Verifying Properties of (Timed) Event Driven Process Chains by Transformation to Hybrid Automata

Stefan Denne*
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3
66123 Saarbrücken

Abstract: Event-driven Process Chains (EPCs) are a commonly used modelling technique for design and documentation of business processes. Although EPCs have an easy-to-understand notation, specifying entire information systems leads to rather large and complex models. Questions like for instance the termination of a process (within some given time)—easy to answer for small EPCs—can hardly be answered for those models. Nevertheless, questions like these can be vital for the execution of the described business processes. Whereas simulation might be able to give a hint on whether the process terminates, only verification can give such guarantees.

In this paper we introduce a method to verify properties of what we call *Timed EPCs* (EPCs annotated with time attributes). We transform Timed EPC to (hybrid) automata and thereby define EPCs formally. Based on the formal definition, properties of EPCs (like e. g. is the ending event always reached within 20 time units) can be verified by transforming these properties to corresponding properties of the resulting automata. The transformation of EPCs and properties works fully automatic. The ultimate verification takes place in utilising commonly available verification tools.

1 Introduction and Overview

Verification is a method to gain assertions on a system's behaviour. As for EPCs, verification is rarely used . And if at all, with a focus on the correctness of the EPCs. In our approach verification enables the user to get information about the validity of the design of the modelled EPC. Shortcomings can be revealed, quality as well as performance can be assured at a desired level. With *user defined properties* in the sense of Rump [Rum98] it becomes possible for the user to prove assertions 'the account will always be checked before any withdrawal is granted'. If the proof fails, a counter example in form of a path in the EPC can be provided. Since we focus on *Timed EPC* the user will even be able to verify time properties as, for instance, 'the process chain always terminates within 30 seconds'.

In this paper we will introduce a method for the verification of properties of Timed EPCs. Verification can be done in two ways. One is to define semantics on EPCs directly (as it has been done by Nüttgens and Rump [NR02]). The second possibility is to utilise

*stefan.denne@dfki.de

transformation to models that are formally defined and for which it is well known how verification is performed (even supported by tools). We chose the latter approach and give a formal transformation of EPCs to hybrid automata. (Thereby formal semantics of EPCs are defined indirectly.)

This paper will essentially concentrate on the basic transformation ideas even if this means to somewhat restrict the EPCs under consideration. That is to say we define, in terms of a context free grammar, a suitable subset of EPCs which covers a broad variety of models without being meant to be comprehensive.

This work has been done formally (cp. [Den06]) but for this paper we will abstract from technical details and illustrate the transformation of EPCs and corresponding properties using figures and examples. By the transfer of Formal Methods to the field of business processes and the use of ready-to-go tools, the verification—even of timed properties—is possible with reasonable effort.

This paper is organised as follows: we first describe the restriction of the EPCs we use and introduce the aspects of time we will consider. Then we formulate and classify the properties we can verify by the transformation to automata. Afterwards we introduce the grammar that we use for the representation of EPC. In the succeeding sections, we define what we mean by automata. The transformation is described in several steps according to the grammars used. We will not present the technical details but the overall plot using examples to illustrate the transformation process and the results. We also show how to describe and transform the properties of the EPCs. In Section 4 we give a brief glance at some features not presented in this paper. The last but one section gives a brief overview about related work. We conclude the paper with a summary about what has been achieved and what can be done as future work.

2 Modelling

In this section, we introduce the different modelling techniques we use in our approach. First, the EPCs that will be transformed are characterised. Then we introduce the automata that are the target of the transformation.

2.1 Representation of EPCs

In the representation of the EPCs we use we try to be as close as possible to commonly used EPCs. Nevertheless, we will use a restricted form of EPCs in order to minimise the number of transformation rules necessary. We will not deal with open ends, this means, concurrent chains join in any case. Likewise, we do not have joins without splits before, that means we only have one starting event. Additionally the inner structure of concurrent and alternative chains will not allow two (or a multiplicity of two) elements (functions or events). Further we will only consider two different cases of connector combinations, namely XOR–XOR and AND–AND connector pairs (we do not consider OR-OR connector pairs). We call

EPCs with identical connector combinations *regular*.

Despite these restrictions the grammar introduced below fulfils the informal definition of EPC given by [KNS92], and [HKS93].

2.1.1 The use of Time within EPCs

Initially, time has intentionally not been considered in EPC. By leaving out time, modelling with EPCs does not have to consider problems that time involves. For example, simultaneous functions and events do not occur,[1] since a (temporal) 'before' and 'after' does not exist.

The the simulation facility integrated in the ARIS Toolset[2] uses given time attributes to calculate process ratios, for example the minimal and the maximal time to pass from the starting to the ending event of the chain. Functions can be annotated with time slots describing a 'stage'in the progress of a function. *Setup time* is the amount of time required to get ready to execute a function, *Process time* is the time needed for its execution.[3] Looking for example at a machine that heats metal before bending it, one can identify these two stages; setup time: heating the metal and process time: bending it. The time attributes are related to each other. For instance 'setup time' lies before 'process time'.[4]

We call EPCs that have time attributes *Timed Event-Driven Process Chains*.

2.1.2 Properties of EPCs

Properties of an EPC can be described as situations (or as sequences of situations) that—in terms of events and functions—occur or do not occur in the EPC.

Imagine a shop selling books. Before any order is processed it shall be checked whether the book is in the stock to guarantee a delivery date. In order to guarantee this check, an appropriate property must hold for the model. In other words: if one can verify that such a property holds, the delivery date can be guaranteed. If the model is small, this might be easy to check, but if the model is complex or it is large, it might not be obvious that this check is performed in all cases.

As in Bernard et al. ([BBF$^+$01, 79, 83, 91, 103]), properties can be classified in *Reachability properties:* is a situation[5] reachable under some circumstances? *Safety properties:* will an undesirable situation never occur (under certain condition)? *Liveness properties:* will a

[1]If an event E_1 occurs *and* an event E_2 occurs, this 'and' is a logical 'and'.

