

QUT Digital Repository:  
<http://eprints.qut.edu.au/>



Mending, Jan and La Rosa, Marcello and ter Hofstede, Arthur H.M. (2008)  
Correctness of Business Process Models with Roles and Objects.

© Copyright 2008 (The authors)

# Correctness of Business Process Models with Roles and Objects

J. Mendling<sup>a,\*</sup> M. La Rosa<sup>a</sup> A.H.M. ter Hofstede<sup>a</sup>

<sup>a</sup>*Queensland University of Technology, PO Box 2434, QLD 4000 Brisbane, Australia*

---

## Abstract

The verification of business process models is an important step in the design phase of process-aware information systems. While a process model often describes different perspectives like control flow, object flow, and role assignment, most of the contributions in the areas of verification consider only the control flow. Hardly any work considers these three perspectives in a combined verification approach. In this paper we address this gap and introduce Integrated EPCs (iEPCs), a business process modeling language that extends EPCs with a concept of object flow and role assignment. By abstracting from the subtle differences of recent EPC semantics definitions, we show how any of these formalizations can be enhanced with transition rules that consider object existence and role availability as part of the state concept. Furthermore, we define three theorems that relate soundness of EPCs to soundness of iEPCs with different initial role and object set. These theorems provide the basis for a systematic verification approach of iEPCs that first identifies control-flow problems, then object-flow problems, and finally suitable role subsets. This way, our work contributes to a better identification of correctness issues already in conceptual process models in the early design phases.

*Key words:* Business Process Modeling, Verification, Soundness, Systems Analysis and Design;

---

---

\* Corresponding author

*Email addresses:* [j.mendling@qut.edu.au](mailto:j.mendling@qut.edu.au) (J. Mendling),  
[m.larosa@qut.edu.au](mailto:m.larosa@qut.edu.au) (M. La Rosa), [a.terhofstede@qut.edu.au](mailto:a.terhofstede@qut.edu.au) (A.H.M. ter Hofstede).

## 1 Introduction

The correctness of models is a major stream of research in process modeling. Its importance stems from the observation that incorrect models can lead to wrong decisions regarding a process and to unsatisfactory implementations of information systems. Already 25 years ago, it was clearly understood in the software engineering discipline that correcting a modeling error in the early design phase is feasible with reasonable effort while a post-implementation correction costs drastically more [1]. In process modeling, the importance of verification techniques has recently been confirmed by studies that reveal a significant error rate of more than 20% (depending on the sample) in process model collections from practice [2,3].

Based on Petri nets analysis techniques and respective tool implementations, it has become feasible to verify different notions of soundness for various modeling languages, e.g. Workflow nets, EPCs, YAWL, or BPMN [4,5,6,7,8]. All these approaches check aspects related to proper completion of a process. To achieve that, they mainly focus on the control-flow. Although there are correctness issues that relate to other perspectives of a process model, e.g. data flow, these problems are hardly considered in current research. The chapter on properties of business processes in [9] highlights this imbalance. Since most process modeling in the early design phase is done with languages like Event-driven Process Chains (EPCs) or Business Process Modeling Notation (BPMN), we need a better understanding of correctness issues in these conceptual languages that goes beyond the control-flow.

This paper addresses this gap by extending the existing EPC language with formal concepts of object flow and role assignment. This extension is called Integrated EPC (or for short iEPC). The motivation for our formalization builds on requirements that we identified for reference process models in practice. In particular, we found that the optional and non-deterministic usage of objects as well as role assignment is not appropriately supported by existing tools such as ARIS and Colored Petri nets. Our contribution is twofold. In a first step, we formalize a notion of state that reflects the existence of objects and the availability of resources. Then, we specify the formal semantics of iEPCs as a transition system. This way we can use analysis tools like the reachability graph to detect execution problems such as deadlocks. This sort of analysis is important to resolve inconsistencies already in early-phase conceptual models. Such quality assurance is deemed to reduce correction costs in later phases and to increase the acceptance of high-level models for information system implementation. Our second contribution is a set of three theorems that show how the soundness of an iEPC with different sets of initial objects and roles relates to the soundness of a corresponding EPC without the extensions. Based on this, we describe how one can resolve problems with control-flow, object-flow,

and roles in a step-by-step manner.

Against this background, the rest of the paper is structured as follows. Section 2 provides an illustrative introduction to iEPCs using an extract of a reference process model from the film industry. In this section we also discuss correctness issues with the object and resource perspective of the process and we informally sketch our approach. In Section 3 we define the formalism to analyze the problem. Building on a definition of syntactical correctness for iEPCs, we specify a notion of state along with a definition of initial and final marking. Then, we formalize a transition system such that we can calculate the reachability graph for deadlock analysis. Beyond that, we present theorems on the relationship between an iEPC and its corresponding EPC. Section 4 discusses related work and Section 5 concludes the paper.

## 2 Integrated Event-driven Process Chains

In this section we introduce Integrated Event-driven Process Chains (iEPCs). iEPCs basically extend EPCs with an object and a role perspective. In Section 2.1 we informally describe iEPCs with the help of a process model from the film industry. In Section 2.2 we identify a set of problems that might prevent the process from being executed properly.

### 2.1 *The Dialog Editing Process*

We illustrate the concepts of iEPCs by means of a sample process model shown in Fig. 1. This model is an extract of a reference process model for screen post-production which focuses on dialog editing—the stage following the shooting of a movie in which the dialogs from set are integrated with additional dialogs recorded in studio. This reference process model has been constructed and validated with domain experts of the Australian Film Television & Radio School.

EPCs main elements are events, functions, control-flow connectors, and arcs linking these elements. Events model triggers or conditions, functions correspond to tasks and connectors denote splits and joins of type AND, OR or XOR. The iEPC notation extends these concepts by associating roles and objects with functions. A role, depicted on a function’s left hand side, captures a class of organizational resources that is able to perform that function: e.g. the role Dialog Editor captures the set of all the persons with this role in a given screen project. A role is dynamically bound to one concrete resource at runtime (e.g. for a specific project, the Dialog Editor associated to function

*Sync dialogs* will be bound to Michelle Portland). A resource can be human or non-human (e.g. an information system), but for simplicity, we only consider human resources in the example. An object, depicted on a function's right hand side, captures a physical or software artifact of an enterprise, that is used (input object) or produced (output object) by a function. Each object in the process model is statically bound to a concrete artifact.

