External and Internal Events in EPCs: e²EPCs

Oliver Kopp, Matthias Wieland, Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany
{kopp,wieland,leymann}@iaas.uni-stuttgart.de

External and Internal Events in EPCs: e^{2}EPCs

Oliver Kopp, Matthias Wieland, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany
`lastname@iaas.uni-stuttgart.de`

Summary. The notion of event-driven process chains (EPC) is widely used to model processes. It is an ongoing discussion of how to reach executable workflows from EPCs. While the transformation of the general structure and the functions is well-understood, the transformation of events is an open issue. This paper discusses different possible event types and their semantics. Furthermore, it presents a transformation of the introduced event types to workflow constructs respecting the semantics of each event.

1 Introduction

Companies have a strong interest in Business Process Management (BPM) technology to align and support their business processes with IT infrastructure. A business process is a collection of related, structured activities and tasks that produce a specific service or product for customers. Business processes can be executed on an IT infrastructure using workflows. Business processes are expressed using specialized visual process modeling languages such as the Business Process Modeling Notation (BPMN, [1]) or Event-Driven Process Chains (EPC, [2,3]). BPMN and EPCs are mostly concerned with the modeling aspect of business processes, and therefore put an emphasis on being easy to use by providing a standardized set of visualization elements, whereas not defining exact execution semantics. They are designed for the use by non-technical thinking people who want to concentrate on modeling the high-level business process.

In contrast, the Web Service Business Process Execution Language [4] (WS-BPEL or BPEL for short), facilitates business process execution by providing and standardizing execution semantics for orchestrating business activities as workflows. It defines the way in which basic services (business activities in the form of Web Services) are used to build new, coarser grained services. For example, a loan approval workflow orchestrates the basic services “RiskAssessment”, “CreditCheck” and “IncomeReview”. Since Web Services are an implementation of the SOA architectural style, process systems using BPEL as orchestration language are naturally embedded into an existing service oriented architecture implemented by Web Services.

The BPM lifecycle (Phases: Modeling, Execution, Analysis and Optimization) has the aim to continuously improve the process. This is known as business process reengineering. Thus, a process definition is never stable and is permanently adapted. As a consequence, the workflow implementing the changed business process also has to be remodeled all the time. This is the main motivation for automatic transformation of business processes to workflows without ignoring parts of the business process that have to be inserted again in the workflow. In this paper we show how EPC processes can be transformed to workflows not only based on the functions but also transforming the events of the EPC. In the following, we assume that the modeled processes are intended to serve as basis for an automatic execution by a workflow engine.

Event-driven Process Chains (EPCs, [2, 3]) are an event-centric business process modeling language that treats events as "first class citizens", i.e. the occurrence of events are fundamental elements of the business process. EPCs are part of the ARIS framework [5], a "holistic modeling approach" to design and document architectures of integrated information systems from a business' perspective. In ARIS, EPCs are used in the "control view" to describe business processes, allowing for integration and reuse of elements from other views of a model. EPCs consist of four main elements: (i) events (depicted as hexagons), (ii) functions (depicted as rounded boxes), (iii) connectors (depicted as circles) and (iv) control flow arcs. Events in EPCs are *passive*, i.e. they represent a state change in the system, but do not cause it (e.g. they do not provide decisions, but represent decisions taken). Events trigger functions, which are *active* elements that represent the actual work and again raise events upon completion. Connectors are used to *join and split* control flow, represented by arcs in the EPC graph. An EPC starts and ends with one or more events, process control flow itself strictly follows an *alternating sequence of functions and events*, possibly with connectors specifying the kind of control flow join and split in between. The extended event driven process chain (eEPC) extends the EPC by associations to functions. For example, a function may be associated with the organizational unit performing the function or the data needed and produced by the function. Common accepted associations may be found in the EPC Markup Language (EPML, [6]).

eEPCs are in strong contrast to other established process languages such as the Web Services Business Process Execution Language (WS-BPEL or BPEL for short, [4]) or the Business Process Modeling Notation (BPMN, [1]). BPEL and BPMN are rather *service centric* and do not enforce the usage of events as an integral part already at the modeling level. BPMN distinguishes a wide range of events including timer and message events. Mapping of events to BPEL is not an issue here, since there exists a corresponding BPEL construct for each event. BPMN exists in parallel to EPCs. Since EPCs currently do not offer an explicit distinction between internal and external events, we use eEPCs as basis for integration of process logic with the environment.

