

A Formal Approach to Adaptive Processes using Scenario-based Concepts

Dirk Fahland

Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
fahland@informatik.hu-berlin.de

Abstract. The problem and need for adapting business processes and service behavior to cope with changing circumstances is identified well. Standard models for business processes still rely on a fixed process logic, the change of which is rather hard to achieve. Ad-hoc changes to a standard model are usually considered too ‘dangerous’ as they are performed in not well-defined manner. Other models for adaptive processes deviate to some extent from established business process models. This deviation comes at the price of limited understandability and loss in analysis capabilities.

We propose a model for adaptive processes based on Petri nets which have successfully been applied in modeling and analyzing business process and web services. Our operator to adapt the behavior of such models is formalized by the help of scenario-based concepts known from live-sequence charts in purely mathematical terms. This combination of concepts allows to write down the result of the adaptation rather than how adaptation shall be performed.

1 Introduction

The established models of business processes and (web) services assume known and predictable circumstances within which processes and services are executed. An execution is typically based on a fixed set of fully specified workflow processes, manipulating a certain kind of data and involving a known set of resources in a certain organizational structure [16]. Some application domains like disaster management or more flexible kinds of business models violate these assumptions. Established solutions are insufficient in such a situation.

These problems have been researched in the workflow community under the term ‘change in workflow (management)’. Starting with an initially fixed set of workflow processes and resource adjusted for typical circumstances, any violation of the circumstances leads to an observable change in the workflow’s environment that is handled by changing the workflow as well. A workflow that can be changed in this context is called *adaptive*; if this may happen at runtime such that running instances are transferred, the workflow is *dynamic* [13].

The adaptation of a workflow process has to happen in a well-defined manner to guarantee properties like soundness. Most approaches for adaptive workflows

use graph-based models like Petri nets; an adaptation is an operation on the graph structure [8, 1, 3, 4, 12, 7]. But these approaches have two main deficits: They are confined to closed systems without interfaces to their environment. More importantly, these approaches describe *how* the change of the model shall be performed. A business modeler rather describes the result of the change at those parts of the business process he is interested in.

In this paper we present an idea for a formal foundation of business processes that can be adapted at runtime through the notion of behavioral scenarios. We combine a Petri net model for business processes and services with a set of artifacts that describe what the behavior of the changed business process looks like. Each artifact, called *oclet*, provides parts of partial-ordered runs of a Petri net that are annotated using concepts of live-sequence charts [5]. The annotation tells in an intuitive way how the behavior specified in the oclet changes the behavior of the Petri net that models the business process.

The paper is organized as follows. We recall the models and concepts relevant to our approach in Sect. 2. In Sect. 3 we characterize structures and operations that can be used to define adaptive business processes. Our example of Sect. 4 fills this characterization with life and demonstrates our idea. We conclude and mentioned related work in Sect. 5.

2 Preliminaries

2.1 Open Workflow Nets

Business processes are an application domain of workflows. A workflow's process logic that relates tasks, data and resources to each other has successfully been modeled in Petri nets [2]. Communicating business processes that are encapsulated as a service can be conceived as a workflow with an interface [10]. Open workflow nets (oWFN) have been suggested as a quite liberal formal notion of (communicating) business processes [11].

An *open workflow net* N is an ordinary place/transition net [15] (P, T, F, m^0) with two disjoint sets $P_i, P_o \subseteq P$ of *input* and *output* places N , and a set Ω of *final markings* (we conceive a marking as a multi-set here).

Input places have no incoming arc ($\forall p \in P_i : \bullet p = \emptyset$), output places have no outgoing arc ($\forall p \in P_o : p^\bullet = \emptyset$). Together they constitute the interface of N , separated into channels for incoming and outgoing messages, respectively. The restrictions on the pre- and post-sets of the channels guarantee that an incoming message was not generated by N and that an outgoing message is not received by N . The channels are initially empty: $\forall p \in P_i \cup P_o : p \notin m^0$.

The set Ω of final markings characterizes the set of feasible final states of N ; we require that N 's interface is empty in any final marking ($\forall m_f \in \Omega : (P_i \cup P_o) \cap m_f = \emptyset$), and we require that no final marking enables a transition of N .

It is obvious that an open workflow net N with a non-empty interface requires actions from its 'environment' in order to reach a final marking. A typical concept

to realize an environment of N is the composition of open workflow nets $N \oplus M$ that results in a closed system without an interface [11].

In this paper we confine ourselves to a single oWFN N and suggest an alternative approach to formalize the behavior of the environment. Details will be given in our example in Sect. 4

2.2 Occurrence Nets and Branching Processes

Branching processes are a partial-order semantics of Petri nets. Intuitively, branching processes are ‘unfolded’ Petri nets. For a detailed introduction to occurrence nets and branching processes, the reader is referred to [9]; we will recall terminology and notations.

Intuitively, the graph of a transition system can be unfolded to a tree, where each node is path in the transition system and each edge represents the extension of a path by a further step. In a run of a Petri net, a directed acyclic graph unambiguously identifies the firing of a transition or the token on a place. Just like the union of paths yields a tree, a similar notion of union yields a branching process of a Petri net.

We write $x < y$, $x \# y$, and $x || y$ if the nodes x and y of a Petri net are in *causal relation*, *conflict*, and *concurrency relation*, respectively.