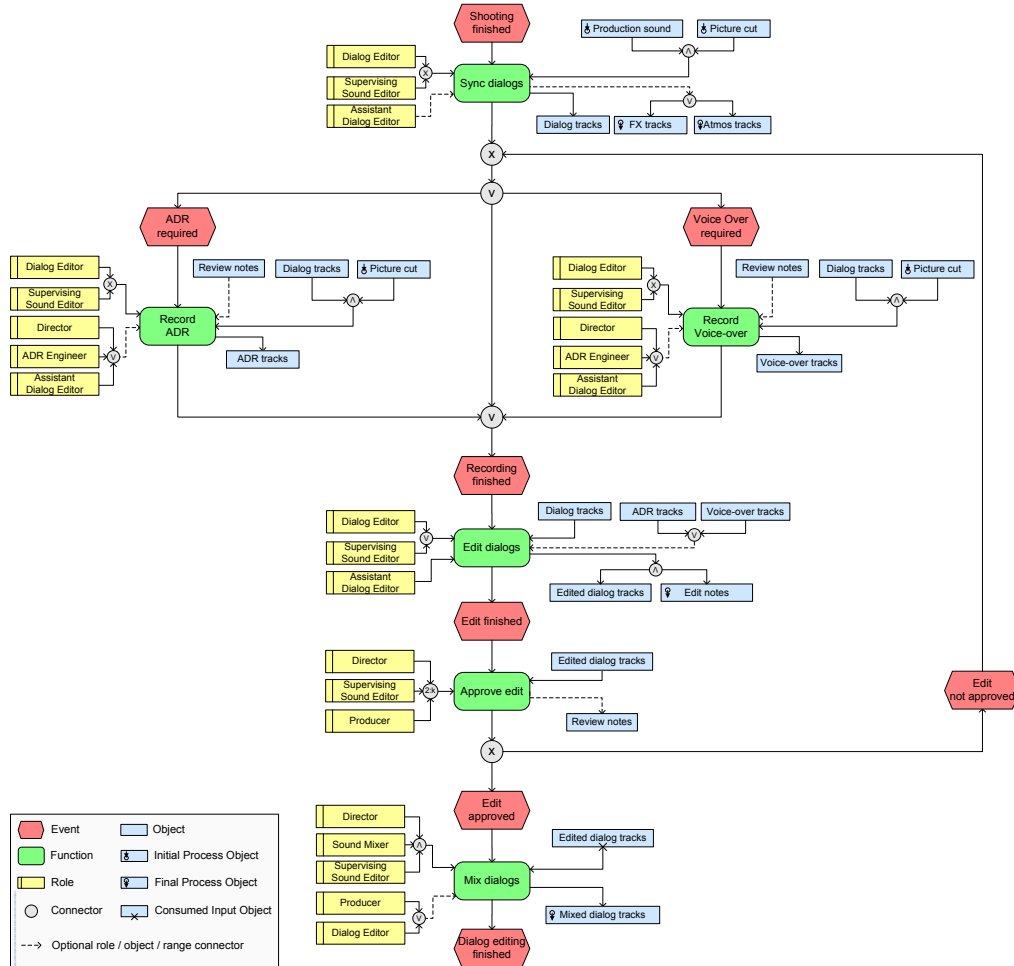


Figure 1. The Dialog editing process model in iEPC.

The first function of the example is *Sync dialogs*, starting when the shooting has completed. Roles and objects are linked to functions either directly or via a connector: the XOR-join between Dialog Editor and Supervising Sound Editor indicates that exactly one of these roles is required to perform this activity, although both are capable of doing it. The Dialog Editor or the Supervising Sound Editor may require the help of an Assistant Dialog Editor. This role is connected to the function via a dashed arc, which indicates that a role, object, or a combination thereof is optional, whereas a solid arc indicates mandatoriness. To ensure that at runtime every function has a resource for its execution, functions have to be connected to a mandatory role or a mandatory combination of roles in the iEPC.

The synchronization of the dialogs consists of extrapolating the Dialog tracks from the Production sound (the sound recorded on set) based on a Picture cut (a preview of the movie). Thus, Production sound and Picture cut are both input objects connected via an AND-join, while Dialog tracks is a mandatory output object. Sometimes, the Production sound also contains effects (FX) and/or atmospheres (atmos) recorded on set, which can also be extrapolated at this stage. FX tracks and Atmos tracks are thus optional outputs of *Synch dialogs*, linked by an optional OR-split. Connecting two or more roles or objects via an optional OR, is a shortcut to directly connecting each role or object to the function via a dashed arc.

In an iEPC, an *initial process input* is an object which is input to the whole process, i.e. an object for which there is no function in the process model that produces it. Similarly, a *final process output* is an object which is output to the whole process, i.e. an object for which there is no function in the process model that uses it. Production sound and Picture cut are initial inputs, while FX tracks and Atmos tracks are final outputs as they are only used by other process models in post-production.

Once the dialogs have been synchronized, automated dialog replacements (ADR) may be recorded in the studio through the function *Record ADR*, to replace those production dialogs of poor quality. Similarly, a Voice-over may also be recorded through the function *Record Voice-over*, if it is needed in the movie. These functions are carried out by the same roles, require the same inputs, and produce ADR tracks and Voice-over tracks as output.

In *Edit Dialog*, the Dialog Editor and/or the Supervising Sound Editor work with the Assistant Dialog Editor to clean-up and integrate Dialog tracks with ADR tracks and/or Voice-over tracks, depending on which tracks have been recorded in the studio. This function results in Edited dialog tracks as well as Edit notes reporting on the choices made (a final process output).

The Edited dialog tracks need to be approved in *Dialog edit approval* by at least two roles among Director, Supervising Sound Editor and Producer, that have creative authority in the project. These roles are linked to a *range* connector. Generally speaking, a range connector indicates the upper and lower bound for the number of elements (roles or objects) required for the function that it is connected to (where  $k$  refers to the indegree for a join or to the outdegree for a split; in the case of the range connector connected to Approve edit  $k = 3$ ). If the edit is not approved, the result of this task is a set of Review notes describing the changes required. In this case, *Record ADR* and *Record Voice-over* must be repeated by using the Review notes as input to guide the required amendments.

If the edit is approved (either directly or after a re-recording session), the

process ends with function *Mix dialogs*, in which the Edited dialog tracks are mixed by the Director, Sound Mixer and Supervising Sound Editor and delivered to the next stage. Producer and Dialog Editor may help in this activity. The cross below Edited dialog tracks indicates that the object is consumed by the function and is no longer available afterwards. In fact, the mixing is directly performed on the edited tracks, by fine-tuning each track's volume.

Although the above example is only an extract of a broader reference model, it shows how interdependencies among roles, objects and control-flow elements can be quite intricate and thus need proper consideration.

## 2.2 Potential Problems with the *iEPC* Model

We now discuss the process model from a correctness point of view. The control-flow of this model is quite simple and it is easy to verify that the process is sound, i.e. there are neither deadlocks nor livelocks and proper completion of the process is guaranteed. Indeed, the model is also structured: there is one OR-block which is properly nested within an XOR-loop. Beyond the control-flow, there can be further issues related to resources and objects. Such issues may occur when a required input object does not exist or a required resource is not available at the time of executing a given function. In the following, we will discuss some causes for these issues.

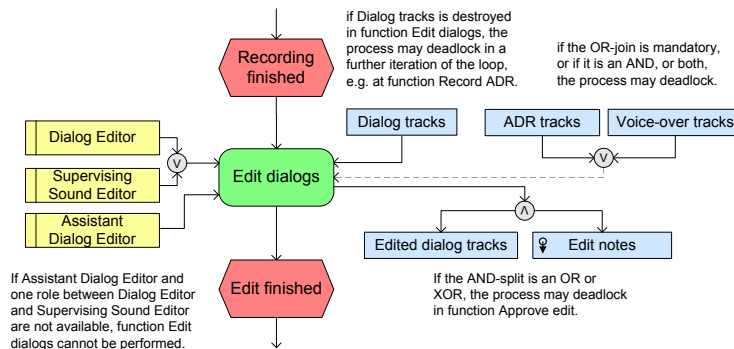


Figure 2. Potential issues related to function *Edit dialogs*.

Fig. 2 shows the *Edit dialogs* function and some reasons why this function might not be properly executed if the process model is varied. Firstly, this function is on a loop and it requires Dialogs tracks as input object. If this object was destroyed by this function, it would no more be available for the second iteration of the loop. Secondly, the OR-join for ADR tracks and Voice-over tracks is optional. Since the functions that create these objects (*Record ADR* and *Record Voice-over*) are optional, the objects might not be produced in some instances of these functions. In this case, there would be an issue

if these objects were mandatory for *Edit dialogs*. Thirdly, if we replaced the AND-split between the output objects Edited dialog tracks and Edit notes with an OR-split or XOR-split, we would create non-deterministic behavior. This may block the execution of function *Approve edit*, which always requires Edited dialog tracks. Finally, if the Assistant Dialog Editor is not available, the function could not be executed due to the lack of a mandatory resource, i.e. a resource with special expertise.