In current eEPC models, only functions may be annotated with additional information. In the case of events, the semantics is given by their label only. The number of events is three times the number of functions in the SAP reference model containing about 10.000 models [7]. Thus, events are an important information container. While [2] states that "events may reference information objects of the data model", this possibility is not used in products and not regarded in research.

Figure 1 presents an example eEPC. It models an excerpt of the business process “order processing” and is taken from [3]. The excerpt shows the function “Manufacture Item” and its context: after the supplier processed the order (event “(Supplier) Order Processed”) the manufacturing plan is completed (event “Manufacturing Plan Completed”), the function “Manufacture Item” can start. The conjunction of the two events is modeled by the and connector. The function itself is executed at the shop floor (association with the organizational unit “Shop Floor”), which produces an item. The IT relevant data of the item is represented in the data object “Item”. After “Manufacture Item” completes, the event “Item Completed” occurs.

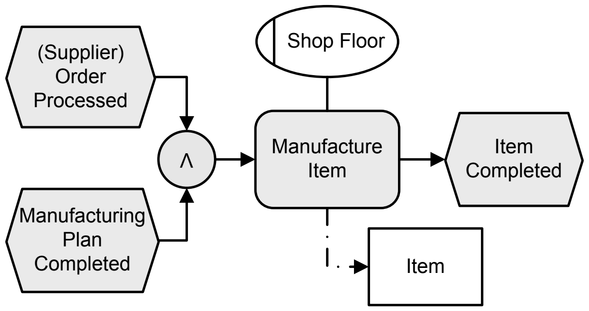

Fig. 1. Example scenario, taken from [3]

Events in eEPCs may be internal or external. *Internal* denotes that the event occurs as a direct result of a function. *External* denotes that the event occurs because of a state change in the environment. The EPC metamodel does not foresee explicit distinctions between internal and external events. Therefore it is not stated whether the event “Item Completed” is an internal or an external event. In case the function “Manufacture Item” denotes that a new manufacture request is sent to the shop floor without waiting for completion, the “Item Completed” events gets an external event, since the shop floor has to notice the process of completion. The other possibility is that the “Manufacture Item” function models the manufacturing of the item and finishes as soon as the item is finished. In this case, the event “Item Completed” is an internal event and the data produced by the function can be used to decide whether this event occurs.

In workflows, internal events are transition conditions between activities and external events are notifications by a message. Current transformation approaches either ignore events, treat them all as external events or treat them all as internal events. In this paper, we propose a modeling extension for eEPCs to allow the business modeler to distinguish internal and external events. This distinction allows generating a fine-grained BPEL workflow model out of the input eEPC. In addition, we use the additional information to generate a participant topology capturing the relation between the process and its environment. This artifact can then be used to wire existing services with the generated process.

Consequently, this paper is organized as follows: The concept and metamodel of our extension to extended event-driven process chains, e^{2}EPCs, is presented in Sec. 2. Section 3 shows how the introduced distinction between internal and external events in e^{2}EPCs can be transformed to BPEL and a participant topology which forms a choreography description. Section 4 provides an overview on current work on transformation of EPCs to BPEL. Finally, Sec. 5 concludes and provides an outlook on future work.

2 Concept and Metamodel of e^{2}EPCs

The scenario in Figure 1 contains events internal and external to the process. Without semantical analysis, it is not possible to distinguish them because the intended usage of events has to be guessed out of the used IT systems. As future work, it would be interesting to classify the events based on the analysis of audit and monitoring logs. Therefore, we propose to extend the eEPC metamodel by adding associations between events and outputs of functions or organizational units to enable the explicit modeling of internal and external events.