An *occurrence net* is an acyclic Petri net $O = (B, E, F)$ where each place has at most one input transition, each net element is finitely preceded and no element is in conflict with itself. A place of an occurrence net is called *condition*, a transition is called *event*. The *minimal elements* $Min(O)$ of an occurrence net have an empty preset. In contrast to [9] we allow that O contains net elements with empty presets that are not in $Min(O)$, e.g. input places of an oWFN; further events may be minimal elements as well.

A *labeled occurrence net* (B, E, F, λ) labels its elements with elements of a Petri net $N = (P, T, W, m^0)$, $\lambda : B \cup E \rightarrow P \cup T$ with $\lambda(B) \subseteq P$ and $\lambda(E) \subseteq T$.

A labeled occurrence net $\beta = (B, E, F, \lambda)$ is called *branching process* if

1. The labeling λ preserves the environment of transitions: for each $e \in E$, $\lambda|_{\bullet e}$ is a bijection between $\bullet e$ in β and $\lambda(\bullet e)$ in N , and similarly for e^\bullet ;
2. β ‘starts’ at m^0 : $\lambda|_{Min(\beta)}$ is a bijection between $Min(\beta)$ and $m^0 \cup \{t \in T \mid \bullet t = \emptyset\}$; and
3. β does not duplicate transitions of N : $\forall e_1, e_2 \in E : \bullet e_1 = \bullet e_2 \wedge \lambda(e_1) = \lambda(e_2) \rightarrow e_1 = e_2$.

A branching process β' is a prefix of a branching process β if β' is a causally closed subnet of β that is complete wrt. input- and output conditions of events in β and contains $Min(\beta)$. A Petri net N has a unique¹ maximal branching process $\beta(N)$ wrt. this prefix relation. This branching process that “unfolds as much as possible” is called *unfolding* of N , and can be conceived as a partial-ordered representation of N ’s entire behavior.

¹ up to renaming of conditions and events

A state of N can be defined as a configuration of $\beta(N)$: a *configuration* $E \subseteq C$ of a branching process is causally closed ($e \in C \rightarrow [e] =_{\text{df}} \{e' \in E \mid e' < e\} \subseteq C$) and conflict-free ($\forall e, e' \in C : \neg e \# e'$). Each finite configuration induces a set of concurrent conditions, called *cut*: $Cut(C) = (Min(\beta) \cup C^\bullet) \setminus \bullet C$. Configuration C can be *extended* by an event $e \in E$ if $\bullet e \subseteq Cut(C)$ and $e \notin C$, the resulting configuration is denoted $C \oplus e$.

A cut corresponds to the marking $Mark(C) = [\lambda(b) \mid b \in Cut(C)]$ that can be reached by ‘firing’ the configuration C . Extending a configuration $C \oplus e$ corresponds to firing $\lambda(e)$ in $Mark(C)$. [9]

2.3 Live-Sequence Charts

Live-sequence charts (LSCs) are an extension of message sequence charts (MSCs) and we will briefly recall its concepts informally. Our approach for adaptive processes makes use of these concepts rather than of the formal definition of LSCs. The interested reader is referred to [5].

As in MSCs, a LSC-chart c specifies the causal ordering of a finite number of events in a concurrent execution of a number of processes. Each event is assigned to a process where it occurs; the events of one process are in total causal ordering ($<$); a message relates events of two processes as sending the message is causally before receiving it. Any two events that are not causally ordered are concurrent. Because an LSC-chart specifies occurrences of events in a single run, there is no notion of conflicting events.

In a run, an instance of an LSC-chart ‘observes’ the occurrence of events it contains. Any event may occur at any time. A new instance of a chart is created as soon as one of its events initial events occurs. An event $e \in c$ is *activated in chart instance* c if all events $e' < e$ in c have already occurred in the run – the control-flow has reached e . If e is activated and occurs, the control-flow of c proceeds beyond e .

An LSC-chart can be separated into a *pre-chart* and a *main-chart*. If the behavior of the pre-chart has been observed completely, the main-chart gets *activated*.

An event $e \in c$ can be *hot* or *cold*. If e is hot and activated, e *must occur* in that run before any other event $e' > e$ occurs; otherwise, that run *violates* the chart instance c . If e is cold, it only may occur, that is if e does not occur, the chart instance is not violated. A pre-chart contains only cold events.

A chart instance *completes* in a run if all of its events have been observed; a violated chart instance cannot complete. A chart instance *holds in a run* if it either is not activated or if it completes. An LSC-chart itself can be *universal*, *existential* or an *anti-chart*. An existential chart has an empty pre-chart.

A LSC specification consists of a finite set of LSC-charts; its specified behavior is the set of runs – (partially ordered) sequences of events – such that: each instance of each universal chart holds in every run, for each existential chart exists a run where an instance holds, and no instance of an anti-chart completes in any run.

