming release' and 'Update release notes'.

At the current stage, a full EPC has been modeled that as a final step should be conferred with key stakeholders in order to validate whether the model corresponds to reality. In this case, a PM might for example feel that the scenario of duplicate issues is insufficiently handled, which after discussion could result in the addition of an event after 'Check issue'.

Finally, when all key stakeholders agree on the EPC, it is ready to use.

3 Related Literature

As briefly mentioned before, EPC was first introduced by Keller et al. (1992) in the context of the ARIS framework. The ARIS framework aims to describe enterprises by integrating five perspectives: data, function, organisation, output, and process. Within this context, Keller et al. developed EPC as the language to model the process views of an organization.

The EPC language is originally based on the concepts of Petri nets and stochastic networks, but is less rigid than those methods as it does not require a formal framework (Scheer et al., 2005). As a consequence, EPC models based on the original description are not easily verified. However, multiple efforts have been made to address this issue by formalizing the EPC. Examples of such efforts include Aalst (1999), Mendling and Aalst (2007), and Mendling (2008). In all cases, formal definitions of the syntax are defined using logic and set theory from mathematics.

Comparing EPC to UML State Diagrams (SD) and BPM, two other process modeling languages, Söderström, Andersson, Johannesson, Perjons, and Wangler (2002) found as a difference that EPC does not explicitly use the concept of state. SD and BPM include various event types that represent a state change, however this is not the case in EPC. As a result, the EPC syntax requires less symbols, at the cost of being more implicit.

In an experiment to compare EPC and Petri net models from the end-user perspective, Sarshar and Loos (2005) found that the perceived ease-of-use was highest for EPC and BPM models. However, the non-local semantics of the OR connector, i.e. the OR connector depends on information from other parts of the model than its direct neighbours, negatively impacted its comprehension.

Finally, an interesting use of EPC models has been introduced by Dongen, Dijkman, and Mendling (2008). In their paper, Dongen et al. describe the situation of large organizations that have thousands of process models. To increase the quality of the overall set of models, duplication in models should be prevented. To assist maintainers of such repositories, they propose a novel approach to detect similarity between process models such as EPC, based on comparing abstract representations of EPC, i.e. causal footprints.
4 Process-Deliverable Diagram

This document describes a meta-model for the technique Process Modeling Using Event-Driven Process Chains (EPC), which is represented as a Process-Deliverable Diagram (PDD) in Figure 3. EPC is a business process modeling language first introduced by Keller et al. (1992), and is mainly used to configure workflows in the domain of ERP software.

A PDD is the result of a meta-modeling technique in which both the activities, and concepts of a method or technique are described. The activities represent the process and are depicted on the left-hand side of the PDD in the form of a UML activity diagram, while the concepts represent the deliverables and are depicted on the right-hand side represented as a UML class diagram. This meta-model allows method engineers to assemble situational methods by analyzing and selecting relevant method fragments (Weerd & Brinkkemper, 2008).

The PDD has been created based on both the work of Scheer et al. (2005) and Mendling (2008), where the former mainly provided input on the procedural aspect, while the latter source provided a comprehensive description of the notational aspects of Process Modeling using EPC.

In general, the activities of the technique are divided in three main phases: Analysis, Design and Evaluation. In the Analysis phase, the process to be modeled is analyzed and translated to EPC on a conceptual level. The Design phase is concerned with drawing the actual process model, while finally during the Evaluation phase, the produced model is verified and validated. The main deliverable of the process is the EPC model, which comprehensively describes a specific business process.

The activities and concepts shown in Figure 3 are described in more detail in respectively Table 1 and 2.
Rule: An EPC consists of at least one start event, one end event and one function.

Rule: It is allowed skip sub activities in order to go directly to the sub-activity that is concerned with the error.

**Figure 3.** Process-Deliverable Diagram of Process Modeling using EPC
Table 1. Activity table for the PDD of Process Modeling using EPC