This results in a new version of the scenario as shown in Fig. 2. In this figure, the new associations are marked and can be used to distinguish between internal and external events. The two start events (Order processed, Manufacturing plan completed) are connected to organizational units. This means, they receive messages from these systems. Start events always are external events and have to be connected to an organizational unit. In contrast, the "Item completed" event is an internal event recognized by the association to the output "Item" of the function "Manufacture Item". Thus, the event can be evaluated based on that data only and does not need further information. In summary, an event is an internal event if it is associated with output data. An event is an external event, if it is associated with an organizational unit. It is not possible to associate an event with both an organizational unit and output data. e^{2}EPCs allow an event to be unassociated with any organizational unit or output data. In this case, the event cannot be transformed to BPEL, since it is not clear whether it is an internal or external event. Other possibilities to model internal and external events include the usage of swim lanes. The drawback of that approach is that the layout of existing EPCs has to be changed, since for each organizational unit and data item, a separate lane has to be introduced.

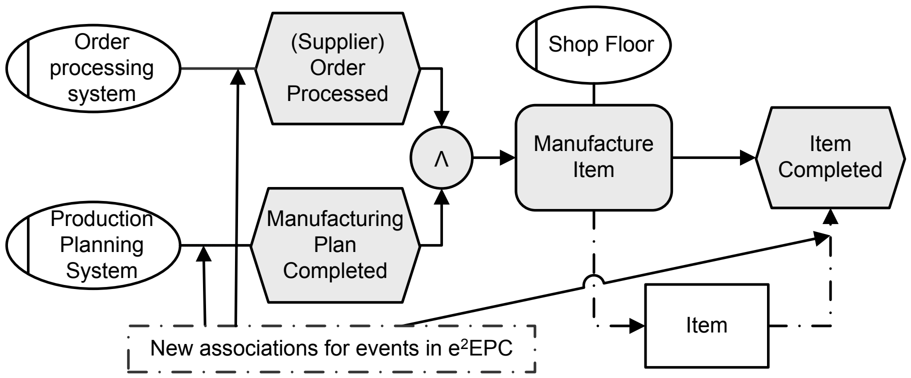
Order processing system
(Supplier) Order Processed
Shop Floor
Manufacture Item
Item Completed
Production Planning System
Manufacturing Plan Completed
Item
New associations for events in e²EPC

Fig. 2. Example Scenario modeled using e^{2}EPC

In Fig. 3 all types of associations added to the eEPC metamodel [3] are shown. They are used to distinguish the two different event types. In e^{2}EPCs, the symbol for an organizational unit is used as superclass for any kind of executor such as an IT system (computer hardware, machine, application software as listed in [3]), a Web Service and a human user. The different types of organizational units shown in Fig. 3 are used to illustrate the different possible types of organizational units used in the transformation. For the modeler, these organizational units do not visually differ. The different types have to be stored in the repository of the used modeling tool and the type information of each organizational unit has to be handed over to the transformation. In that way, the business user has not to be aware of the different types, but the IT expert responsible for the addition of existing IT service operations as organizational units to the repository.

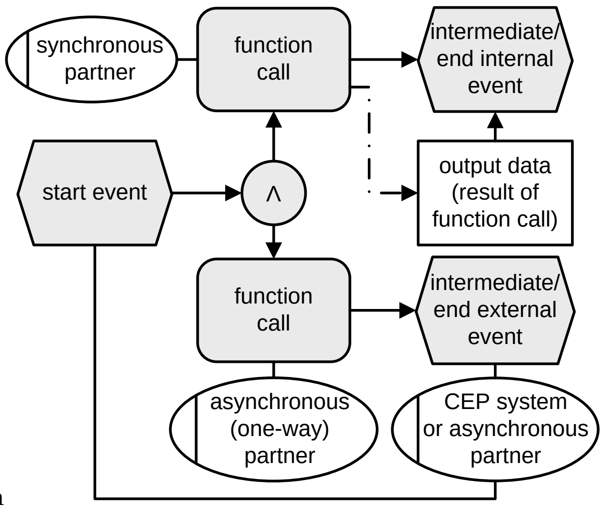
synchronous partner
function call
intermediate/ end internal event
output data (result of function call)
start event
∧
function call
intermediate/ end external event
asynchronous (one-way) partner
CEP system or asynchronous partner

Fig. 3. Elements of e^{2}EPCs

Start events are at the beginning of a process and have no incoming arcs. These events are always triggered by messages and start the process execution. An association between a start event and the organizational unit producing the message is the only possible association allowed in the meta model.