3 Open Adaptive Processes

We want to use the principles of occurrence nets and live sequence charts (LSCs) to specify adaptive processes in a well-conceivable manner. On their combination, we have a number of goals in mind:

1. At any given point, there exists an artifact that describes the known and intended behavior of the process.
2. The description of the behavior which is the basis for executing the process is operational.
3. The description of the behavior is formally well-defined.
4. Changes to the description follow from formally well-defined artifacts.
5. The ‘artifacts of change’ themselves shall describe the behavior to be changed rather than a mechanism that implements this behavior.
6. Artifacts that describe change shall denote the reason for change and the effect of change together in an understandable way.
7. Changes to the description of the behavior shall be related back to a partial, observed execution.
8. The description of the behavior allows the occurrence of unexpected actions.

Formalizing Behavior of Adaptive Processes. We want to achieve the goals given above by modifying the complete branching process $\beta = (B, E, F, \lambda)$ of an open Workflow Net N (i.e. its unfolding) in a well-defined way. Since a (complete) branching process describes the behavior of a Petri net operationally, its use satisfies goals 2 and 3. We will define an operator \triangleleft that transforms a branching process on the basis of a formally well-defined artifact. The artifacts we have in mind are parts of a labeled occurrence net, called *oclet*. For now let \mathcal{O} denote set of all oclets, a detailed definition follows. The branching process will be changed by set operations like union, intersection, or subtraction of the branching process with oclets. This satisfies goal 5.

Changing the branching process β will be understood as changing an occurrence net that is labeled with transitions and places of N . The result of the change, say β' , will be a labeled occurrence net which might not be generated by ‘unfolding’ N . But β' describes the partially ordered behavior of some distributed system, say N' . We do not want to consider the implicit description of N' for instance in terms of Petri nets in this paper, or how it can be generated from N – this is future work (see Sect. 5). But we conceive β' as *the* branching process of N' . We will therefore use the term ‘branching process’ in the relaxed meaning of describing the entire behavior of some system at a given point in time.

We now fix the space of behavior that we want to cover with our approach and its fundamental mechanics. The signature of our transformation operator is

$$\triangleleft : (\mathcal{B} \times \mathcal{C}) \times \mathcal{O} \rightarrow (\mathcal{B} \times \mathcal{C}).$$

Let $\langle \beta, C \rangle \in (\mathcal{B} \times \mathcal{C})$ be a state where C is a configuration of β . The *occurrence* of an enabled event $e \in E$ extends the configuration $\langle \beta, C \rangle \xrightarrow{e} \langle \beta, C \oplus \{e\} \rangle$, and

leaves the behavior untouched. The behavior can be *adapted* on the basis of an oclet $o \in \mathcal{O}$, $\langle \beta, C \rangle \xrightarrow{o} (\langle \beta, C \rangle \triangleleft o)$. Occurring events and adaption of behavior span the *behavior space* $\mathcal{S} = (\mathcal{B} \times \mathcal{C}, \xrightarrow{\cdot})$ and cover all system dynamics we are interested in. We therefore limit adaptation to the expressiveness of the, now to be defined, oclets \mathcal{O} and the transformation operator \triangleleft .

Oclet - An Artifact of Change. An oclet $o = (\beta_o, pre, abstr, m, type)$ consists of a labeled occurrence net $\beta_o = (B_o, E_o, F_o, \lambda_o)$ with a causally closed, conflict-free *precondition* $pre \subseteq (B_o \cup E_o)$, $\forall x, y \in pre : [x] \subseteq pre \wedge \neg x \# y$. The oclet may contain *abstract* elements, $abstr \subseteq (B_o \cup E_o)$, $\forall x \in abstr : \lambda_o(x) = \perp$. Non-abstract maximal conditions of pre may be *marked*: $m \subseteq (B_o \cap (pre \setminus abstr))$ and $\forall b \in m : b^\bullet \cap pre = \emptyset$. The entry $type \in Type$ denotes the kind of change to the branching process and shall be left abstract for the moment. Let \mathcal{O} denote the set of all oclets. Let β_o^{pre} denote the restriction of β_o to pre and let β_o^{main} denote the remaining part of β_o . An example o_{ex} is depicted in Fig. 1.

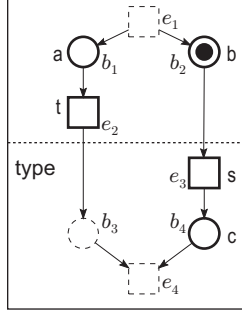


Fig. 1. Example oclet o_{ex} .

The occurrence net β_{ex} of oclet o_{ex} contains four events e_1, \dots, e_4 , two labeled t and s , and two abstract events; it contains four conditions b_1, \dots, b_4 , labeled a, b, c and one being abstract. The precondition of o_{ex} is depicted above the dashed line, comprising e_1, e_2, b_1, b_2 ; condition b_2 is marked.

Characterizing Preconditions. An oclet's precondition denotes what kind of behavior has to be observed in the execution of a branching process in order to activate the adaptation that is specified by the oclet. Abstract events and conditions in the precondition are used to establish the partial order we wish to see. Additionally, the adaptation specified by the oclet might require some structural properties of the branching process to be adapted. These structural properties are expressed by abstract events and conditions in β_o^{main} .

For instance, assume that o_{ex} 's type in Fig. 1 yields that β_{ex}^{main} shall be added to the branching process. However, adding e_3 and b_4 shall happen in such a way

that in the adapted branching process event b_4 has a (transitive) post-event e' with $e_2 \in [e']$.