[2]The ARIS Toolset is part of the ARIS Design Platform. (IDS Scheer AG)

[3]Other time attributes are: *Waiting time* — the time that needs to pass before a function can be started (for example because a machine is occupied before) and *Transfer time* — the time that is needed to enter a function (for instance to pass some material to the current function). 'Transfer time' is an attribute that is assigned on arcs leading to a function. Thus, we interpret 'transfer time' to belong to the (following) function.

[4]'Transfer time' lies before 'waiting time', 'waiting time' lies before 'setup time', 'setup time' before 'process time'. 'Waiting time', 'setup time' and 'process time' are associated to a function.

[5]A situation can be a (boolean) combination of some events or functions.

situation ultimately occur? *Fairness properties:* will a situation, under certain conditions occur (or fail to occur) infinitely often?

In general, properties of EPCs describe a behaviour that corresponds to certain sequences of events and functions.

Accordingly to Rump ([Rum98, 120]) there are typical types of (user defined) properties concerning elements (events and functions) of EPCs. The elements E_1 and E_2 can either be events or functions.

1. There is a concrete sequence represented by the EPC-schema, in which E_1 is activated.
2. E_1 is activated in any possible sequence.
3. If E_1 is activated in a sequence, then E_2 will sometime be activated.
4. E_2 will never be activated, after E_1 has been activated.
5. In a sequence of the business process E_1 and E_2 will never be activated both, irrespective of their order
6. The element E_1 will only be activated once in one sequence.

Note that the first two questions are reachability properties and the following are liveness properties. The term 'activated' corresponds to 'reachable' in our terminology.

In general, temporal logics are eligible to express properties of that kind. We use a CTL-like[6] temporal logic to describe these properties (see Section 3.3).

For Timed EPCs, the properties above can be enriched by assertions about time. One may ask for example: Is there a sequence where a situation E_1 is reached in less than five time units? Or the property might be: E_2 is always reached within 20 time units. (See Section 3.5).

A formal description of CTL and TCTL is presented in [Den06].

Another type of property is the maximal and/or minimal time that is needed between some elements (events or functions) within the EPC. In comparison to the properties described so far, a question about the validity does not lead to a 'yes' or 'no', but to a parameter that is the maximal or minimal time. For instance: 'There is a sequence, where E_1 is reached in less than P time units.' The parameter P then represents the maximum in focus.

2.1.3 A Grammar for Timed EPCs

The EPCs we use are syntactically defined by a context free grammar. Context free grammars define (context free) languages using a set of derivation rules. This definition is constructive. By the application of the rules of the grammar a sentence of the language may be generated .

[6]CTL is an abbreviation for Computation Tree Logic

The advantage of the constructive approach is that the rules of the grammar describe how a proper EPC looks like. Only such EPCs can be constructed that were defined by the grammar.

In this paper we focus on Timed EPCs. Every time attribute introduced in Section 2.1.1 represents a duration and is associated to a function. We introduce two different time attributes only: ‘setup time‘ and ‘processing time’, as we only show how this is done in principle. These time attributes appear in a fixed order: ‘setup time’ lies before ‘processing time’. We will keep this order in the grammar. Only ‘processing time’ is mandatory, therefore we need two rules; one that contains ‘processing time’ only (rule TS10) and a second that contains both, ‘set-up time’ and ’processing time’ (rule TS9).

Of course there are many different grammars possible that have other or additional rules that allow the construction of EPCs where different elements (like time attributes, organisational units) are possible[7]

G$_{TS}$ = (N$_{TS}$, T$_{TS}$, R$_{TS}$, EPC) with N$_{TS}$ = {EPC, E-PART, F-PART, T-ATTR}, T$_{TS}$ = {Event, Function, AND , XOR, (,), Setuptime, Processtime, →} and

R$_{TS}$ = {	EPC	:==	Event	→	F-PART	→	Event	(TS1)
	E-PART	:==	E-PART	→	F-PART	→	E-PART	(TS2)
	F-PART	:==	F-PART	→	E-PART	→	F-PART	(TS3)
	E-PART	:==	Event					(TS4)
	F-PART	:==	Function (T-ATTR)					(TS5)
	E-PART	:==	AND (E-PART,E-PART)					(TS6)
	F-PART	:==	AND (F-PART,F-PART)					(TS7)
	E-PART	:==	XOR (E-PART,E-PART)					(TS8)
	T-ATTR	:==	Setuptime, Processtime					(TS9)
	T-ATTR	:==	Processtime }					(TS10)

The rules TS1 to TS3 allow the construction of linear chains of an arbitrary length. (TS1 assures hat each chain will start and end by an event.) Concurrent branches are introduced by the rules TS6 and TS7, TS8 constructs alternative branches. Events are derived by TS4. The rule TS5 extends a function with a list of time attributes.

2.2 Automata

Automata are finite graphs whose nodes correspond to global states. Such global states represent some sort of general observational situation, as, for instance, the heater is on or the the heater is off. Automata change there states by travelling along a transition that connects two states.

In this section we only give a informal introduction to automata as far as they were used

[7]In [Den06] a grammar including organisational units and time attributes is defined and used for the transformation into timed automata.

for the transformation. Formal details can be found in [Den06].

2.2.1 Communicating Automata

We use communicating automata to model a system by modelling its components. Each component is represented by an automaton and can be rather simple. But as the components are able to communicate with each other the behaviour of one component can depend on the behaviour of some other. Thus, the behaviour of the system is the result of the composition of the behaviours of the components. Consequently, a system with simple components can have a quite complicate behaviour.

We prefer a modular approach to keep the transformation straightforward. Each of the rules of the context free grammars will be transformed into a small set of automata. All sets are composed to a system representing the behaviour of the original EPC.

2.2.2 Composition of Communicating Automata

When building systems out of components we have to define if and how they interact. If there is no interaction at all the behaviour of the resulting system is the combination of the independent behaviours of the components and, as the behaviour of each component is determined by the set of states, the system behaviour is the *Cartesian product* of the states of the automata.