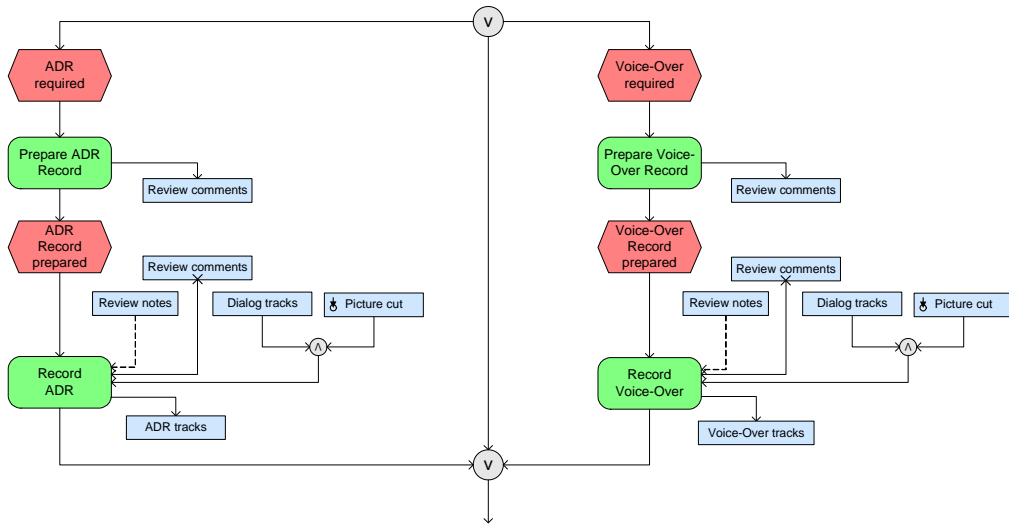


Figure 3. Potential Problems with concurrent object deletion.

The potential problem with the OR-join (second case) already points to the fact that the resource and object perspectives interact with the control-flow one. Fig. 3 further highlights this interaction. It shows a variation process fragment enclosed between the two OR connectors, in which functions *Record ADR* and *Record Voice-over* are performed in parallel. There is now an additional preparation step for each of the functions. This step is needed to create, for each of the two branches, an object Review Comments that is used by the subsequent recording function. It turns out that the proper execution of the parallel branches depends upon the order in which the four functions are processed. Consider the case in which *Prepare ADR Record* is executed before the *Prepare Voice-over Record*. The first function would create the Review Comments object. The second function would then replace it with a new object. The *Record ADR* function would take the Review Comments as an input and destroy them during execution. As a consequence, the *Record Voice-over* function would miss one of its mandatory input objects.

In the following section we will formalize this problem and capture control-flow, object existence, and resource availability in an extended notion of state.

### 3 Formal Semantics and Verification of iEPCs

In this section we formalize iEPCs and their behavior. In Section 3.1 we specify the syntax of iEPCs. This definition provides the basis for formalizing the iEPC semantics in Section 3.2. For the semantics we define a mechanism to extend the transition rules of existing EPC formalizations, e.g. [10,8] and add rules to handle objects and roles assigned to functions. Furthermore, we discuss soundness of iEPCs and some interesting properties related to it.

#### 3.1 iEPC Syntax

The following definition formalizes the notion of iEPC, which adds to EPCs a precise representation of roles and objects participating in a process. In an iEPC each node represents an instance of a function, role or object. The range connector is modeled by a pair of natural numbers: lower bound ( $n$ ) and upper bound ( $m$ ). Indeed, an AND, OR and XOR correspond to a range connector resp. with  $n = m = k$ , with  $n = 1, m = k$  and with  $n = m = 1$ . So we do not need to model the logic operators with separate connectors for roles and objects, although they can be graphically represented with the traditional EPC notation, as in Fig. 1. For the sake of keeping the model consistent with previous formalizations of EPCs, the range connector is not allowed in the control-flow. Minimal effort would however be required to add this construct. The optionality of roles, objects and range connectors, shown in the process as a property of the arc that links the node with the function, is modeled in iEPC as an attribute of the nodes. The consumption of input objects is modeled in the same way.

**Definition 1 (iEPC)** *Let  $F$  be a finite set of functions,  $R$  a finite set of roles and  $O$  a finite set of objects. An integrated EPC over  $F, R, O$  is a tuple  $iEPC_{F,R,O} = (E, F_N, R_N, O_N, nm, C, A, L)$ , where:*

- $E$  is a finite, non-empty set of events;
- $F_N$  is a finite, non-empty set of function nodes for the process;
- $R_N$  is a finite, non-empty set of role nodes for the process;
- $O_N$  is a finite set of object nodes for the process;
- $nm = nf \cup nr \cup no$ , where:
  - $nf \in F_N \rightarrow F$  assigns each function node to a function;
  - $nr \in R_N \rightarrow R$  assigns each role node to a role;
  - $no \in O_N \rightarrow O$  assigns each object node to an object;

Furthermore, for a set  $X$  we define  $nm(X) = \{nm(x) \mid x \in X\}$ .

- $C = C_{CF} \cup C_R \cup C_{IN} \cup C_{OUT}$  is a finite set of logical connectors, where:
  - $C_{CF}$  is the set of control-flow connectors,
  - $C_R$  is the set of range connectors for role nodes (role connectors),
  - $C_{IN}$  is the set of range connectors for input nodes (input connectors),
  - $C_{OUT}$  is the set of range connectors for output nodes (output connectors),  
where  $C_{CF}, C_R, C_{IN}$  and  $C_{OUT}$  are mutually disjoint;
- $A = A_{CF} \cup A_R \cup A_{IN} \cup A_{OUT}$  is a set of arcs, where:
  - $A_{CF} \subseteq (E \times F_N) \cup (F_N \times E) \cup (E \times C_{CF}) \cup (C_{CF} \times E) \cup (F_N \times C_{CF}) \cup (C_{CF} \times F_N) \cup (C_{CF} \times C_{CF})$  is the set of control-flow arcs,
  - $A_R \subseteq (R_N \times F_N) \cup (R_N \times C_R) \cup (C_R \times F_N)$  is the set of role arcs,
  - $A_{IN} \subseteq (O_N \times F_N) \cup (O_N \times C_{IN}) \cup (C_{IN} \times F_N)$  is the set of input arcs,
  - $A_{OUT} \subseteq (F_N \times O_N) \cup (F_N \times C_{OUT}) \cup (C_{OUT} \times O_N)$  is the set of output arcs,  
where  $A_R, A_{IN}$  and  $A_{OUT}$  are intransitive relations;
- $L = l_C^T \cup l_C^N \cup l_C^M \cup l_R^M \cup l_O^M \cup l_O^U$  is a set of label assignments, where:
  - $l_C^T \in C_{CF} \rightarrow \{AND, OR, XOR\}$  specifies the type of control-flow connector,
  - $l_C^N \in (C_R \cup C_{IN} \cup C_{OUT}) \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{k\}) \cup \{(k, k)\}$ , specifies lower bound and upper bound of the range connector,
  - $l_C^M \in (C_R \cup C_{IN} \cup C_{OUT}) \rightarrow \{MND, OPT\}$  specifies if a role connector, an input connector or an output connector is mandatory or optional,
  - $l_R^M \in R_N \rightarrow \{MND, OPT\}$  specifies if a role node is mandatory or optional;
  - $l_O^M \in O_N \rightarrow \{MND, OPT\}$  specifies if an object node is mandatory or optional;
  - $l_O^U \in O_N^{IN} \rightarrow \{USE, CNS\}$  specifies if an input object node is used or consumed,  
where  $O_N^{IN} = \text{dom}(A_{IN}) \cap O_N$ .

Given a connector  $c$ , let  $l_C^N(c) = (n, m)$  for all  $c \in C \setminus C_{CF}$ . Then we use  $\text{lwb}(c) = n$  and  $\text{upb}(c) = m$  to refer to lower bound and upper bound of  $c$ . If  $F$ ,  $R$  and  $O$  are clear from the context, we drop the subscript from  $iEPC$ . Also, we refer to all the function nodes, role nodes and object nodes simply as functions, roles and objects, wherever this does not lead to confusion.

We introduce the following notation for a more concise characterization of  $iEPC$ s.