If the branching process to be adapted does not satisfy the structural requirements of the oclet, the adaptation would not be valid. For such a case we have chosen to define that the ‘adaptation’ does not change the branching process.

Let C be a configuration of β . The oclet o is *activated* in state $\langle \beta, C \rangle \in \mathcal{S}$ iff there exists a place/transition refinement γ of the abstract elements of β_o such that

1. the refined occurrence net of the precondition $\gamma(\beta_o^{pre})$ can be mapped to $Min(\beta) \cup C \cup C^\bullet$ by an injective morphism $\varphi : \gamma(\beta_o^{pre}) \rightarrow \beta$ wrt. λ_o^{pre} and λ , and
2. the marked conditions of o are in the current cut: $\varphi(m) \subseteq Cut(C)$.

Characterizing Adaptation. We will allow changes to β only by applying our transformation operation $\triangleleft : (\mathcal{B} \times \mathcal{C}) \times \mathcal{O} \rightarrow (\mathcal{B} \times \mathcal{C})$. Let C be a configuration of β . We require that $\langle \beta, C \rangle \triangleleft o = \langle \beta, C \rangle$ if o is not activated in $\langle \beta, C \rangle$. This satisfies goals 1 and 4. Since $\langle \beta, C \rangle$ can be changed only if there is an activated oclet o , goal 7 is satisfied.

Those changes on β due to o by \triangleleft that we have in mind may only involve elements of β and o , and must take place beyond the configuration C : For any state $\langle \beta', C' \rangle \in \mathcal{S}$ with $\langle \beta, C \rangle \triangleleft o = \langle \beta', C' \rangle$ we require that either $\langle \beta, C \rangle = \langle \beta', C' \rangle$ or

- there exists a place/transition refinement γ of the abstract elements in o , and
- there exists an injective morphism $\varphi : \gamma(\beta_o) \rightarrow (\beta \cup \beta')$ wrt. $\lambda_o, \lambda, \lambda'$ (the oclet’s causal structure is related to β and β' in some way) such that
 1. $\bullet C' \cup C' = \bullet C \cup C$ (we cannot change the past),
 2. for every $x \in \beta' \setminus \beta$, $\varphi^{-1}(x) \in (\beta_o^{main} \setminus abstr)$ (any new element is caused by a non-abstract element of o), and
 3. for every $x \in \beta \setminus \beta'$ either there exists $y \in \bullet x \cap (\beta \setminus \beta')$, or $\varphi^{-1}(x) \in (\beta_o^{main} \setminus abstr)$ (an element is removed either due to a non-abstract element of o or because one of its causal predecessors was removed – due to o)

Please observe that if the structural requirements of o given by its abstract elements cannot be met by β , this characterization of \triangleleft guarantees that $\langle \beta, C \rangle \triangleleft o = \langle \beta, C \rangle$. Altogether, we achieve goal 6.

Open Adaptive Processes. The concepts we have presented so far satisfy goals 1-7. Before we deal with goal 8 we need to fix what is ‘expected’ and what is ‘unexpected’ in our system.

Our notion $A = (N, \{o_1, \dots, o_n\}, s^0)$ of an adaptive process consists of an oWFN $N = (P, T, F, in, out, m^0)$ and some oclets o_1, \dots, o_n , and is called *open adaptive process*. The set P_A of places of A , consists of the places of N and the

places that are mentioned in the oclets o_i as labels of conditions; similarly for transitions:

$$P_A = P_N \cup \bigcup_{i=1}^n \{p \mid \exists b \in B_{o,i} : \lambda_{o,i}(b) = p\}$$

$$T_A = T_N \cup \bigcup_{i=1}^n \{t \mid \exists e \in E_{o,i} : \lambda_{o,i}(e) = t\}$$

The *initial state* s^0 of A denotes the intended behavior of A and its initial configuration therein: $s^0 = \langle \beta(N), \emptyset \rangle$. The *behavior space of A wrt. a state s* is the subspace $\mathcal{S}[A](s)$ of $\mathcal{S} = (\mathcal{B} \times \mathcal{C}, \rightarrow)$ that contains s and that restricts \rightarrow for oclets to $\{o_1, \dots, o_n\}$. It is easy to see that $\mathcal{S}[A] =_{\text{df}} \mathcal{S}[A](s^0)$ contains the complete behavior of N and any adapted behavior due to o_1, \dots, o_n . Further, $\mathcal{S}[A]$ contains only events and conditions labeled with T_A and P_A , respectively. In this sense, $\mathcal{S}[A]$ describes the *expected behavior* of our system.

Unexpected Events. Following Petri's idea, any change in the system, even unexpected ones, involves an action that, if the change is meant to be recognized, must be observable. Therefore, we consider only those changes that can be reified as the occurrence of an event as in Sect. 2.