<table>
<thead>
<tr>
<th>Activity</th>
<th>Sub-Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Acquire process description</td>
<td>The business process to be modeled is described by a PROCESS DESCRIPTION in natural language, which has to be acquired by the process modelers to base the EPC on (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
<tr>
<td></td>
<td>Define unique title</td>
<td>A unique title is required to distinguish the EPC from other EPC models. A good title also makes it clear to everyone involved what the topic of the process is (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
<tr>
<td></td>
<td>Derive start and end events</td>
<td>By defining the start and end EVENTS, the context of the process becomes clear. Each EPC has at least one start EVENT and at least one end EVENT (Mendling, 2008).</td>
</tr>
<tr>
<td></td>
<td>Translate verbs to functions</td>
<td>In addition to EVENTS, an EPC also consists of one or more FUNCTIONs and zero or more PROCESS INTERFACEs which represent the activity of the actors in the process. These can be discovered by identifying relevant verbs from the PROCESS DESCRIPTION. A PROCESS INTERFACE differs from a FUNCTION in that it represents another process that is in turn described by an EPC (Mendling, 2008).</td>
</tr>
<tr>
<td>Design</td>
<td>Draw nodes</td>
<td>After the analysis phase is finished, the identified nodes, i.e. EVENTS, FUNCTIONs and PROCESS INTERFACEs must be drawn using the EPC notation. The nodes must be arranged according to the order of execution (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
<tr>
<td></td>
<td>Draw logical connectors</td>
<td>Use LOGICAL CONNECTORs to properly model the control flow, i.e. AND where all branches, XOR where one, and OR where at least one branch must be executed (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
<tr>
<td></td>
<td>Draw intermediate events</td>
<td>For each transition from one FUNCTION to another, determine at least one EVENT that describes the post-conditions of the source FUNCTION and the pre-conditions of the target FUNCTION (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
<tr>
<td></td>
<td>Draw control flows</td>
<td>Model all transitions between the EPC NODEs using CONTROL FLOWS.</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Verify structural correctness</td>
<td>After the EPC is finished, the process modeler should inspect the model for any SYNTAX ERROR(s).</td>
</tr>
<tr>
<td></td>
<td>Validate EPC</td>
<td>At this stage, APPROVAL from the stakeholders of the process should be acquired to validate the EPC (Scheer, Thomas, &amp; Adam, 2005). The validation process itself differs between organisations and processes, and is outside the scope of this document. If both the EPC is structurally verified, and the stakeholders have approved the model, it is ready. Otherwise, the process modeler should go back to one or more of the previously described sub-activities to fix the model.</td>
</tr>
</tbody>
</table>
Table 2. Concept table for the PDD of Process Modeling using EPC

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS DESCRIPTION</td>
<td>A PROCESS DESCRIPTION is a comprehensive description of the business process to be modeled in natural language (Scheer, Thomas, &amp; Adam, 2005). The specific form is not relevant in this context, as it may differ from organization to organization.</td>
</tr>
<tr>
<td>EPC</td>
<td>An EPC in this context is defined as the end result of the process modeling procedure (Scheer, Thomas, &amp; Adam, 2005), i.e. a process model that adheres to the notation of the Event-Driven Process Chains. The EPC has a title that is defined in the activity 'Define unique title'.</td>
</tr>
<tr>
<td>EVENT</td>
<td>An EVENT is an EPC NODE that through its name describes the post-conditions for the preceding FUNCTION(s), and the pre-conditions for the succeeding FUNCTION(s) (Mendling, 2008).</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>A FUNCTION is an EPC NODE that captures a task or activity that is performed as part of the business process to be modeled (Mendling, 2008). The name is typically in the form of a verb and an object.</td>
</tr>
<tr>
<td>PROCESS INTERFACE</td>
<td>A PROCESS INTERFACE is an EPC NODE that refers to another EPC that describes a process that is either executed before or after the process of the subject EPC (Mendling, 2008). The attribute 'EPC title' refers to the title of the other EPC.</td>
</tr>
<tr>
<td>LOGICAL CONNECTOR</td>
<td>A LOGICAL CONNECTOR is an EPC NODE that connects three or more FUNCTION(s) and EVENT(s). Each LOGICAL CONNECTOR is either an OR, a XOR, or an AND connector, and can either be a split (one inbound and multiple outbound control flows), or a join (multiple inbound flows and one outbound flow).</td>
</tr>
<tr>
<td>OR</td>
<td>An OR-type LOGICAL CONNECTOR activates or waits for one or more flows from the set of connected control flows (Mendling, 2008).</td>
</tr>
<tr>
<td>XOR</td>
<td>A XOR-type LOGICAL CONNECTOR activates or waits for exactly one flow from the set of connected control flows (Mendling, 2008).</td>
</tr>
<tr>
<td>AND</td>
<td>An AND-type LOGICAL CONNECTOR activates or waits for all flows from the set of connected control flows (Mendling, 2008).</td>
</tr>
<tr>
<td>EPC NODE</td>
<td>An EPC NODE is a generic concept that represents either an EVENT, a FUNCTION, a PROCESS INTERFACE or a LOGICAL CONNECTOR. It is not defined in the main literature sources used to create the diagram, but has been introduced to increase the clarity of the Process-Deliverable Diagram.</td>
</tr>
<tr>
<td>CONTROL FLOW</td>
<td>A CONTROL FLOW is a directed arc in an EPC which connects two EPC NODEs (Mendling, 2008).</td>
</tr>
<tr>
<td>SYNTAX ERROR</td>
<td>A SYNTAX ERROR is defined as a violation of the structural or grammatical rules defined for a language (ISO/IEC/IEEE, 2010), which in this case is the EPC modeling language.</td>
</tr>
<tr>
<td>APPROVAL</td>
<td>The APPROVAL deliverable is obtained from the stakeholders and indicates the EPC is conform their understanding of the process (Scheer, Thomas, &amp; Adam, 2005).</td>
</tr>
</tbody>
</table>
5 References