Automata can interact by synchronising their discrete steps. The synchronised automata cannot perform a step without the other performing a corresponding step as well. With the system behaviour being a combination of the components, the number of system states is reduced compared to the non-synchronised system. If two automata $\mathcal{A}$ and $\mathcal{B}$ share the same synchronisation label l, then a transition of $\mathcal{A}$ labelled with $l!$ (or $l?$) must be accompanied by a transition of $\mathcal{B}$ labelled with $l?$ ($l!$) too.

Figure 1 shows two automata that synchronise via a corresponding synchronisation label l. Both of them start concurrently in their initial location (n_1 and n_5). While automaton $\mathcal{A}$ can travel from location n_1 to location n_2, $\mathcal{B}$ being in n_5 cannot leave because it has to synchronise with some partner having a corresponding label ($l_1!$). So, $\mathcal{A}$ being in location n_2 and $\mathcal{B}$ in n_5, the two automata can only perform a common step leading from n_2 to n_3 and from n_5 to n_6. The subsequent transitions can be taken independently.

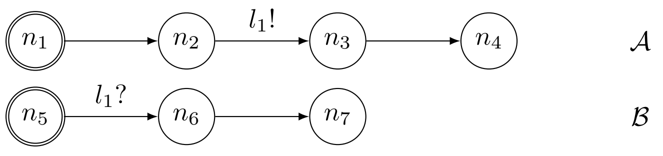


Figure 1: Two synchronising automata

Because we use binary synchronisation (one automaton sending a signal with one —out of several other—receiving it), only two automata out of the set (of automata) can synchronise by a *compatible label*[8]. As our approach requires to synchronise more than two automata we use label sets. When travelling along a transition all the labels in the label sets have to be synchronised regardless of the number of automata to be synchronised.

Here a brief example for compatible label sets: the three label sets $L_1 = \{l!\}$, $L_2 = \{l?, m!\}$ and $L_3 = \{m?\}$ are compatible, because every label in the different label sets has a compatible partner.

The composition of two automata leads to a new automaton. Each location of the composed automaton is a tupel combination of two locations of the original automata. The starting location of the automaton $\mathcal{C}$ $\langle n_1, n_5 \rangle$ is a combination of the automata $\mathcal{A}$ and $\mathcal{B}$. The following two steps of $\mathcal{C}$ are the only possible steps the combined composed system can perform. (The first step is restricted as only the automaton $\mathcal{A}$ can perform a step, the second step is the one that is synchronised via the label l). The labels are no longer required after the composition.

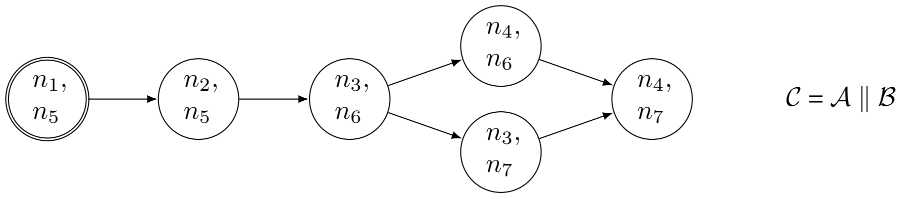


Figure 2: Composition of two communicating automata

2.2.3 Timed Automata

Timed Automata (a special kind of hybrid automata) are an extension of the communicating automata. The global states of communicating automata were discrete. That means after the entry into a state no further action happens while the automaton remains within the state. Within the global states of timed automata some continuous activity takes place. A *clock* (or chronometer or watch) defined in a location, rises continuously (by a rate of 1). Depending on the value of that clock the automaton can change from one state to the other travelling along a transition. These transitions are usually guarded with some constraint formula[9] , that is required to hold if the transition is supposed to be taken. Similarly, nodes have some attached constraint formula that describes an invariant for this very node, this means, some property that has to be true while the system resides in this node. The dynamics of the systems behaviour, on the other hand, is given by a description of how the data changes with time. Additionally, transitions are annotated with a general assignment that

[8]Compatible labels have the same name and one has a exclamation mark and the other a question mark. The exclamation mark has the meaning of ‘sending’and the question mark the meaning of ‘receiving’. If there is more than a question mark as a possible partner, the exclamation mark can arbitrarily synchronise with one of the question marks available.

[9]A constraint formula is for instance a inequality that limits a *clock* to an certain value like $c \leq 5$.

is responsible for the discrete action to be performed by taking the transition (For example resetting a clock to zero).

Timed automata can behave in two ways: (1) by the change of one state into another (by passing a transition) or (2) by the passing of time, which changes the value of a clock.

Figure 3 shows an example of an timed automaton. The automaton has a clock t that is set to zero when passing the transition $\langle n_1, n_2 \rangle$. The invariant $t \leq 5$ forces that the location has to be left when the clock has reached 5 time units and the guard $t \geq 4$ hinders to leave the location before the clock t is at least equal to 4 time units. Thus the location n_2 can be left when the clock t is at least 4 and must be left as soon as it is 5. The automaton does not synchronise with other automata therefore the label set is empty and the transitions do not have labels.

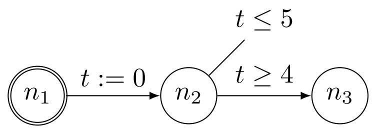


Figure 3: A timed automaton

2.2.4 Composition of Timed Automata

The composition of timed automata is similar to the composition of communicating automata. The transitions with labels are the synchronisation points of the automata. For the composed location the invariant is the conjunction of the invariants of the location of the original automata. In order that the new transition (that leads from the actual tupel of the composed automaton to the following tupel) can be passed all guards of the transitions of the original automata (for that step) have to be true. The actions for the new transition are just the set of all actions of the original transitions (that would have been taken).

3 Transformation

The transformation of Event-driven Process Chains to automata is defined recursively on the structure of EPCs by transformation rules. This structure is immediately introduced by the grammar rules that allow to construct EPCs. The processing of a transformation rule leads to a set of new automata or modifies an existing set of automata in order to model the behaviour of the EPC.