An 'unexpected' action in the context of A is not mentioned in N or o_1, \dots, o_n and hence does not appear in $\mathcal{S}[A]$. It requires no pretext on the behavior of A and may occur at any time. But an unexpected action may interfere with A in a way that affects the behavior A . Based on this observation, we can classify 'unexpected' actions as oclets: An oclet $o = (\beta_o, pre, abstr, m, type)$ describes the occurrence of an action a that is *unexpected in A* iff

1. $pre = abstr = m = \emptyset$,
2. $E_o = \{e\}, \lambda_o(e) = a$,
3. $a \notin T_A$, and
4. for all $\langle \beta, C \rangle$ with $\langle \beta, C \rangle \triangleleft o = \langle \beta', C' \rangle$, exists $e' \in E'$ with $\lambda'(e') = a$.²

Let $\mathcal{O}[\bar{A}]$ denote class of all oclets with actions that are unexpected in A ; we say 'unexpected oclet' to simplify the language. This might be an arguable notion of unexpected events in the behavior of A . But we do argue that this notion produces the smallest artifacts describing observable unexpected events: An oclet $o_a \in \mathcal{O}[\bar{A}]$ formalizes the unconditioned occurrence of a single action a which is not known in A and that may or may not interfere with A by having pre- and post-conditions labeled with places of A . Any other artifact of unexpectedness either involves more events or restricts the modes of interference with A .

Let $s = \langle \beta, C \rangle \in \mathcal{S}[A]$ be an expected state of A . Any unexpected oclet $o \in \mathcal{O}[\bar{A}]$ is activated in s and leads to a state $s' = s \triangleleft o$ which is outside the

² Because neither *Type* nor \triangleleft have been defined yet, we cannot provide a structural criterion on oclets that guarantees the extension of a branching process. This is the reason why we resort to a characterization of what the interplay of *Type* and \triangleleft shall accomplish.

expected behavior of A , because $s' = \langle \beta', C' \rangle \notin \mathcal{S}[A]$ contains $e' \in E'$ with $\lambda'(e') = a \notin T_A$.

The *reachable behavior space of A wrt. state s* is the subspace $\mathcal{S}[A + \mathcal{O}[\bar{A}]](s)$ of $\mathcal{S} = (\mathcal{B} \times \mathcal{C}, \rightarrow)$ that contains s and that restricts \rightarrow for oclets to $\{o_1, \dots, o_n\} \cup \mathcal{O}[\bar{A}]$. The expected and unexpected behavior of A is $\mathcal{S}[A + \mathcal{O}[\bar{A}]] =_{\text{df}} \mathcal{S}[A + \mathcal{O}[\bar{A}]](s^0)$. Each reachable state $s \in \mathcal{S}[A + \mathcal{O}[\bar{A}]](s^0)$ induces the subspace of expected behavior $\mathcal{S}[A](s)$.

Executions of Open Adaptive Processes. A *run* ϱ of A is a (possibly infinite) path in $\mathcal{S}[A + \mathcal{O}[\bar{A}]]$ that begins in s^0 . A run $\varrho = \langle s^0, s^1, s^2, \dots \rangle$ could contain cycles of adaptations $s^i \xrightarrow{o^{i+1}} s^{i+1} \dots s^j \xrightarrow{o^{j+1}} s^i$. For any run ϱ , let $\bar{\varrho}$ be the path in ϱ without cycles of adaptations. Whether constructing cycles of adaptations can be avoided depends on the occurrence of unexpected events and how we actually execute A by ‘pushing buttons’.

We suggest a possible notion of execution which shall serve as a model for further discussion.

- As usual, the execution is driven by the occurrence of expected events as they are specified in the branching process β of a state s .
- At any point in time, an unexpected event could become part of the system behavior, formalized as adaptation of β by an oclet $o \in \mathcal{O}[\bar{A}]$, adding an event to β .
- The behavior can expectedly be adapted by an oclet in $\{o_1, \dots, o_n\}$.

Put differently, in any execution the occurrence of an event e is preceded by a possibly empty sequence of expected and unexpected adaptations by oclets $\mathbf{o} = \langle o^1, \dots, o^k \rangle$ with $o^j \in \{o_1, \dots, o_n\} \cup \mathcal{O}[\bar{A}]$. Then an *execution* $\varepsilon = \langle \mathbf{o}^1, e^1, \mathbf{o}^2, e^2, \dots \rangle$ is an alternating sequence of sequences \mathbf{o}^i of oclets and single events e^i that yields a run $\varrho(\varepsilon)$ of A : For all $i = 1, 2, \dots$ with $\mathbf{o}^i = \langle o^{i,1}, \dots, o^{i,k(i)} \rangle$,

$$s^{i-1} \xrightarrow{o^{i,1}} s^{i,1} \xrightarrow{o^{i,2}} s^{i,2} \dots \xrightarrow{o^{i,k(i)}} s^{i,k(i)} \xrightarrow{e^i} s^i,$$

which results in the run

$$\varrho(\varepsilon) = \langle s^0, s^{1,1}, \dots, s^{1,k(1)}, s^1, s^{2,1}, \dots \rangle.$$

3.1 Defining the Transformation Operator and the Oclet Type

In this section we will briefly discuss the remaining free parameters *Type* and \triangleleft of open adaptive processes and their use. To make sense, \triangleleft depends on *Type*. As in the last section, let $o = (\beta_o, pre, abstr, m, type)$ be an oclet. For the moment, we allow an oclet’s type to be *hot, cold, anti* $\in Type$.

1. The *hot* type denotes a mandatory change in the behavior: whenever a *hot* oclet is activated in β , its main occurrence net β_o^{main} *must be added* to the branching process β .