**Definition 2 (Auxiliary sets, functions and predicates)** For an  $iEPC$  we define the following subsets of its nodes, functions and predicates:

- $N_{CF} = E \cup F_N \cup C_{CF}$ , as its set of control-flow nodes;
- $N_R = F_N \cup R_N \cup C_R$ , as its set of role nodes;
- $N_{IN} = F_N \cup O_N^{IN} \cup C_{IN}$ , as its set of input nodes;
- $N_{OUT} = F_N \cup O_N^{OUT} \cup C_{OUT}$ , as its set of output nodes, where  $O_N^{OUT} = \text{dom}(A_{OUT}) \cap O_N$ ;
- $N = N_{CF} \cup N_R \cup N_{IN} \cup N_{OUT}$ , as its set of nodes;

- $\forall n \in N_\alpha \quad \overset{\alpha}{\bullet} n = \{x \in N_\alpha \mid (x, n) \in A_\alpha\}$ , as the  $\alpha$ -preset of  $n$ ,  $\alpha \in \{CF, R, IN, OUT\}$ ;
- $\forall n \in N_\alpha \quad n \overset{\alpha}{\bullet} = \{x \in N_\alpha \mid (n, x) \in A_\alpha\}$ , as the  $\alpha$ -postset of  $n$ ,  $\alpha \in \{CF, R, IN, OUT\}$ ;
- $\forall t \in F_N \quad in(t) = \bigcup_{o \in O_N} : (o,t) \in A_{IN}^+ \quad no(o)$  as the set of input objects for function node  $t$ ;
- $\forall t \in F_N \quad out(t) = \bigcup_{o \in O_N} : (t,o) \in A_{OUT}^+ \quad no(o)$  as the set of output objects for function node  $t$ ;
- $O_N^i = \bigcup_{t \in F_N} in(t) \setminus \bigcup_{t \in F_N} out(t)$  as the set of initial process objects;
- $O_N^f = \bigcup_{t \in F_N} out(t) \setminus \bigcup_{t \in F_N} in(t)$  as the set of final process objects;
- $E_s = \{e \in E \mid | \overset{CF}{\bullet} e | = 0 \wedge | e \overset{CF}{\bullet} | = 1\}$  as the set of start events;
- $E_e = \{e \in E \mid | \overset{CF}{\bullet} e | = 1 \wedge | e \overset{CF}{\bullet} | = 0\}$  as the set of end events;
- $A_s = \{(x, y) \in A_{CF} \mid x \in E_s\}$  as the set of start arcs;
- $A_e = \{(x, y) \in A_{CF} \mid y \in E_e\}$  as the set of end arcs;
- $A_{int} = A \setminus (A_s \cup A_e)$  as the set of intermediate arcs;
- $degree(x) = \begin{cases} | \overset{R}{\bullet} x |, & \text{if } x \in C_R, \text{ returns the indegree of a role connector,} \\ | \overset{IN}{\bullet} x |, & \text{if } x \in C_{IN}, \text{ returns the indegree of an input connector,} \\ | x \overset{OUT}{\bullet} |, & \text{if } x \in C_{OUT}, \text{ returns the outdegree of an output connector;} \end{cases}$
- $p = \langle n_1, n_2, \dots, n_k \rangle$  is a control-flow path such that  $(n_i, n_{i+1}) \in A_{CF}$  for  $1 \leq i \leq k-1$ . For short, we indicate that  $p$  is a path from  $n_1$  to  $n_k$  as  $p : n_1 \hookrightarrow n_k$ . Also,  $P(p) = \{n_1, \dots, n_k\}$  indicates the alphabet of  $p$ .

The following definition captures the essential syntactical requirements of an iEPC.

**Definition 3 (Syntactically Correct iEPC)** *An iEPC is syntactically correct if it fulfills the following requirements:*

- (1) *iEPC is a directed graph such that every control-flow node is on a control-flow path from a start to an end event:*  
let  $e_s \in E_s$  and  $e_e \in E_e$ , then  $\forall n \in N_{CF} \quad \exists_{p \in N_{CF}^+, p: e_s \hookrightarrow e_e} [n \in P(p)]$ .
- (2) *There is at least one start event and one end event in iEPC:  $|E_s| > 0$  and  $|E_e| > 0$ .*
- (3) *Events have at most one incoming and one outgoing control-flow arc:*  
 $\forall e \in E \quad [| \overset{CF}{\bullet} e | \leq 1 \wedge | e \overset{CF}{\bullet} | \leq 1]$ .
- (4) *Functions have exactly one incoming and one outgoing control-flow arc:*  
 $\forall f \in F_N \quad [| \overset{CF}{\bullet} f | = | f \overset{CF}{\bullet} | = 1]$ .

- (5) *Control-flow connectors have one incoming and multiple outgoing arcs or vice versa:*  
 $\forall c \in C_{CF} [(|c^{\bullet CF}| = 1 \wedge |c^{\bullet CF}| > 1) \vee (|c^{\bullet CF}| > 1 \wedge |c^{\bullet CF}| = 1)],$  (*split, join*),  
*Role connectors have multiple incoming arcs and exactly one outgoing arc:*  
 $\forall c \in C_R [|\bullet^R c| > 1 \wedge |c^{\bullet R}| = 1],$  (*join*),  
*Input connectors have multiple incoming arcs and exactly one outgoing arc:*  
 $\forall c \in C_{IN} [|\bullet^{IN} c| > 1 \wedge |c^{\bullet IN}| = 1],$  (*join*),  
*Output connectors have exactly one incoming arc and multiple outgoing arcs:*  
 $\forall c \in C_{OUT} [|c^{\bullet OUT}| = 1 \wedge |c^{\bullet OUT}| > 1],$  (*split*).
- (6) *Roles have exactly one outgoing arc:*  
 $\forall r \in R_N |r^{\bullet R}| = 1.$
- (7) *Objects have exactly one outgoing input arc or one incoming output arc:*  
 $\forall o \in O_N [(|o^{\bullet IN}| = 1 \wedge |o^{\bullet OUT}| = 0) \vee (|o^{\bullet IN}| = 0 \wedge |o^{\bullet OUT}| = 1)].$
- (8) *Functions are linked to at least a mandatory role or a mandatory role connector:  $\forall f \in F_N [\exists_{r \in R_f} [l_R^M(r) = MND] \vee \exists_{c \in R_f} [l_C^M(c) = MND]]$ , it follows that  $|\bullet^R f| > 0.$*
- (9) *Roles and objects linked to connectors are mandatory:*<sup>1</sup>  
 $\forall r \in R_N [r \in \text{dom}((R_N \times C_R) \cap A_R) \Rightarrow l_R^M(r) = MND],$   
 $\forall o \in O_N^{IN} [o \in \text{dom}((O_N \times C_{IN}) \cap A_{IN}) \Rightarrow l_O^M(o) = MND],$   
 $\forall o \in O_N^{OUT} [o \in \text{dom}((C_{OUT} \times O_N) \cap A_{OUT}) \Rightarrow l_O^M(o) = MND].$
- (10) *Upper bound and lower bound of range connectors are restricted as follows:*  
 $\forall c \in C_R \cup C_{IN} \cup C_{OUT} [1 \leq \text{lwb}(c) \leq \text{upb}(c) \wedge (\text{lwb}(c) \leq \text{degree}(c) \vee \text{upb}(c) = k)],$   
*where  $n \leq m$  iff  $(n \leq m) \vee (m = k) \vee (n = m = k).$*

The editing process model of Fig. 1 is syntactically correct. However, Def. 3 does not prevent behavioral issues (e.g. deadlocks) that may occur at run-time.

### 3.2 iEPC Semantics

The dynamic behavior of iEPC has to take into account not only the routing rules of the control-flow, but also the availability of the resources and the existence of the objects present in the model. A state of the execution of an iEPC can be identified by a marking of tokens for the control-flow, plus the set of roles indicating the availability of the respective resource, and the set of objects indicating their existence. A function is enabled and can fire if it receives control (i.e. if at least a token marks its control-flow input arc), if at least all resources for its *mandatory* roles are available and if at least all its *mandatory* input objects exist. The state of roles and objects is evaluated directly or via the respective range connectors. During a function's execution, the associated roles become unavailable and once the execution is concluded, the output objects are created (i.e. they come into existence), and those ones

<sup>1</sup> The optionality of a group of roles/objects linked by a range connector is modeled by making the connector optional

that are indicated as *consumed*, are destroyed. Initial process objects, i.e. those ones that are provided from outside the process, exist before the process execution starts. A function does not wait for an optional role to become available. However, if such a role is available before the function is executed, it is treated as a mandatory role.