Intermediate events have an incoming and an outgoing arc. An *external intermediate event* is triggered by a message from an organizational unit which makes it similar to a start event. No additional information, such as output data, may be needed to check whether the event happens. Otherwise, the event is not an external intermediate event anymore. An *internal intermediate event* is always connected with an output of a preceding function. The data contained in the output has to be sufficient to determine whether the event happens. If more information was needed, the event would have to receive a message or would have to use an information system for evaluation. In this case, it is not an intermediate event anymore and possibly an external intermediate event or even a function with a subsequent event.

End events are at the end of a process and have no outgoing arcs. The distinction between internal end events and external end events is the same as in the case of intermediate events.

3 Transformation of e^{2}EPCs to BPEL

To execute an EPC model on a workflow machine, there are two general ways: (i) give the EPC an execution semantics or (ii) define a mapping to a workflow language with an execution semantics. In this paper, we focus on the second option, since BPEL workflow engines are widely available, whereas EPC workflow engines are not. By mapping the EPC to a workflow language with a defined execution semantics, the EPC is also given an execution semantics: the semantics of the workflow language used. The Web Services Business Process Execution Language (BPEL, [4]) is the current de-facto standard for workflow execution. Thus, the current approaches map EPCs to BPEL workflows.

The main reason for introducing e^{2}EPCs is to allow a higher value transformation to executable workflows. This means, more information of the process specification is used in the transformation and the resulting workflow model is more detailed in comparison to other transformation approaches. The SAP reference model shows that EPCs contain in average 3 times more events than functions. By adding new associations to the events we enable the inclusion of them in the generated workflows. Without that association the events usually are simply ignored. In the following, we present a transformation which makes use of the events and transforms them to elements in the generated abstract workflow.

A BPEL workflow does not need to be executable by itself. The BPEL specification offers modeling abstract workflows, which may hide operational details. So called opaque activities can be used to model left-out behavior. Abstract workflows may be refined by IT experts to executable workflows enabling the execution on a workflow engine. It is widely acknowledged that a transformation cannot generate an executable workflow, since necessary execution details, such as the concrete message formats and format transformation is missing.

Figure 4 provides an overview on the transformation. The list of participants is essential for the choreography the abstract workflow is embedded in. The participants can be derived from the associations to the functions, start events, intermediate external events and end external events. Each organizational unit becomes a participant in the choreography. A choreography captures the interplay between different workflows [8].

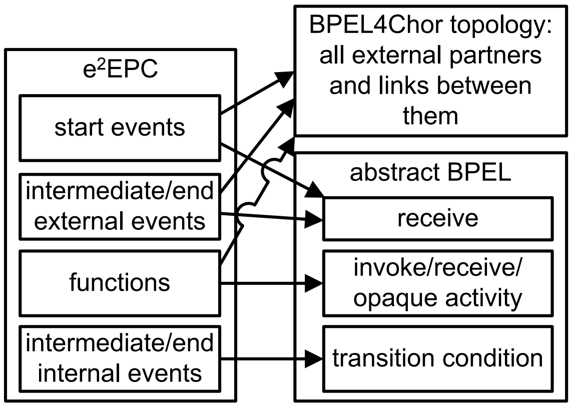
e2EPC
start events
intermediate/end external events
functions
intermediate/end internal events
BPEL4Chor topology: all external partners and links between them
abstract BPEL
receive
invoke/receive/ opaque activity
transition condition