Basically, functions are transformed into automata locations and events are transformed into transitions. We keep information about which function or event is related to which location or transition by a function.

We will not give the formal transformation rules and technical details of the transformation here, they can be found in [Den06].

3.1 Transformation of Linear Sequential Chains

To illustrate how the transformation works, we first show how sequential chains are transformed. These chains are constructed using the rules TS1 to TS5 of the grammar $\mathbf{G}_{TS}$.

In the following, the transformation rule for the transformation of TS1 will be presented in detail and an example will show the complete transformation of a sequential chain. The examples used will have no time attributes[10].

We assume that the set of automata $\mathfrak{B}$ is already transformed (see Figure 4 (1)). The dashed arrow symbolises that the structure of the transformed set is not relevant (and not available) for the transformation of the current transformation step.

The transformation rule for TS1 creates two automata (each of them in a created set) that represent the starting and ending event and modifies the existing automata in a way that the sets are 'executed'linearly after each other. This is done by inserting compatible synchronisation labels (Figure 4 (2)).

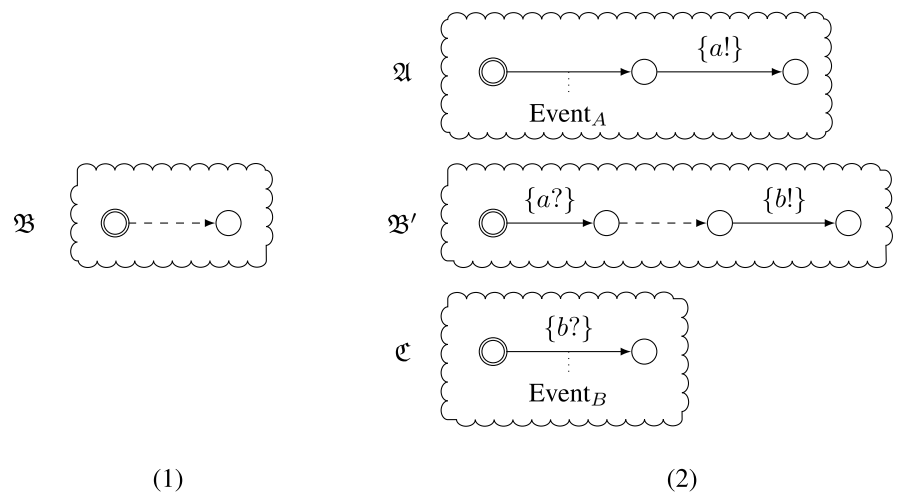


Figure 4: Sketch of the transformation of the rule TS1

EXAMPLE 1 (TRANSFORMATION OF A SIMPLE SEQUENCE) As an example we transform the following simple sequence (Figure 5) into automata.

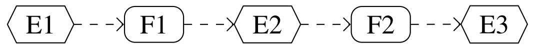


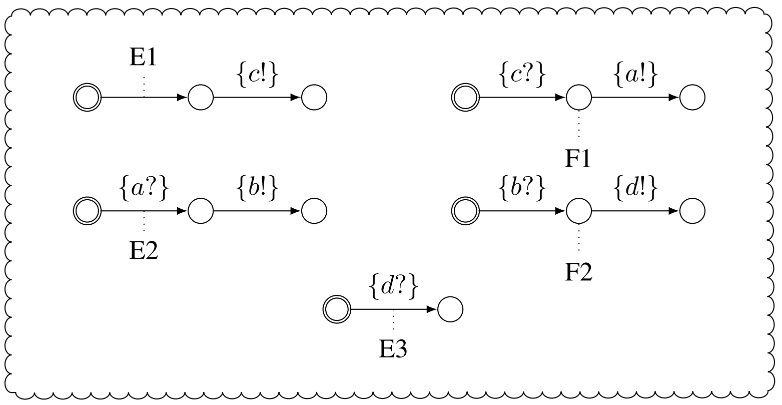
Figure 5: A simple linear sequence

The following sequence shows how the example is constructed using the rules of the grammar $\mathbf{G}_{TS}$.

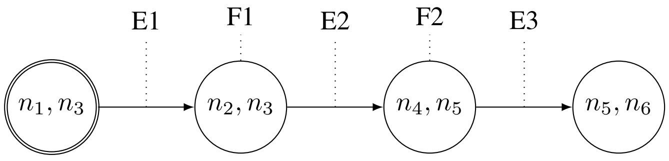
[10]Formally this means that instead of the rule TS5 the following rule is used: F-PART :== Function.

EPC ⇒$\text{E}_1 \rightarrow$ F-PART$_1 \rightarrow \text{E}_3$ (rule TS1)
F-PART$_1$ ⇒F-PART$_2 \rightarrow$ E-PART$_1 \rightarrow$ F-PART$_3$ (TS3)
F-PART$_2$ ⇒F$_1$ (TS5) E-PART$_1$ ⇒E$_2$ (TS4) F-PART$_2$ ⇒F$_2$ (TS5)

The transformed set of automata looks as follows:



Composing the set of automata , we get the following automaton:



The sequence of referenced events and functions corresponds to the sequence of the original EPC (of Figure 5).

3.2 Transformation of Concurrent and Alternative Chains

Concurrent and alternative chains occur when EPCs have AND and XOR branches. For the transformation of these branches we introduce the concept of *Schedulers*. A scheduler is a regular automaton, but its locations and transitions do not have a reference to any event or function. Schedulers control other automata. A scheduler synchronises the starting and the ending of the ‘underlying’ automata. In the case of concurrent chains the first and the last location are to be synchronised by the scheduler; on alternative chains the scheduler synchronises but one chain only.

Two concurrent branches of an EPC have to be started and ended synchronously. This behaviour is transferred to automata. The two sets of automata are controlled with a scheduler that synchronises both chains. Using two label sets $\{a!, c!\}$ and $\{b!, d!\}$ the scheduler

$\mathcal{S}_{\text{AND}}$ synchronises the resulting automata sets each representing the two branches. The two scheduled sets $\mathfrak{A}$ and $\mathfrak{B}$ have correspondent labels. (Figure 6).