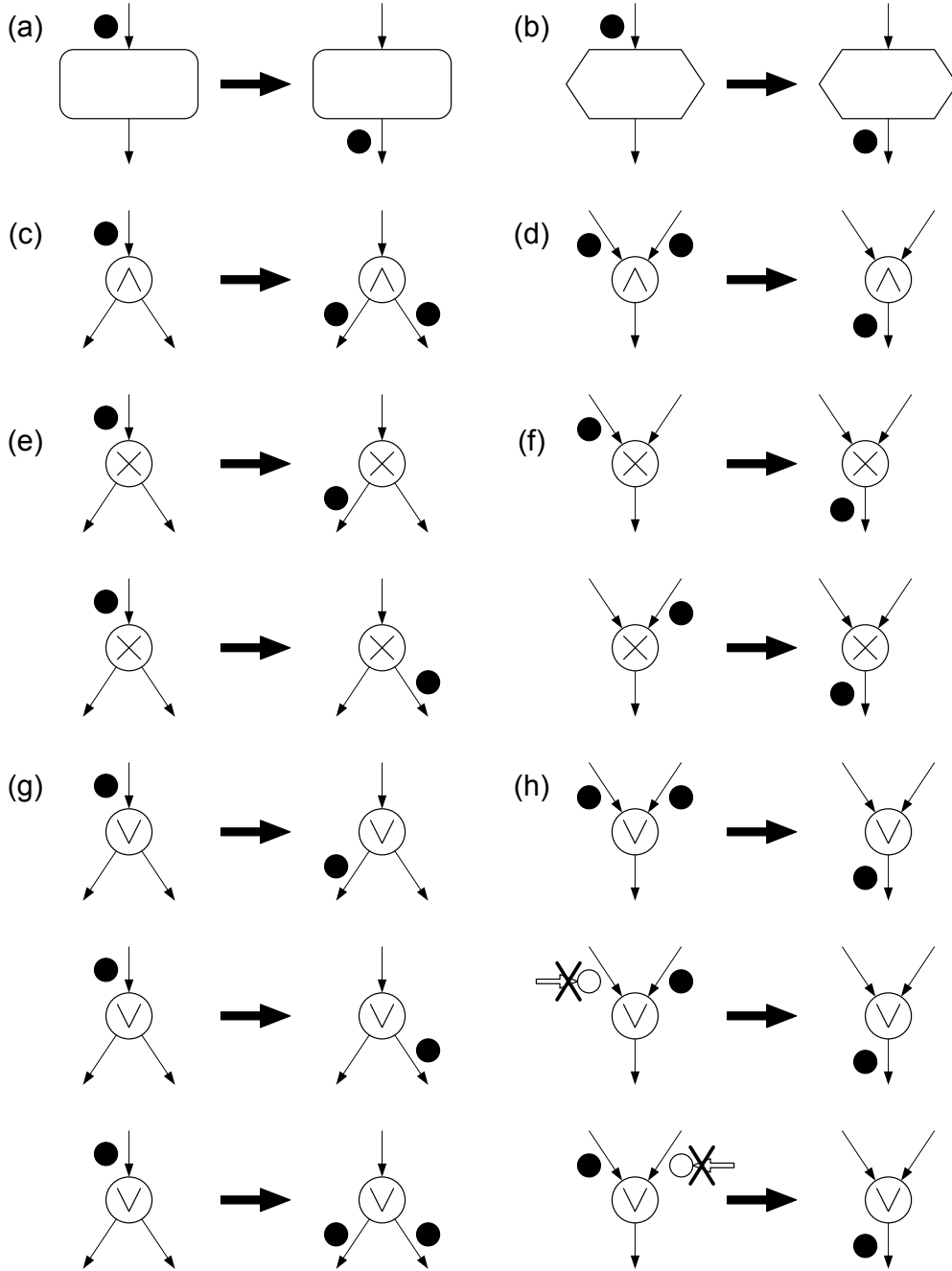


Figure 4. The transition system of an EPC [10].

For our formal definition we assume that the control-flow semantics of a standard EPC and its nodes  $N$  has been defined as a transition relation

$TS_{base} \subseteq M_{base} \times N \times M_{base}$  on the state space  $M_{base}$ . In the most simple case, a marking  $m_{base} \in M_{base}$  is an assignment of tokens to the arcs of the EPC. Such a transition relation has been formalized in [10,8] using different concepts of state. In essence, the semantics of an EPC can be described informally as shown in Figure 4. The AND-split concurrently activates all subsequent branches. The XOR-split represents a choice between one of several alternative branches. The OR-split triggers any combination of the outgoing branches based on conditions. The AND-join waits for all incoming branches to complete, and then it propagates control to the subsequent EPC element. The XOR-join merges alternative branches. The OR-join synchronizes all active incoming branches. Its behaviour is called *non-local* since the state of all transitive predecessor nodes has to be considered [10]. Here, we abstract from different formalizations like [10,8] and take  $M_{base}$  as a starting point for extension with object existence and resource availability. Then we define how  $TS$  of the *iEPC* can be derived from  $TS_{base}$  of the base EPC. In essence, we modify the transition relation of functions to reflect objects and roles. First, we have to define what the base EPC is.

**Definition 4 (Base EPC)** For an  $iEPC_{F,R,O} = (E, F_N, R_N, O_N, nm, C, A, L)$  we define its base EPC as  $EPC_{iEPC} = (E, F, C_{CF}, A_{CF}, l_C^T)$ .

If it is clear which *iEPC* the base  $EPC_{iEPC}$  belongs to we simply write  $EPC_{base}$ . In the following we assume *iEPCs* to be syntactically correct. This way we fulfill syntactic requirements of the base EPC that respective semantics definitions typically assume.

**Definition 5 (Marking of an iEPC)** Let  $EPC_{base}$  be the base EPC of an *iEPC* and  $m_{base} \in M_{base}$  a marking of  $EPC_{base}$ . Then  $m = (m_{base}, \Theta, \Gamma)$  is the marking of *iEPC* with  $\Theta \subseteq \mathcal{P}(O)$  and  $\Gamma \subseteq \mathcal{P}(R)$  with  $\mathcal{P}$  referring to the power set. We say an object  $o$  exists if and only if  $o \in \Theta$ . We say a role  $r$  is available if and only if  $r \in \Gamma$ .

The transition relation relates to the three sets  $O_{in}, O_{out}$ , and  $R$ , such that  $O_{in}$  makes the constraint regarding incoming objects *true*,  $R$  fulfils the role requirements, and  $O_{out}$  complies with the output condition. In this context we need to define four cases:  $\models^{IN}$  for the set of input objects that make the activation condition true based on lower bound satisfaction,  $\models^{CNS}$  for the set of consumed input objects that is restricted by the upper bound,  $\models^{OUT}$  for the set of output objects that make the output condition true, and  $\models^R$  for the set of roles that make the condition on required roles true.