Fig. 4. Transformation

The BPEL workflow itself is transformed out of the EPC process. Each output data element is transformed to a variable declaration in the process. Then, the structure of the process is determined as described in [9], which applies the techniques presented in [10, 11] to EPCs. In general, the graph-structure is preserved and thus this transformation follows the Element-Minimization strategy presented in [12] with the addition that `pick` and `while` structures are transformed to the respective structure in BPEL. A BPEL workflow defines an orchestration of Web services and consists of structured and basic activities. The actual business functions are not implemented by BPEL itself, but by Web services, where the business data is sent to and received from using messages (events are represented as messages, too). Hence, the most important basic activities are `invoke` and `receive`. An `invoke` activity is used to send a message to a Web service. A `receive` activity is used to receive a message. The structured activity `pick` realizes an one-out-of-n choice of messages to receive: the first arrived message wins and the other messages are ignored at that activity. Control flow itself is either modeled block-structured using `if` and `sequence` activities or using graph-based constructs realized by the `flow` activity. In a `flow` activity, activities are connected using links. The issue of non-local join semantics is solved by applying Dead-path Elimination (DPE) which in turn uses negative control tokens. DPE itself is formally defined in [13], specified for BPEL in [4] and explained in detail in [14].

An EPC function is mapped to an `invoke`, `receive` or and `opaque` activity based on the associated organizational unit. In case the organizational unit models an IT service operation, which is already specified, the interaction is known. In the case of current common IT service operations, the interaction patterns are (from the view of the service) in, in/out and out. The view of the business process is dual, therefore in and in/out are transformed to `invoke` and out to a `receive`. In case the organizational unit does not model an IT service operation, an `opaque` activity is generated. If the association from the external event to the organizational unit was directed, the interaction pattern could be derived. We did not introduce directed associations in e^{2}EPCs, since it is unlikely that a business user is aware of the interaction paradigm of a special IT system.

The work of [15] shows that start events in EPCs can be interpreted as message events and also as condition filters. To instantiate a process, BPEL supports message events only. Due to the design of BPEL, we will also treat EPC start events as message events. Similar to [9, 16], start events joined by a XOR connector are transformed to a `pick` activity. Start events joined by an AND connector are transformed to `receive` activities. End events are treated as intermediate events targeting a special function. This special function is transformed to an `empty` activity used as target for the link.

Intermediate external events are transformed dependent on the preceding connector. In the case of a XOR connector, the each external event is transformed to a `receive` activity. In the case of an AND connector, the external event is transformed to a branch of a `pick` activity. OR predecessors are not supported. This part of the transformation is described in detail in [9].

Intermediate internal events are transformed to transitions conditions on a control link. The link connects the transformation of the element preceding the event to the transformation of the element succeeding the event. The label of