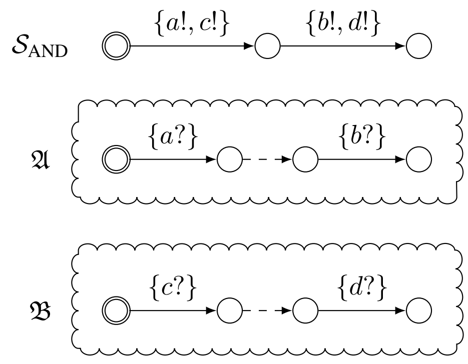


Figure 6: The scheduling of an AND-part

Each of the XOR-branches is an alternative path within the EPC. So, the scheduler must alternatively synchronise with one of the transformed sets representing a branch of the original EPC.

Let us suppose the scheduler has the label a! at the start (and b! at the end) and each of the sets ($\mathfrak{A}, \mathfrak{B}$) has the same label a? (and b?). Then there is only one compatible pair a! and a? at a time. Hence only one of the sets will by synchronised with the scheduler S_{XOR}. A sketch of the transformation is shown in Figure 7.

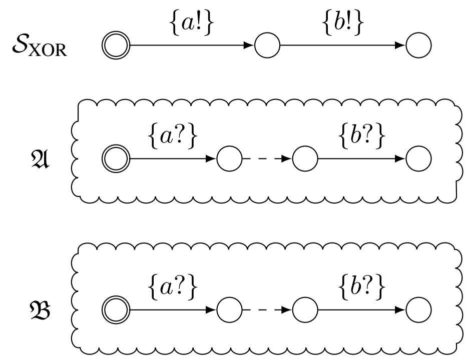


Figure 7: The scheduling of an XOR-part

Example 2 (Transformation of Concurrent and Alternative Chains) Given is the EPC shown in figure 8. We omit the details of the transformation here and show the resulting set of automata in Figure 9. The named locations were used to show the transformation of properties (Section 3.3). The composition of the resulting automata is depicted in Figure 10. We omit the details but show the reference to the original EPC. The original EPC 'reappears ' when travelling along the locations and transitions.

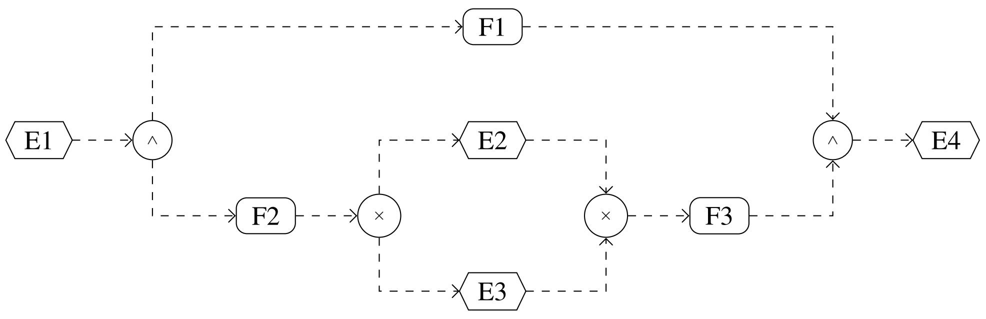


Figure 8: Example 2 — Concurrent Chains

3.3 Transformation of Properties of EPCs

In 2.1.2 we have raised several kinds of properties. With respect to EPCs these questions can be expressed in CTL[11]. Here an example.

EXAMPLE 3 (EXPRESSING PROPERTIES OF EPCS USING CTL) The properties described in this example refer to the EPC of Figure 8.

	Description	CTL-formula
(1)	Is the event E_3 reachable?	EF E_3
(2)	Is the (end) event E_4 always reached?	AF E_4
(3)	If the (start) event E_1 is reached then inevitably the (end) event E_4 is reached too.	$AG(E_1 \Rightarrow AF(E_4))$
(4)	Either E_2 or E_3 are reached (never both)	$(AG((E_1 \Rightarrow AG(\neg\ E_2)) \wedge AG(E_2 \Rightarrow AG(\neg\ E_1))))$

Intuitively AF, EF, AG, EG, AU and EU, mean "inevitably", "possibly", "always", "possibly always", "inevitably until" and "possibly until".

As CTL is also used for describing the properties of automata, the transformation can easily be done. Recall, that events and functions are related to transitions and locations. Using that relation, a property of an EPC is transformed directly to a property of the resulting automata.

[11]Here, we use a CTL-like language since we do not use all temporal qualifiers.