**Definition 6 (Satisfaction of Input, Output, and Role Conditions)** For an *iEPC*, a function node  $f \in F_N$ , a set of objects  $\Theta$ , and a set of roles  $\Gamma$  we define:

- $\Theta \stackrel{IN}{\models} f$  if and only if
  - $\forall_{o \in \bullet^{IN} f \cap O_N, l_O^M(o)=MND} [no(o) \in \Theta]$
  - $\forall_{c \in \bullet^{IN} f \cap C_{IN}, l_O^M(c)=MND} [lwb(c) \leq |no(\bullet^{IN} c) \cap \Theta|]$ .
- $\Theta \stackrel{CNS}{\models} f$  if and only if
  - $\forall_{o \in \bullet^{IN} f \cap O_N, l_O^M(o)=MND, l_O^U(o)=CNS} [no(o) \in \Theta]$
  - $\forall_{c \in \bullet^{IN} f \cap C_{IN}, l_O^M(c)=MND, l_O^U(c)=CNS} [lwb(c) \leq |no(\bullet^{IN} c) \cap \Theta| \leq upb(c)]$ .
- $\Theta \stackrel{OUT}{\models} f$  if and only if
  - $\forall_{o \in f^{OUT} \cap O_N, l_O^M(o)=MND} [no(o) \in \Theta]$
  - $\forall_{c \in f^{OUT} \cap C_{OUT}, l_O^M(c)=MND} [lwb(c) \leq |no(c^{OUT}) \cap \Theta| \leq upb(c)]$ .
- $\Gamma \stackrel{R}{\models} f$  if and only if
  - $\forall_{r \in \bullet^R f \cap R_N, l_R^M(r)=MND} [nr(r) \in \Gamma]$
  - $\forall_{c \in \bullet^R f \cap C_R, l_R^M(c)=MND} [lwb(c) \leq |nr(\bullet^R c) \cap \Gamma|]$ .

These satisfaction conditions are used to define the transition relation of an iEPC. For a transition of a function the lower bound of mandatory input objects and roles is considered for activation. The transition destroys at least as many mandatory input objects as defined as consumable, but not more than the upper bound as specified by the input object range connectors. After that, the lower bound of the mandatory output connectors define the minimum amount of objects to be added to  $\Theta$ , the set of existing objects.

**Definition 7 (Transition Relation of an iEPC)** *Let  $EPC_{base}$  be the base EPC of an iEPC,  $m_{base} \in M_{base}$  a marking of  $EPC_{base}$ , and  $TS_{base}$  its transition system. Then the transition system  $TS$  of the iEPC is defined as follows:*

- For all  $n \in N \setminus F$ :
  - if and only if  $(m_{base}, n, m'_{base}) \in TS_{base}$ , then  $(m, n, m') \in TS$ .
- For all  $n \in F$ :
  - (1) If and only if  $(m_{base}, n, m'_{base}) \in TS_{base}$  and
  - (2) There exists  $\Theta \subseteq no((\bullet^{IN} f \cup \bullet^{IN} (\bullet^{IN} f)) \cap O_N)$  such that  $\Theta \stackrel{IN}{\models} f$  and
  - (3) There exists  $\Delta \subseteq no(\bullet^{IN} f \cup \bullet^{IN} (\bullet^{IN} f)) \cap O_N$  such that  $\Delta \stackrel{CNS}{\models} f$  and
  - (4) There exists  $\Gamma \subseteq nr((\bullet^R f \cup \bullet^R (\bullet^R f)) \cap R_N)$  such that  $\Gamma \stackrel{R}{\models} f$  and
  - (5) There exists  $\Sigma \subseteq no((f^{OUT} \cup (f^{OUT})^{OUT}) \cap O_N)$  such that  $\Sigma \stackrel{OUT}{\models} f$ .

Then  $(m, n, m') \in TS$  with  $m = (m_{base}, \Theta, \Gamma)$  and  $m' = (m'_{base}, (\Theta \setminus \Delta) \cup \Sigma, \Gamma)$ .

Fig. 5 shows an execution instance for function *Edit Dialogs*. The function is enabled as it fulfills the conditions of Def. 7. Firstly, there is a control token on the input arc to the function. Secondly, the combination of existing

input objects (marked with a check) is sufficient to execute *Edit Dialogs*: the mandatory Dialog tracks exist and the optional Voice-over tracks are provided, i.e.  $\{\text{Dialog tracks, Voice-over tracks}\} \stackrel{IN}{\models} \textit{Edit Dialogs}$ . Thirdly, the roles Dialog Editor and Assistant Dialog Editor are available, i.e.  $\{\text{Dialog editor, Assistant Dialog Editor}\} \stackrel{R}{\models} \textit{Edit Dialogs}$ . Firing this function moves the control token to its output arc and adds both the output objects to the set of existing objects, i.e.  $\{\text{Edited dialog tracks, Edit notes}\} \stackrel{OUT}{\models} \textit{Edit Dialogs}$  and  $\emptyset \stackrel{CNS}{\models} \textit{Edit Dialogs}$ .

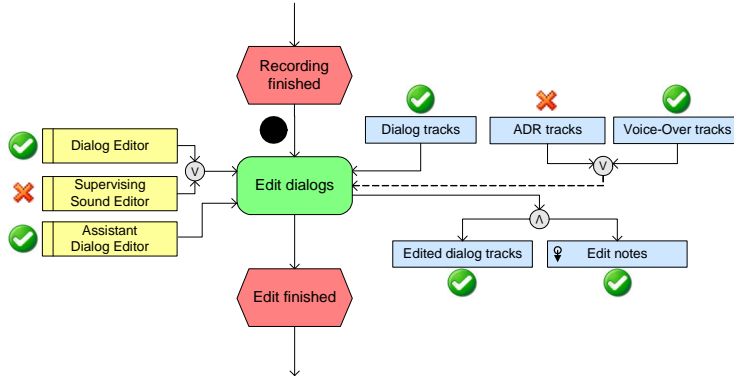


Figure 5. A marking that enables function *Edit dialogs*

The definition of a transition relation adds three sources of complexity compared to plain EPCs. There are different combinations of roles that activate a function. At this stage we do not consider durations and dynamics. Therefore, those roles that are available before the execution of the function remain available afterwards. For simulation purposes one could consider duration of unavailability which affects concurrent tasks for which a role is needed. Furthermore, there can be different combinations of input objects that activate a function. In Fig. 5 any combination of inputs that covers Dialog tracks is allowed. Finally, different combinations of output objects (e.g. if the output connector was an XOR) can introduce non-determinism to the firing of a function. In case of the *Edit Dialogs* example there is only one output set permitted.

Based on this definition of a transition system we can define a reachability graph for iEPCs. In order to do so we first need to define a notion of initial and final marking. Since the definition of those may be different depending on the concept of state, we extend an abstracted version of the definition by [8] to reflect object existence and role availability.

**Definition 8 (Initial Marking of an iEPC)** Let *iEPC* be a syntactically correct iEPC and  $M_{iEPC}$  its marking space.  $I_{iEPC} \subseteq M_{iEPC}$  is defined as the set of *all* possible initial markings, i.e.  $i = (i_{base}, \Theta, \Gamma) \in I_{iEPC}$  if and only if:

- $\exists a_s \in A_s : i(a_s) > 0$ ,
- $\forall a \in A_{int} \cup A_e : i(a) = 0$ ,

- $\Theta \subseteq O_N^i$ ,
- $\Gamma \subseteq R$ .

**Definition 9 (Final Marking of an iEPC)** Let  $iEPC$  be a syntactically correct iEPC and  $M_{iEPC}$  its marking space.  $O_{iEPC} \subseteq M_{iEPC}$  is defined as the set of *all* possible final markings, i.e.  $o = (o_{base}, \Theta, \Gamma) \in O_{iEPC}$  if and only if:

- $\exists a_e \in A_e: o(a_e) > 0$  and
- $\forall a \in A_s \cup A_{int}: o(a) = 0$ .
- There are no requirements regarding output objects and roles.

Based on the set of initial markings  $I_{iEPC}$  and the transition relation  $TS$ , we can calculate the reachability graph. This algorithm determines all initial markings and adds for each possible transition a new marking to the reachability graph. Using the semantics of [11], the reachability graph is finite.