2. A *cold* oclet denotes conditional behavior. If activated, it *may be added* to β .
3. An *anti*-oclet denotes behavior that *must not be executed (completely)*. For Damm and Harel [5], an *anti*-oclet is technically a mandatory scenario that leads to a failure state of the system. We do not have a notion of failure state and suggest the following: for an activated *anti*-oclet, the events of β_o^{main} *must be removed* from the branching process.

A formal definition \triangleleft would require a number of technicalities we do not want to detail out in this paper. But the characterization of \triangleleft together with the characterization of *Type* given above should provide a good understanding of what the semantics are – the example in the next section will help.

4 Explaining Example

We want to use this section to illustrate our ideas for adaptive processes presenting the inevitable travel agency example. The standard process with slight extensions is given as an oWFN model N_{travel} in Fig. 2.

The process connects a customer (input `request`, output `itinerary`) to a hotel booking service (output `hotel`, input `hotel ack`), a flight reservation service (input and output respectively), and a credit card service (output `credit card`). Upon receiving a `request` from the customer, the process invokes the hotel services and awaits its reply, then proceeds likewise with the airline service before charging the credit card and sending the itinerary to the customer. The single final marking of N_{travel} is the place `complete` that must be reached for a successful completion.

The net N_{travel} is isomorphic to its unfolding $\beta(N_{travel})$ which is therefore not shown here. The entire open adaptive process we are considering additionally provides the oclets $\{o_{cust}, o_{hotel}, o_{airline}, o_{credit}, o_{concur}, o_{sync}, o_{twice}, o_{it}\}$:

Oclet o_{cust} in Fig. 3 specifies the behavior of the customer. Without precondition, the customer sends a `request` and expects a causally related itinerary which she receives.

Similarly in Fig. 4, oclet o_{hotel} specifies the behavior of the hotel booking services. After a `hotel` booking request has been sent, the service receives the request and sends a reply `hotel ack` that is going to be correlated to the request. The oclet $o_{airline}$ isomorphic to o_{hotel} and hence not shown here.

The credit card service just awaits the `credit card` message which it consumes, see Fig. 5.

Each of these oclets is *hot* which means that they must be added to $\beta(N_{travel})$ once their preconditions are satisfied. For instance, o_{cust} guarantees that the `?request` event gets enabled: $\langle \beta(N_{travel}), \emptyset \rangle \xrightarrow{o_{cust}} \langle \beta_1, \emptyset \rangle \xrightarrow{!request} \langle \beta_1, \{!request\} \rangle \xrightarrow{?request} \langle \beta_1, \{!request, ?request\} \rangle = \langle \beta_1, C_1 \rangle$. This kind of *hot* oclets is necessary to close the open workflow net and obtain a branching process which we can execute.

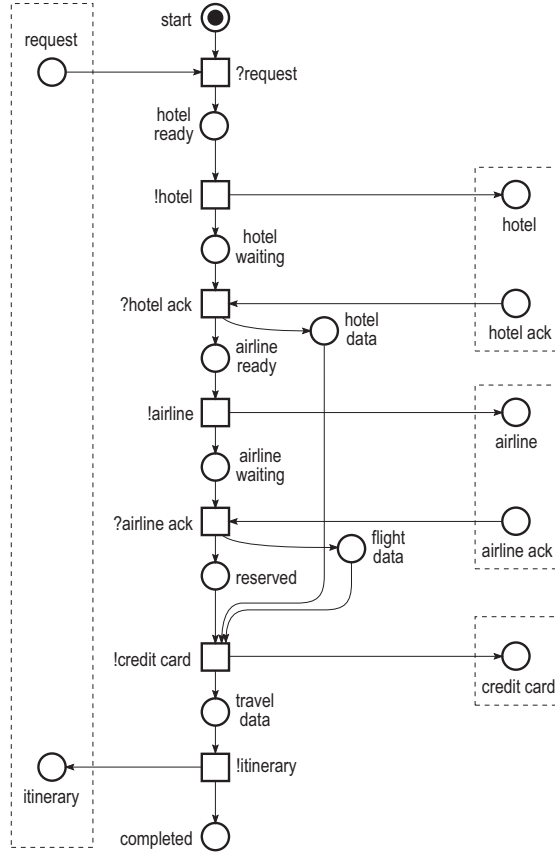


Fig. 2. Travel Process, oWFN N , intended scenario, the final markings are $\Omega = \{[complete]\}$

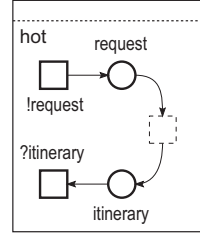


Fig. 3. Oclet o_{cust} specifying the behavior of the customer.

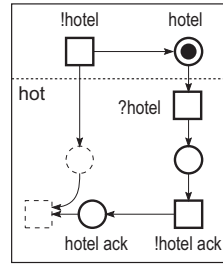


Fig. 4. Oclet o_{hotel} specifying the behavior of the hotel booking partner.

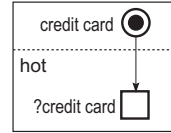


Fig. 5. Oclet o_{credit} specifying the behavior of the credit card partner process.