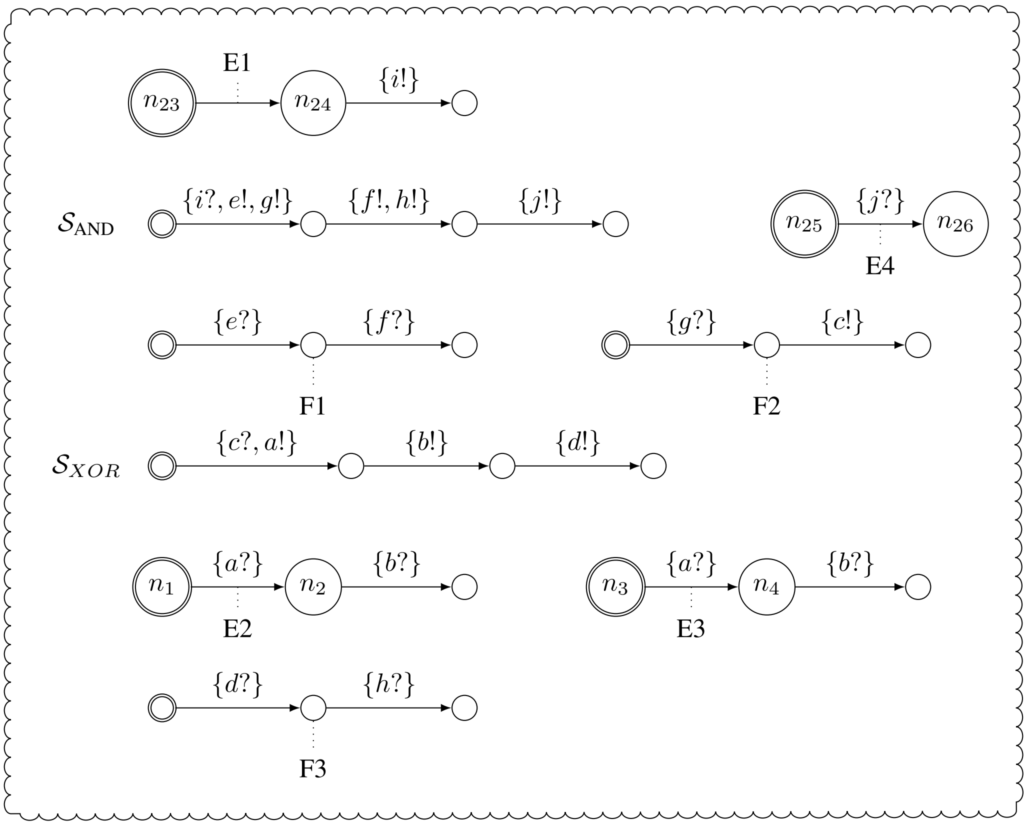


Figure 9: Transformed set of automata of Example 2

Then this property can be proved or rejected (with a counterexample) using a model checker like UPPAAL ([BLL$^+$95]) or HyTech ([HH95]).

The properties above are transformed to properties referring to transitions of the transformed set of automata:

(1) EF $\langle n_3, n_4 \rangle$
(2) AF $\langle n_{25}, n_{26} \rangle$
(3) AG($\langle n_{23}, n_{24} \rangle \Rightarrow$ AF($\langle n_{25}, n_{26} \rangle$))
(4) (AG(($\langle n_1, n_2 \rangle \Rightarrow$ AG($\neg\langle n_3, n_4 \rangle$)) $\wedge$ AG($\langle n_3, n_4 \rangle \Rightarrow$ AG($\neg\, \langle n_1, n_2 \rangle$)))))

3.4 Transformation of Timed EPCs

In EPCs the time we are dealing with is represented by attributes attached to functions.

Timed automata use (reset) clocks, invariants and guards to deal with time. The time

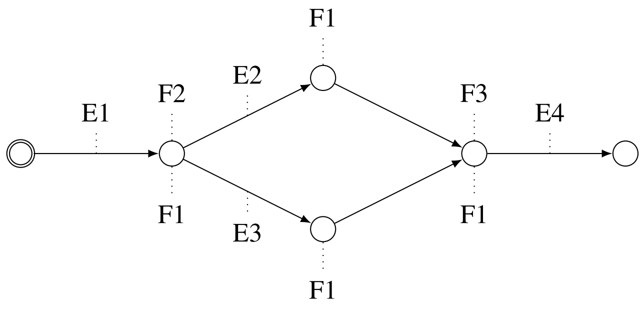


Figure 10: Composed automaton of Example 2

spent in each location can be restricted by defining how long to stay and when to leave a location.

For the transformation of Timed EPC we use so-called *Urgent Transitions*. In general, time can pass in a location when no time restriction is given. If no invariant forces to leave a location it is possible to stay there infinitely (long). It is not mandatory to take a transition. Urgent transitions change this behaviour. An urgent transition must be taken 'as soon as possible'. This means, if a guard of an urgent transition is true (what is the case if there is no guard given explicitly) it must be taken immediately. Thus, no time will ever pass in a location with an urgent transition if no time restrictions are given.

Transformation of Functions Figure 11 (1) shows the transformation of a function F1 that has no time attributes. The transformation of such a function leads to (a set of automata with) a single automaton having two locations and one transition. The second location is associated with the corresponding function.

If the function F1 has a processing time p of 5 time units a clock c is inserted. The clock is set to 0 when entering the location n (by the assign $c := 0$). A guard (on an inserted transition) hinders the location n to be left until 5 time units passed. An invariant forces that the location n is left when more than 5 time units passed (see Figure 11 (2)).

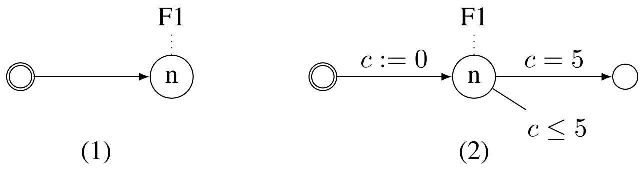


Figure 11: Changes of the transformation when a has got a time attribute.

Transformation of Events There is no need to change the transformation of events. (As we now use urgent transitions no time will pass in location that are not guarded.)

Here is an example to show how an EPC with 'setup-time' (s) and 'process time' (p) is transformed.

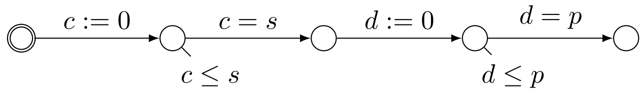


Figure 12: Transformation of 'setup-time' and 'process time'

3.5 Transformation of Properties of Timed EPCs

A so-called *Timed Property* is a normal property provided with additional time constraints. For example: 'E_4 is inevitably reached and *the clock t is greater or equal to 20*'. Timed properties can be expressed using TCTL. In TCTL this property is expressed as: AF($E_4 \wedge t \geq 20$). $t \geq 20$ is a so-called *Constraint Formula*[12]

The clock that is referred to, is not part of the system modelled. It can be imagined as a 'global' clock that is started at the beginning of the EPC. The time spent in the functions of the EPC is then sum up in this 'virtual' clock.

Basically, this is also done when a timed property is transformed: A new clock is introduced at the 'very beginning' and reset. The 'very beginning' is the transition that corresponds to the transformed starting event. (If the property consists of several clocks each of them must be inserted analogously.) After this preparatory work, the property can be translated in terms of automata.