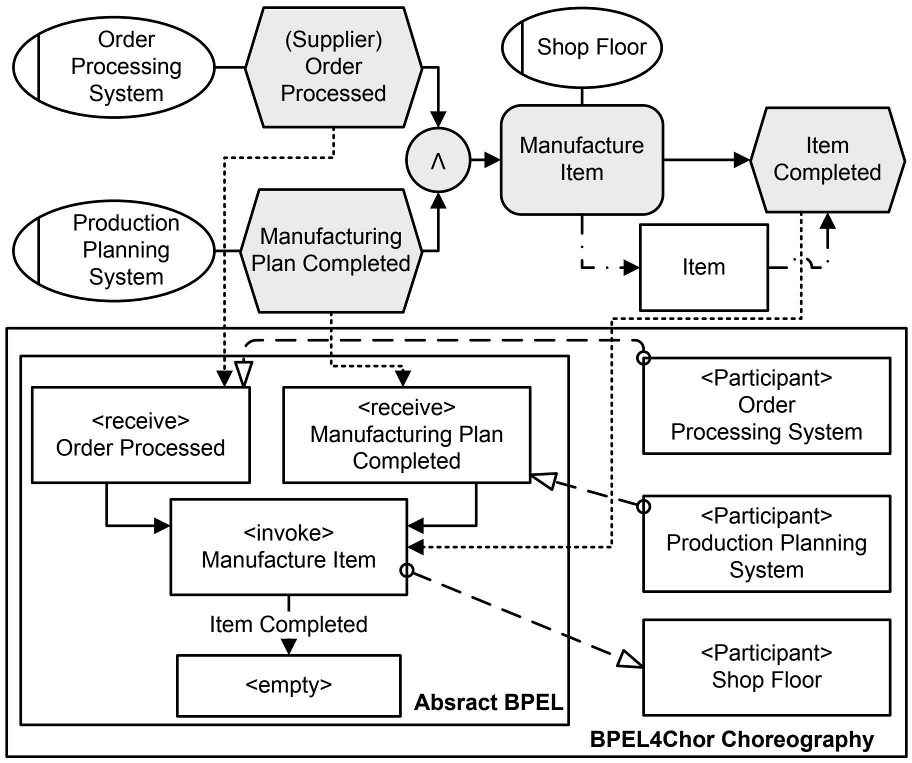
Order Processing System
(Supplier) Order Processed
Shop Floor
Manufacture Item
Item Completed
Production Planning System
Manufacturing Plan Completed
Item
<receive> Order Processed
<receive> Manufacturing Plan Completed
<Participant> Order Processing System
<invoke> Manufacture Item
<Participant> Production Planning System
Item Completed
<empty>
<Participant> Shop Floor
Absract BPEL
BPEL4Chor Choreography

Fig. 5. Transformation and the result as BPEL4Chor choreography

the event is put as condition on the control link. This part of the transformation follows the algorithm described in [17].

Currently, BPEL4Chor is the only language based on BPEL which is capable to capture the links between multiple process models [18]. BPEL4Chor's participant topology lists the participants of the choreography and the message links between them. For each generated communication activity, a message link in the BPEL4Chor topology is generated. For example, the message link for the "(Supplier) Order Processed" event is as follows: `<messageLink sender="OrderProcessingSystem" receiver="OrderProcess" receiveActivity="OrderProcessed"/>`. Note that the activity gets the camel case version of the label of the respective EPC function or event as name.

After the abstract BPEL workflow and the topology information has been generated, the abstract workflow has to be manually refined to an executable workflow model which can be enacted by a workflow engine. For wiring the generated workflow with the other participants in the choreography, a BPEL4Chor participant grounding has to be defined, where each message link is assigned to a Web Service operation. Using that information, the workflow can be deployed. If the other participants do not exist, their behavior can be derived by generating a view on the generated abstract BPEL workflow containing only the interaction with the missing participant as outlined in [19]. Note that a participant in a choreography does not necessarily need to be implemented as a BPEL workflow. It may also be implemented as plain Web Service, since the participant behavior description in a choreography only specifies the public visible behavior and not the actual implementation.

Figure 5 presents the transformation idea and a graphical representation of the transformation result. An implementation is not available, but is possible by extending the ProM tool or by extending other EPC to BPEL transformations. There exists a formal syntax of EPCs [20] and BPEL [21]. Thus, a formal transformation can be defined but is not part of this paper.

Since the BPM lifecycle has the aim to continuously improve the process model, a process definition is never stable and is permanently adapted. Thus, the EPC process has to be changed all the time and consequently the BPEL workflow will change accordingly. The abstract BPEL workflow $BPEL_g$ is manually refined to an executable workflow $BPEL_e$. In order to keep the added technical details, the BPEL workflow cannot simply be regenerated, but rather needs to be updated in a smart way. Therefore we take the original generated model $BPEL_g$ and the model generated within the second lifecycle round $BPEL'_g$ to compute the difference $\Delta(BPEL_g, BPEL'_g)$. Now, it is possible to apply this difference Δ to the original executable workflow $BPEL_e$ in order to get a starting point for the executable workflow $BPEL'_e$ that contains both, the new semantics of the process model and the refinements made in the previous lifecycle round. It may be possible that not all differences can be applied to the new model in case the model has significantly changed. Nevertheless, the derived executable workflow $BPEL'_e$ contains more information than the generated abstract workflow $BPEL'_g$. A detailed discussion of advantages and drawbacks in the case of applying differences to models is presented in [22].