In $\langle \beta_1, C_1 \rangle$, the **!hotel** event is enabled and C_1 can be extended. Likewise o_{concur} of Fig. 6 is activated because **?request** $\in C_1$. Since o_{concur} is *cold*, we could proceed without adapting the process by o_{concur} and finish the process the way β_1 allows it. Let us follow the more interesting execution and adapt β_1 with o_{concur} . This oclet adds the airline reservation scenario as a concurrent branch to β_1 since **?request** gets a new post-condition **airline ready**. The adaptation

$$\langle \beta_1, C_1 \rangle \xrightarrow{o_{concur}} \langle \beta_2, C_2 \rangle$$

yields a process where $!hotel$ and $!airline$ are enabled concurrently. In the current state $\langle \beta_2, C_2 \rangle = \langle \beta_2, \{!request, ?request\} \rangle$ consider the execution

$$\langle \langle \rangle, !hotel, \langle o_{hotel} \rangle, !airline, \langle o_{airline} \rangle \rangle$$

yielding state $\langle \beta_3, C_3 \rangle$.

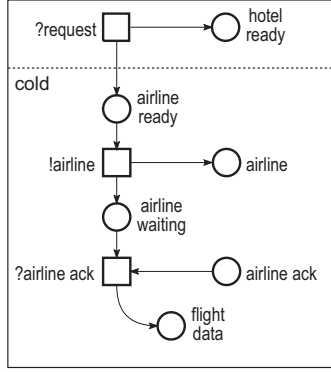


Fig. 6. Oclet o_{concur} specifying the provision of an additional resource for the concurrent booking of flights.

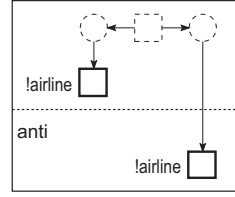


Fig. 7. Oclet o_{twice} specifying that there must not be two concurrent flight bookings.

The adapted process β_3 still contains the flight reservation scenario of N_{travel} that lies causally after $?hotel\ ack$. But we want to book at most one flight. The *anti*-oclet o_{twice} of Fig. 7 disallows a repeated invocation of the flight reservation service in concurrent branches. As soon as $!airline$ has occurred, any further occurrence is disallowed, which means that any other such event must be removed from the branching process.

This is the case in $\langle \beta_3, C_3 \rangle$: $!airline \in C_3$ and the abstract elements of o_{twice} can be refined such that the concurrent $!airline$ after $!hotel$ matches the oclet. Therefore the adaptation $\beta_3 \triangleleft o_{twice}$ removes the latter $!hotel$ and any other causally subsequent event. The result of this step $\langle \beta_3, C_3 \rangle \xrightarrow{o_{twice}} \langle \beta_4, C_3 \rangle$ is depicted in Fig. 8. The grey events constitute the configuration C_3 .

The process β_4 has lost events to invoke the credit card service and notify the customer because of o_{twice} . These events need to occur after hotel and flight have been booked. These concurrent branches can be synchronized by oclet o_{sync} of Fig. 9 upon completion of the booking. The oclet is *hot* and specifies that as soon as hotel booking and flight reservation succeeded ($hotel\ data$ and $flight\ data$ have been reached) and are not billed yet (conditions are in the current cut), the ‘credit card billing and customer notification’ scenario must be possible.

The oclet o_{concur} has not replicated the control-flow path of N_{travel} for the newly added $!airline$ branch. Consequently, o_{sync} cannot synchronize on control-flow tokens. We consider it to be more natural to synchronize on available data

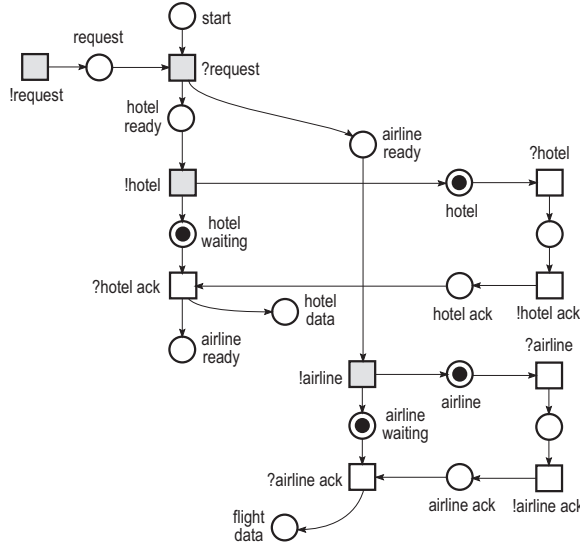


Fig. 8. Branching process β_4 in configuration C_3 .

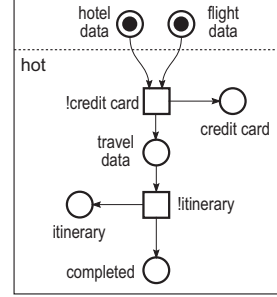


Fig. 9. Oclet o_{sync} specifying that payment and notification proceed after all data has (concurrently) been collected.

as this reflects the way the process would be enacted in the physical world. Control-flow is an artificial concept that helps to bring things in order which are not causally ordered otherwise.