---

**Algorithm 1** Pseudo code for calculating the reachability graph of an iEPC

---

**Require:**  $iEPC, I \subseteq I_{iEPC}$

```

1:  $RG \leftarrow \emptyset$ 
2:  $toBePropagated \leftarrow I$ 
3:  $propagated \leftarrow \emptyset$ 
4: while  $toBePropagated \neq \emptyset$  do
5:    $oldMarking \leftarrow toBePropagated.pop()$ 
6:    $nodeNewMarking \leftarrow currentMarking.propagateTokens(iEPC)$ 
7:    $propagated.add(oldMarking)$ 
8:   for all  $(node, newMarking) \in nodeNewMarkings$  do
9:      $RG.add(oldMarking, node, newMarking)$ 
10:    if  $newMarking \notin propagated$  then
11:       $toBePropagated.push(newMarking)$ 
12:    end if
13:  end for
14: end while
15: return  $RG$ 

```

---

The calculation of  $RG$  requires an  $iEPC$  as input and a set of initial markings  $I \subseteq I_{EPC}$ . Algorithm 1 uses an object-oriented pseudo code notation to define the calculation. In particular, we assume that  $RG$  is an instance of the class *ReachabilityGraph*,  $propagated$  an instance of class *Set*, and  $toBePropagated$  an instance of class *Stack* that provides the methods  $pop()$  and  $push()$ . Furthermore,  $oldMarking$ , and  $newMarking$  are instances of class *Marking*. Finally,  $propagateTokens(iEPC)$  returns a set of (node,marking) pairs including the node that can fire and the marking that is reached after the firing.

In lines 1-3, the sets  $RG$  and  $propagated$  are initialized with the empty set, and the stack  $toBePropagated$  is filled in with all initial markings of the set  $I$ . The while loop between lines 4-14 calculates new markings for the marking

that is on top of the stack *toBePropagated*. In particular, the top marking from the stack is written to *oldMarking* (line 5). Then, in line 6, the pairs of nodes and new markings that can be reached from the old marking are stored in the set *nodeNewMarking*. After that, the old marking is added to the *propagated* set (line 7). In lines 8-12, for each pair of node and new marking a new transition (*oldMarking, node, newMarking*) is added to *RG*. If a new marking has not yet been propagated, it is pushed on top of the *toBePropagated* stack (lines 10-12). Using a stack, the reachability graph is calculated in a depth-first manner. Finally, in line 15 *RG* is returned.

The reachability graph can then be analyzed for verification purposes. The soundness definition for workflow nets cannot be directly used for an iEPC since it may have multiple start and end events. Therefore, we need to extend the property of EPC soundness [8] to iEPCs. This requires that every start arc has an associated initial marking that is included in the set of initial markings. Then, the soundness definition demands that there exists such a non-empty set of initial markings, and that for each initial marking in this set proper completion is guaranteed. Furthermore, there must be a set of final markings reachable from some of these initial markings such that there exists at least one final marking in which a particular end arc holds a token. If that is fulfilled, proper completion is guaranteed for a set of initial markings that cover all start arcs.

**Definition 10 (Soundness of iEPC)** Let *iEPC* be a syntactically correct iEPC,  $N_{CF} = E \cup F_{CF} \cup C_{CF}$  its set of control-flow nodes,  $M_{iEPC}$  its marking space, and  $I_{iEPC}$  and  $O_{iEPC}$  the set of all possible initial and final markings. An *EPC* is sound if there exists a non-empty set of initial markings  $I \subseteq I_{iEPC}$  and a set of final markings  $O \subseteq O_{iEPC}$  such that:

- (i) For each start-arc  $a_s$  there exists an initial marking  $i \in I$  where the arc (and hence the corresponding start event) holds a token. Formally:  

$$\forall a_s \in A_s : \exists i \in I : i(a_s) = 1$$
- (ii) For every marking  $m$  reachable from an initial state  $i \in I$ , there exists a firing sequence leading from marking  $m$  to a final marking  $o \in O$ . Formally:  

$$\forall i \in I : \forall m \in M : (i \xrightarrow{*} m) \Rightarrow \exists o \in O (m \xrightarrow{*} o)$$
- (iii) The final markings  $o \in O$  are the only markings reachable from a marking  $i \in I$  such that there is no node that can fire. Formally:  

$$\forall m \in M : ((i \xrightarrow{*} m) \wedge \nexists m' (m \rightarrow m')) \Rightarrow m \in O$$

Please note that with free-choice behavior of an iEPC, this definition implies that all end arcs can be reached from some initial marking included in  $I_{iEPC}$ . Furthermore, from Definition 7 we can directly derive the following three theorems.

**Theorem 11 (Soundness of iEPC and base EPC)** *If the iEPC is sound,*

the base EPC is also sound.

**Theorem 12 (Initial Objects and Soundness)** *If iEPC is sound and an initial marking  $i = (i_{base}, \Theta, \Gamma)$  is in  $I_{iEPC}$ , then also  $i = (i_{base}, O, \Gamma)$  is in  $I_{iEPC}$  with the complete set of input objects.*

**Theorem 13 (Roles and Soundness)** *If iEPC is sound and an initial marking  $i = (i_{base}, \Theta, \Gamma)$  is in  $I_{iEPC}$ , then also  $i = (i_{base}, \Theta, R)$  is in  $I_{iEPC}$  with the complete set of roles.*

The proof of Theorem 11 builds on the implication of the “if and only if” definition of the transition system of the iEPC and base EPC. Among others, this implies that if a transition of the iEPC can fire, there must be a corresponding transition in the base EPC that can fire, too. Theorem 12 follows from the activation condition  $\stackrel{IN}{\models}$  that builds on the lower bound of the range connectors. This means any enlargement of the set of input objects still fulfills  $\stackrel{IN}{\models}$ . The same argument holds for Theorem 13.

The three theorems have strong implications for an efficient approach to verification of iEPC soundness. Since an iEPC can only be sound if the base EPC is sound, one should first calculate the reachability graph for the base EPC. This way, potential deadlocks can be traced back to control flow problems. When the control flow has been corrected, one can calculate the reachability graph for the iEPC. Since the iEPC can only be sound, if it is sound with the complete set of role and initial objects, these sets should be considered for the initial markings. Obviously, any deadlocks found in this phase stem from missing input objects because all roles are available. Once the object flow is corrected, one can verify the iEPC with different role sets, e.g. when a company is not able to staff all roles defined in the process.

## 4 Related Work

Our work relates to two areas of research: integrated modeling of different process perspectives and verification of process models.

*Integrated modeling of business processes* combines a number of *perspectives* such as the control-flow, the data and the resource perspectives [12]. A common approach to capturing resources in process models is to associate a role, a capability and/or an organizational group to each task [13]. In UML Activity Diagrams (ADs) [14] and BPMN [15], this association is encoded by means of *swimlanes*. Each task (or activity) is associated to a swimlane representing a role or an organizational unit. UML ADs allow multiple swimlanes (or *partitions*) to be associated to an activity. In extended EPCs [16], symbols

denoting roles or organizational units can be attached to tasks, but with no specific semantics. The flow of data and physical artifacts is generally captured by associating objects to tasks. UML ADs support the association of object nodes to tasks to denote inputs and outputs. One can associate multiple objects as input or as output of an activity. The execution of an activity consumes one object from each of the activity's input object nodes and produces one object in each of its output object nodes. Similar features are found in BPMN and extended EPCs. In this paper, we defined more sophisticated role-based resource modeling features, and proposed a more fine-grained approach to object flow modeling, which go beyond those found in UML ADs, BPMN and extended EPCs. Furthermore, we identify suitable verification techniques. Yet, we do not consider data mapping issues which are important for executable languages such as ADEPT<sub>flex</sub> [17], BPEL [18] or YAWL [19]. This body of work is complementary to our proposal.

In the area of *process model verification* the focus has mainly been on control and properties of proper completion. The soundness property [4] and its derivatives play an important role for this purpose. For an overview see [9]. While these properties were originally defined for Petri nets, they have been also applied to other languages, e.g. UML Activity Diagrams, EPCs or BPMN [20,5,7]. Some work directly deals with data flow issues in workflows. In [21] the authors identify different data flow error types. Those errors that do not relate to input constraints can be analyzed with our technique. In [22] the author captures control flow and data flow in UML Activity Diagrams in terms of Colored Petri nets (CPN) [23]. CPN have also been used for the formalization of EPCs in [24]. Our approach extends this work with optional role assignment and non-deterministic data flow. We found optionality of roles and objects to be a major requirement in reference process modeling in practice [25,26]. The non-deterministic choice for using an object and involving a role cannot be expressed appropriately by the formalizations mentioned. Since reference process models describe different variants of a process in a generic way, the variations of input object types and roles have to be captured directly. Our iEPC language addresses this requirement with connectors for objects and roles in an intuitive way, and the transition system allows us to reason about soundness. At this stage we abstract from role-based access control in iEPCs. Most notably, in [27] the authors formalize authorization constraints and discuss their specification and enforcement in workflow management systems. This approach can be adapted for iEPCs in future research.