Events concerning the lifecycle of events are out of scope of the paper. These kinds of events are neither treated in the EPC process nor the BPEL workflow itself, but by the workflow engine.

4 Related Work

This section provides an overview on current approaches to transform EPCs to BPEL and to choreographies. A general overview of all available transformations from EPCs to BPEL is provided in [23, 24].

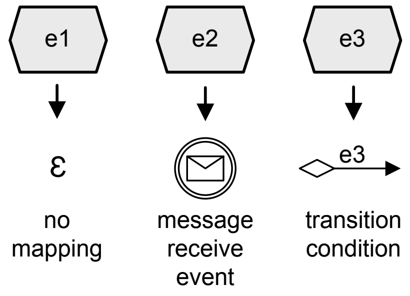

Fig. 6. Different possibilities to transform events into a workflow

Figure 6 summarizes the different possibilities to transform events into a workflow: (i) An event can be ignored. (ii) An event can be transformed to a message receipt. (iii) Finally, an event can be transformed to a transition condition. The transformation approach presented in this paper distinguishes between internal and external events and transforms start events, intermediate events as well as end events. Current related work either does not handle all event types or does not distinguish between external and internal events. Table 1 shows how each related work deals with events. The column "Distinction" shows whether the approach distinguishes between external and internal events. The subsequent three columns show if the approach transforms the respective events. In case the approach transforms the event, the generated BPEL construct is listed.

[17] deals with a variant of EPCs and translates them to a graph-structure. [12] presents different transformation strategies from EPC to BPEL. The transformation strategies are divided into two categories: Preserving the graph-structure or translating as much structures as possible into the corresponding BPEL structures. [25] shows possible annotations to EPCs to enable Web Service specific details in BPEL workflows. [26] identifies workflow patterns [27] in the EPC and translates each pattern to the respective BPEL construct. [16] presents on overview on the transformation using in the ARIS toolset. [9] argues that all events in EPCs should be treated as external events and show how complex event processing can be used to transform EPCs to BPEL. Finally, [28] shows how EPCs can be used to model all details of BPEL processes such as concrete associations to services and variable modifications.

In [29] the authors translate Petri nets into "readable" BPEL code. The algorithm follows the Structure-Maximization strategy presented in [12]. Since Petri-nets are used as input, the translation is not aware of events. Using the event distinction presented in this paper, the translation of [29] may be adopted to use e^{2}EPCs as input language.

Currently, there is no work on transforming EPCs to a choreography definition. Besides choreographies, business landscape [30] also provide an overview on the interplay of services. The work presented in [31] shows how a business landscape can be generated out of EPCs and lists other work generating system overviews out of EPCs.

5 Conclusion and Outlook

This paper presented an extension to eEPCs to enable an unambiguous distinction between internal and external events. We showed how this distinction can be

Reference	Distinction	Start Event	Intermediate Event	End Event
[17] Kopp et al.	–	–	transition condition	–
[12] Mendling et al.	–	`pick/empty`	–	`terminate`
[25] Schmelzle	–	`receive`	transition condition	`reply`
[26] Specht et al.	–	–	transition condition	–
[16] Stein et al.	–	`receive`	–	`invoke`
[9] Wieland et al.	–	`pick/receive`	`pick/receive`	`pick/receive`
[28] Ziemann et al.	+	–	`pick/receive`	–
This paper	+	`pick/receive`	transition condition/ `pick/receive`	transition condition/ `pick/receive`

Table 1. Current EPC-to-BPEL transformation approaches and their treatment of events. – denotes “not supported”, + denotes “supported”

used to improve the mappings from eEPCs to BPEL by generating transition conditions out of an internal event and generating a receiving activity in case of an external event.

Future work is to provide suitable tool support for the presented method. Future area of research includes the evaluation of combined intermediate events. These combined events may be associated with both, an organizational unit and output data. These associations denote that a received message and the evaluation of process internal data is needed to determine whether the event happens.