Minimum and Maximum Durations

Another kind of properties proposed are minimum and maximum durations. These durations refer to a selected starting and ending element in the EPC. In general, it is interesting to know, how long an entire business process lasts at least and at most. We will consider the starting event E_{start} and ending event E_{end} of the EPC for explaining the method.

First, a new automaton $\mathcal{A}$ with three locations and two transitions is created. This automaton is synchronised with the (already transformed) set of automata. The first transition is synchronised with the transformation that is associated with E_{start}. The last transformation is synchronised with the transformation that is associated with E_{end}. A new clock is reset travelling along the first transition. Figure 13 shows an example of a suchlike automaton, having a clock t and two labels z_1 and z_2. (The compatible labels z_1! and z_2! are inserted to the transitions associated with E_{start} and E_{end}.)

[12]A *Constraint Formula* is either $\top$ (truth) or $\perp$ (falsity) or is an equality or inequality between constraint terms. Examples: $t \leq 5$, $p = a + 17$, etc. A *Constraint Term* is either a variable (out of a fixed variable set) or a real valued constant or some arithmetic operation (addition, subtraction and multiplication) of some (real valued) constants. Examples: x, $x + 5$, 17, $17 + 9 - 144 * 6$, $x * 5 - 3$.

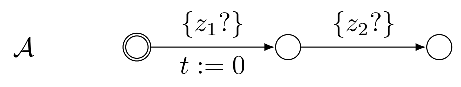


Figure 13: The automaton for the determination of minimum and maximum durations

The property used to determine the maximum or minimum duration of the EPC contains a parameter. Using an appropriate model checker (like for instance HyTech), the model checker tries to find a value for the parameter, so that the property holds.

In general these properties have the following form: 'Is it inevitably that a location n (or transition $\langle m_1, m_2 \rangle$) is reached and the clock $t \leq p$' (p then is the maximum), or 'Is it inevitably that a location n (or transition $\langle m_1, m_2 \rangle$) is reached and the clock $t \geq p$' (p then is the minimum). Since we consider the starting and ending event of the EPC, the transition $\langle m_1, m_2 \rangle$ of the property corresponds to the transformed ending event.

This method can be generalised to select arbitrary elements (functions and events) as starting and ending points. The use of events has just been described and illustrated in the upper example. When using functions there are two cases: (1) If the function is the starting point then the left transition of the resulting automaton must be selected to synchronise with. (2) Otherwise, if it is the ending point then the right transition must be selected.

4 Additional Features

4.1 Back-transformation of Automata to EPC

Back-transforming the transformed set of automata to an EPC provides a sort of valuation or validation of the transformation and shows the modeller the 'interpretation' of the transformed EPC. He/She can compare the original EPC with the back-transformed EPC to compare the differences between the two models.

Because verification of some property can yield to some counter-example (provided by the verification tool), back-transformation can be used to detect where the EPC has a possible design flaws.

4.2 Actor Separation

We defined a process we called *Actor Separation* that utilises the annotation of EPCs with organisational units to extract EPCs that describe the business process under consideration from the perspective of selected organisational units. The resulting automaton is specific to one organisational unit with all the advantages of simulation and verification.

Actor separation can be used in two ways: (1) It is often interesting to show that some properties are (still) valid for this 'sub-model' of the original EPC. (2) By back-transforming such a local perspective to an EPC one gets a customised view that concerns the organisational unit(s) in focus alone. This tailored EPC can be used as guidance for the responsibility of the actors (persons, offices, etc.) in the entire business process.

5 Related Work

To our knowledge there are only two approaches that use verification to get proven assertions about EPCs.

Van der Aalst [vdA99], defines two properties *regular* and *sound* and claims soundness to be a minimal requirement, because it guarantees that the process chain is free of potential deadlocks and lifelocks. Further properties like *well-structuredness* (that is equivalent to our restriction to regular EPCs without the OR-connector) may be used to indicate design flaws of the EPC.

Rump ([Rum98]) distinguishes between *general properties* and *user defined properties*. Whereas the first aim at the correctness of the EPC, also indicating the origin of the design flaw, the latter allow to define arbitrary assertions that can be expressed using CTL. Provided with operational semantics, the defined EPC is transformed into a reachability graph where general properties concerning the structure of the EPC are verified. User defined properties are expressed using CTL and can be verified by the use of a model checker (for instance SMV).

Commonly used is the transformation to Petri nets (e. g. [CS92], [Rod97] [MR00],[Rit99, Rit00], [Deh02], [LSW98]). Elements and structures of an EPC are transformed into elements and structures of Petri nets.

A problem that arises when transforming EPCs to Petri nets, namely the non-locality of join-connectors, was presented in van der Aalst et al. ([vdADK02]) and was solved by Kindler ([Kin03]). Based on these semantics Cuntz built a tool ([Cun04]) that can be used to simulate EPCs answering the questions of cleanness, contact-freeness and deadlock-freeness.

Two other approaches are situated in the context of workflow management. The first approach transfers EPC to state and activity charts and focuses on the specification and verification of the workflows ([WWD$^+$97]). The second approach ([CKSW01]) transforms *extended* EPCs (eEPC)[13] to a language called *Flow Definition Language*, that is used to specify workflows in the workflow management system *MQSeries Workflow* (IBM).

Operational semantics on EPCs have been defined by Nüttgens and Rump ([NR02]). They use a state based approach with tokens travelling over the state graph.

[13]Extended EPC have various additional modelling elements and are extended by a lot of additional attributes. eEPC are for instance provided by the ARIS Toolset.

6 Conclusion and Future Work

6.1 Conclusion

In this paper we introduced a method for the verification of properties of (Timed) EPC. Properties in this understanding are defined by the modeller and are a kind of quality assurance needed in the real world process.