## 5 Conclusions

In this paper we have discussed quality issues of the role and object perspective in conceptual business process models. These issues are neglected by

existing verification approaches. In order to provide suitable verification support, we have formalized iEPCs, a process modeling language that extends EPCs with objects and roles. While abstracting from the subtle differences of EPC semantics approaches, we have shown how any of these formalizations can be extended with transition rules that consider object existence and role availability as part of the state concept. Furthermore, we have defined three theorems that relate soundness of EPCs to soundness of iEPCs with different initial role and object set. These theorems provide the basis for a systematic verification approach of iEPCs that first identifies control-flow problems, then object-flow problems, and finally suitable role subsets.

The foundations defined in this paper allow us to implement verification support for the notion of soundness in a straight-forward manner. We are currently investigating how EPC transition system implementations such as EPC Tools<sup>2</sup> and ProM<sup>3</sup> can be extended. In future research we want to identify reduction rules that take objects and roles into account. The problem that we have discussed related to concurrent creation and deletion of objects shows that the definition of such rules is not trivial. Still, due to the potential complexity of the EPC reachability graph and the non-determinism introduced by role and object connectors we need such rules for scalable verification. Beyond that, the concept of an iEPC provides different access points for further extensions. In this context, one direction is to consider probabilities at XOR-splits and OR-splits and to assign duration times to functions. This way, each leaf in the reachability graph would capture its probability and cumulated execution time.

## References

- [1] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, 1981.
- [2] J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through sese decomposition, in: B. Krämer, K.-J. Lin, P. Narasimhan (Eds.), *Service-Oriented Computing - ICSOC 2007, Fifth International Conference*, Vienna, Austria, September 17-20, 2007, Proceedings, Vol. 4749 of Lecture Notes in Computer Science, Springer, 2007, pp. 43–55.
- [3] J. Mendling, G. Neumann, W. van der Aalst, Understanding the occurrence of errors in process models based on metrics, in: R. Meersman, Z. Tari (Eds.), *OTM Conference 2007, Proceedings, Part I*, Vol. 4803 of Lecture Notes in Computer Science, Springer, 2007, pp. 113–130.

---

<sup>2</sup> <http://wwwcs.uni-paderborn.de/cs/kindler/research/EPCTools/>

<sup>3</sup> <http://is.tm.tue.nl/~cgunther/dev/prom/>

- [4] W. van der Aalst, Verification of Workflow Nets, in: P. Azéma, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, Vol. 1248 of Lecture Notes in Computer Science, Springer Verlag, 1997, pp. 407–426.
- [5] J. Dehnert, W. van der Aalst, Bridging The Gap Between Business Models And Workflow Specifications, *International J. Cooperative Inf. Syst.* 13 (3) (2004) 289–332.
- [6] M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, D. Edmond, Reduction rules for yawl workflow nets with cancellation regions and or-joins, BPMCenter Report BPM-06-24, BPMcenter.org (2006).
- [7] F. Puhlmann, M. Weske, Investigations on soundness regarding lazy activities, in: S. Dustdar, J. Fiadeiro, A. Sheth (Eds.), Business Process Management, 4th International Conference, BPM 2006, Vol. 4102 of Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 145–160.
- [8] J. Mendling, W. van der Aalst, Formalization and Verification of EPCs with OR-Joins Based on State and Context, in: J. Krogstie, A. Opdahl, G. Sindre (Eds.), Proceedings of the 19th Conference on Advanced Information Systems Engineering (CAiSE 2007), Vol. 4495 of Lecture Notes in Computer Science, Springer-Verlag, Trondheim, Norway, 2007, pp. 439–453.
- [9] M. Weske, Business Process Management: Concepts, Languages, Architectures, Springer-Verlag, 2007.
- [10] E. Kindler, On the semantics of EPCs: Resolving the vicious circle., *Data & Knowledge Engineering* 56 (1) (2006) 23–40.
- [11] J. Mendling, B. van Dongen, W. van der Aalst, Getting Rid of the OR-Join in Business Process Models, in: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC’07), IEEE, Annapolis, Maryland, USA, 2007, pp. 3–14.
- [12] S. Jablonski, C. Bussler, Workflow Management: Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, London, UK, 1996.
- [13] W. van der Aalst, K. van Hee, Workflow Management: Models, Methods, and Systems, MIT press, Cambridge, MA, 2002.
- [14] G. Engels, A. Förster, R. Heckel, S. Thöne, Process Modeling Using UML, in: M.Dumas, W. van der Aalst, A. ter Hofstede (Eds.), Process-Aware Information Systems, Wiley, 2005, pp. 85–117.
- [15] S. White et al., Business Process Modeling Notation (BPMN), Version 1.0 (2004).
- [16] A. Scheer, ARIS - Business Process Frameworks, 3rd Edition, Springer, Berlin, 1999.
- [17] M. Reichert, P. Dadam, ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control, *Journal of Intelligent Information Systems* 10 (2) (1998) 93–129.

- [18] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. Liu, R. Khalaf, D. Koenig, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu, Web services business process execution language version version 2.0, Committee specification 31 january 2007, OASIS (2007).
- [19] W. van der Aalst, A. ter Hofstede, YAWL: Yet Another Workflow Language, *Information Systems* 30 (4) (2005) 245–275.
- [20] R. Eshuis, R. Wieringa, Tool support for verifying uml activity diagrams, *IEEE Trans. Software Eng.* 30 (7) (2004) 437–447.
- [21] S. W. Sadiq, M. E. Orlowska, W. Sadiq, C. Foulger, Data flow and validation in workflow modelling, in: K.-D. Schewe, H. E. Williams (Eds.): *Database Technologies 2004, Proceedings of the Fifteenth Australasian Database Conference, ADC 2004, Vol. 27 of Conferences in Research and Practice in Information Technology*, Australian Computer Society, 2004, pp. 207–214.
- [22] H. Störrle, Semantics and verification of data flow in uml 2.0 activities, *Electr. Notes Theor. Comput. Sci.* 127 (4) (2005) 35–52.
- [23] K. Jensen, *Coloured Petri nets: basic concepts, analysis methods and practical use: volume 1, 2nd Edition*, Springer-Verlag, 1996.
- [24] K. van Hee, O. Oanea, N. Sidorova, Colored Petri Nets to Verify Extended Event-Driven Process Chains, in: R. Meersman, Z. Tari (Eds.), *Proceedings of CoopIS/DOA/ODBASE 2005, Vol. 3760 of Lecture Notes in Computer Science*, Springer-Verlag, 2005, pp. 183–201.
- [25] M. La Rosa, A. ter Hofstede, M. Rosemann, K. Shortland, Bringing Process to Post Production, in: *Proceedings of the International Conference "Creating Value: Between Commerce and Commons"* (forthcoming) Brisbane, Australia, 25-27 June 2008.
- [26] M. La Rosa, M. Dumas, A. ter Hofstede, J. Mendling, F. Gottschalk, Beyond Control-Flow: Extending Business Process Configuration to Resources and Objects, in *Proceedings of 27th International Conference on Conceptual Modelling (ER 2008)*, *Lecture Notes in Computer Science*, Springer-Verlag, 2008.
- [27] E. Bertino, E. Ferrari, V. Atluri, The Specification and Enforcement of Authorization Constraints in Workflow Management Systems, *ACM Transactions on Information and System Security (TISSEC)* 2 (1) (1999) 65–104.