Acknowledgments This work has been supported by the BMBF funded project Tools4BPEL (01ISE08B) and the DFG project Nexus (SFB627). We thank Wil van der Aalst for the fruitful discussions at BIS 2009.

References

1. Object Management Group: Business Process Modeling Notation, V1.2. (2009)
2. Keller, G., Nüttgens, N., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Technical Report Heft 89, Universität des Saarlandes (1992)
3. Scheer, A.W., Thomas, O., Adam, O.: Process Modeling Using Event-Driven Process Chains. In: Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley & Sons (2005) 119–146
4. OASIS: Web Services Business Process Execution Language Version 2.0. (2007)
5. Scheer, A.W.: ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen. Springer (2003)
6. Mendling, J., Nüttgens, M.: EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC). ISeB **4**(3) (2006)
7. Mendling, J.: Errors in the SAP Reference Model. BPTrends (June 2006)

8. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology **50**(2) (February 2008) 122–127
9. Wieland, M., Martin, D., Kopp, O., Leymann, F.: SOEDA: A Methodology for Specification and Implementation of Applications on a Service-Oriented Event-Driven Architecture. In: BIS 2009. (2009)
10. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. In: BPM 2008, Springer (2008)
11. Garca-Bauelos, L.: Pattern Identification and Classification in the Translation from BPMN to BPEL. In: On the Move to Meaningful Internet Systems, Springer (2008)
12. Mendling, J., Lassen, K.B., Zdun, U.: On the Transformation of Control Flow between Block-Oriented and Graph-Oriented ProcessModeling Languages. IJBPIM **3**(2) (October 2008) 96–108
13. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)
14. Curbera, F., et al.: Exception Handling in the BPEL4WS Language. In: BPM 2003, Springer (2003)
15. Decker, G., Mendling, J.: Process Instantiation. Data & Knowledge Engineering **68** (2009) 777–792
16. Stein, S., Ivanov, K.: EPK nach BPEL Transformation als Voraussetzung fr praktische Umsetzung einer SOA. In: Software Engineering 2007, GI (2007)
17. Kopp, O., Unger, T., Leymann, F.: Nautilus Event-driven Process Chains: Syntax, Semantics, and their mapping to BPEL. In: EPK 2006. (2006)
18. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting Services: From Specification to Execution. Data & Knowledge Engineering (April 2009)
19. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: WS-FM 2007. (2007)
20. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. Data Knowl. Eng **56**(1) (2006) 23–40
21. Kopp, O., Mietzner, R., Leymann, F.: Abstract Syntax of WS-BPEL 2.0. Technical report, University of Stuttgart, IAAS, Germany (2008)
22. Kindler, E., Könemann, P., Unland, L.: Difference-based model synchronization in an industrial MDD process. In: MDTPI 2009. (2009)
23. Stein, S., Kühne, S., Ivanov, K.: Business to IT Transformations Revisited. In: MDE4BPM 2008. (2008)
24. Wieland, M., et al.: Events Make Workflows Really Useful. Technical report, University of Stuttgart, IAAS, Germany (2008)
25. Schmelzle, O.: Transformation von annotierten Geschäftsprozessen nach BPEL. Master's thesis, Gottfried Wilhelm Leibniz Universität Hannover (2007)
26. Specht, T., et al.: Modeling cooperative business processes and transformation to a service oriented architecture. In: CEC 2005. (July 2005) 249–256
27. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
28. Ziemann, J., Mendling., J.: EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In: MITIP. (2005)
29. v.d. Aalst, W.M.P., Bisgaard Lassen, K.: Translating unstructured workflow processes to readable bpel: Theory and implementation. InfSof **50**(3) (2008)
30. Keller, F., Wendt, S.: FMC: An approach towards architecture-centric system development. In: ECBS 2003. (2003)
31. Kopp, O., Eberle, H., Leymann, F., Unger, T.: From Process Models to Business Landscapes. In: EPK 2007. (2007)