In a first step we take communicating automata as a target description language. Simple linear chains, but also concurrent and alternative chains are then transformed into communicating automata. Properties on these EPCs can be expressed using a variant of CTL (Computation Tree Logic), a temporal logic language. And the transformation into properties on communicating automata allows the verification of such properties within the framework of EPCs.

Using the framework of *timed automata* the transformation and verification of timed EPCs is realised. The time attributes in the EPC express real-time durations of the modelled system. By an adequate transformation, properties such as maximal overall time and minimal overall time of the process (chain) can be found and/or verified. A timed extension of CTL, namely Timed CTL, is used to express these properties.

The presentation of the principle of the transformation (of EPCs and corresponding properties) is given priority over the completeness of the EPCs transformed.

A tool for the transformation of EPC and a (corresponding) tool for verification of automata are currently being developed. Using this tool the transformation of larger and elaborated examples are possible.

6.2 Future Work

Although, we did not consider all the EPCs currently used in practice, the grammars used allow to define a huge variety of EPCs. Nevertheless, there are obvious extensions that will get the approach closer to practice. For instance the missing OR-connector or process interfaces are elements that were commonly used. Missing structure are multiple starting and ending events, multinary connections and hierarchical EPCs.[14]

Beside these obvious extensions some more visionary extensions are possible. [15]

One idea is to extend EPC by arbitrary resources. We annotate functions with 'effort' instead of 'time' and add some role/position that can be adopted by several (different) persons. Then we can calculate the time necessary to perform the function with respect to the number of persons, as it is practised in project management. Or we can ask for properties like: 'How many persons are necessary to perform the business process in 20 time units?' The extension to continuously changing arbitrary resources means that we

[14]Transforming this elements is work in progress. E. g. OR can easily be transformed in a similar way, using its logical equivalent: x OR $y \equiv (x$ AND $y)$ XOR $(x$ XOR $y)$.

[15]More extensions can be found in [Den06].

need to chose *Hybrid Automata*.[16]

The extension of automata by *Abstract Data Types* introduces the possibility to specify and verify properties regarding the content of information. Abstract Data Types can be understood as a generalisation for *Informational Objects* already used to represent data in EPCs.

References

[BBF$^+$01] Béatrice Bérard, Michel Bidot, Alain Finkel, Francois Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, Berlin, New York, 2001.

[BLL$^+$95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - A Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems*, pages 232–243, 1995.

[CKSW01] David Christensen, Achim Kraiss, Anja Syri, and Gerhard Weikum. Automatische Übersetzung von Geschäftsprozessmodellen in ausführbare Workflows. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung,*, pages 505–513, London, UK, 2001. Springer-Verlag.

[CS92] R. Chen and A.-W. Scheer. Modellierung von Prozeßketten mittels Petri-Netz-Theorie. In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 107. Institut für Wirtschaftsinformatik (IWi), 1992.

[Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master's thesis, Universität Paderborn, Juni 2004.

[Deh02] Juliane Dehnert. Making EPCs fit for Workflow Management. In *Proceedings of the GI-Workshop EPK 2002*, pages 51–69, 2002.

[Den06] Stefan M. R. Denne. Verifying Properties of Event-driven Process Chains by Transformation to Hybrid Automata. Master's thesis, Saarland University, April 2006. Available on request to the author.

[HH95] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology Tool. In *Hybrid Systems II*, number 999 in Lecture Notes in Computer Science, pages 265–293. Berlin, New York, 1995.

[HKS93] W. Hoffmann, J. Kirsch, and A.-W. Scheer. Modellierung mit Ereignisgesteuerten Prozeßketten (Methodenhandbuch). In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 107. Institut für Wirtschaftsinformatik (IWi), Saarbrücken, 1993.

[Kin03] Ekkart Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In Markus Nüttgens and Frank J. Rump, editors, *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 71–80, 2003.

[16]In the example just mentioned we are not forced to switch to hybrid automata, as an extension of timed automata, namely automata with rated clocks can be used. In general such kinds specialised approaches do not succeed.

[KNS92] G. Keller, Markus Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). In *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, number 89. Institut für Wirtschaftsinformatik (IWi), 1992.

[LSW98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In Jörg Desel and M Silva, editors, *Application and Theory of Petri Nets*, pages 286–305, 1998.

[MR00] Daniel Mold and Jörg Rodenhagen. Ereignisgesteuerte Prozeßketten und Petrinetze zur Modellierung von Workflows. In *Visuelle Verhaltensmodellierung verteilter und nebenläufiger Software-Systeme, Proceedings 8. Workshop des Arbeitskreises "Grundlagen objektorientierterProgrammierung" der GI-Fachgruppe*, pages 57–63, 2000.

[NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, number P-21, 2002.

[Rit99] Peter Rittgen. Modified EPCs and Their Formal Semantics. In *Arbeitsberichte des Institutes für Wirtschaftsinformatik*, number 19. Koblenz-Landau, 1999.

[Rit00] Peter Rittgen. Quo vadis EPK in ARIS? In *Wirtschaftsinformatik*, number 42, pages 27–35. 2000.

[Rod97] J Rodenhagen. Darstellung ereignisgesteuerter Prozeßketten (EPK) mit Hilfe von Petrinetzen. Master's thesis, Universität Hamburg, 1997.

[Rum98] Frank J. Rump. *Durchgängiges Management von Geschäftsprozessen auf Basis ereignisgesteuerter Geschäftsprozeßketten*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 1998.

[vdA99] W. M. P. van der Aalst. Formalization and Verification of Event-driven Process Chains. In *Information and Software Technology*, number 41, pages 639–650. 10 1999.

[vdADK02] W. M. P. van der Aalst, Jörg Desel, and Ekkard Kindler. On the semantics of EPCs: A vicious circle. In Markus Nüttgens and Frank J. Rump, editors, *Proceedings of the GI-Workshop EPK*, pages 71–79, 2002.

[WWD$^+$97] Gerhard Weikum, Dirk Wodtke, Angelika Kotz Dittrich, Peter Muth, and Jeanine Weißenfels. Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR. *Inform., Forsch. Entwickl.*, 12(2):61–71, 1997.