Continuing our run by firing events $?airline, !airline\ ack, ?hairline\ ack, ?hotel, !hotel\ ack, ?hotel\ ack$ yields the state $\langle \beta_4, C_4 \rangle$ where we can adapt by oclet o_{sync} : $\langle \beta_4, C_4 \rangle \xrightarrow{o_{sync}} \langle \beta_5, C_4 \rangle$. Now observe that oclet o_{cust} of Fig. 3 has become activated again after the last adaptation and that $?itinerary \notin \beta_5$ because of o_{twice} . We therefore finally adapt $\langle \beta_5, C_4 \rangle \xrightarrow{o_{cust}} \langle \beta_6, C_4 \rangle$ from where the process can finish its execution (including another adaptation by o_{credit} of Fig. 5).

Unexpected Behavior. A possible unexpected oclet of our process A_{travel} could be that the customer sends an ‘abort’ message. It would be up to our execution how we dealt with such a message. A feasible concept we envision is the run-time extension of A_{travel} with oclets by the user to stabilize the process and take appropriate measures.

For instance, we could introduce an *anti*-oclet that has as prerequisite an *abort* condition and the oclet’s main process covers all message producing events. This oclet would be activated and prevent any outgoing communication which bounds the behavior. The reader may feel free to imagine further oclets for this scenario.

Exception Handling. Expectable fault-events are usually covered in some kind of exception handling mechanism. We could easily extend A_{travel} at design time to cover various faults. For instance o_{hotel} might allow replying a hotel fail

message. To handle this fault several ways of exception handling can be imagined. We could let !hotel occur again or add compensating events.

5 Conclusion

In the preceding sections, we have demonstrated how the concepts of scenario-based specifications and live-sequence charts can be used for defining a formal model for adaptive business processes. Our work provides a first step towards a fully formal model: We characterized the structures and operations that are suitable. In a case study we have shown how the characterization can be filled with life for modeling adaptive business processes in an intuitive way on formal grounds. The key idea for our approach is to provide artifacts that describe the result of the change and not an operation that realizes the change.

Related Work. We are not aware of any work that applies scenario-based concepts to adapt business processes. A large body of research [8, 1, 3, 4, 12] tackles the problem by providing means to modify graph-based models or check the validity of a change. The approach that is closest to ours is [7]. The authors suggest to compose a process out of single scenarios. Each scenario can be transformed by the help of graph-based transformations, hence the overall process can be adapted. In each of these approaches the modeler/user has to find out how to apply the rules to achieve the desired change. Our approach differs in the sense that we specify intended and illegal (partial) runs directly. The business process is then adapted in any situation to realize these runs. Further, our model covers communicating processes and unexpected events which the other works do not.

Future Work. Obviously, the free parameters of our model, the type of the oclets and the adaptation operation which we presented on an intuitive level, need to be chosen and formalized. Having done that, a number of further questions can be studied.

It will be quite important to determine whether the oclets of an open adaptive process are consistent. We will be interested in examining the entire expected behavior of the adaptive process. Further, we might want to construct the Petri net model that exhibits the adapted behavior – here the results of [7] will play a vital role. We are also planning a proof-of-concept implementation of our model that can be applied in case studies, validating our approach.

Our behavioral model allows the classification of expected and unexpected behavior. Immediately, the question arises whether an open adaptive process contains oclets that render a process *self-stabilizing* under any (some) unexpected events. We have taken our model to a point where we can start looking for relations to established results in this area [6].

In some application domains like disaster management, processes involve *human interaction* to a large degree. We envision that, based on an implementation of our model, a process manager observes the process, and adds or modifies oclets

at runtime for handling unpredictably changing situations. This would allow us to put ad-hoc workflows on safe formal grounds.

Besides these new kind of properties in business process models, analyzing standard properties like soundness [2] and controllability [14] in this new setting deserves attention as well.

References

- [1] W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, Feb 2002.
- [2] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [3] Alessandra Agostini and Giorgio De Michelis. Improving flexibility of workflow management systems. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 218–234, London, UK, 2000. Springer-Verlag.
- [4] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Workflow evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.
- [5] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. *Form. Methods Syst. Des.*, 19(1):45–80, 2001.
- [6] Shlomi Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.
- [7] Hartmut Ehrig, Kathrin Hoffmann, and Julia Padberg. Transformations of petri nets. *Electr. Notes Theor. Comput. Sci.*, 148(1):151–172, 2006.
- [8] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *COCS '95: Proceedings of conference on Organizational computing systems*, pages 10–21, New York, NY, USA, 1995. ACM Press.
- [9] Javier Esparza, Stefan Romer, and Walter Vogler. An Improvement of McMillan's Unfolding Algorithm. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 87–106, 1996.
- [10] Axel Martens. *Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services*. WiKu-Verlag, 2004.
- [11] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
- [12] Manfred Reichert and Peter Dadam. Adept_{flex}-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.
- [13] Shazia Wasim Sadiq, Maria E. Orlowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378, 2005.
- [14] Karsten Schmidt. Controllability of Open Workflow Nets. In Jörg Desel and Ulrich Frank, editors, *Enterprise Modelling and Information Systems Architectures*, volume P-75 of *Lecture Notes in Informatics (LNI)*, pages 236–249, Bonn, 2005. (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH.
- [15] Peter H. Starke. *Analyse von Petri-Netz-Modellen*. Stuttgart, Germany: Teubner, 1990. NewsletterInfo: 38.
- [16] WfMC. Terminology and glossary, 3rd edition. Technical report, Workflow Management Coalition, Winchester, 1999.