Oclets – Scenario-Based Modeling with Petri Nets

Dirk Fahland

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

Abstract. We present a novel, operational, formal model for scenario-based modeling with Petri nets. A scenario-based model describes the system behavior in terms of partial runs, called *scenarios*. This paradigm has been formalized in message sequence charts (MSCs) and live sequence charts (LSCs) which are in industrial and academic use. A particular application for scenarios are process models in disaster management where system behavior has to be *adapted* frequently, occasionally at run-time. An *operational semantics* of scenarios would allow to execute and adapt such systems on a formal basis.

In this paper, we present a class of Petri nets for *specifying* and *modeling* systems with scenarios and anti-scenarios. We provide an operational semantics allowing to iteratively construct partially ordered runs that satisfy a given specification. We prove the correctness of our results.

Keywords: scenarios, operational semantics, partial order, Petri nets.

1   Introduction

A recurring application of formal methods is the design, validation, and verification of distributed systems which consist of several interacting processes or components. For this purpose, *scenario-based* methods like *message sequence charts* (MSCs) and *live sequence charts* (LSCs) [1] have become accepted *specification* techniques: The behavior of a system is specified as a set of *scenarios* being self-contained, partial executions. A scenario can be declared as possible, imperative, or forbidden. A formal semantics allows to validate a system's runs against the scenarios following their intuitively understandable meaning [2].

Following the scenario-based paradigm [1,2], we have shown in [3] that in some domains like disaster response system behavior can only faithfully be captured if the *complete* behavior is given by a set of scenarios and anti-scenarios. Assuming completeness and consistency turns a set of scenarios into a system *model*. This particular representation has advantages when *adapting* a given model by adding, removing, or modifying its scenarios without breaking the entire model. In [3], we sketched an approach for this kind of modeling, executing, and adapting systems with scenarios based on Petri nets.

In this paper, we present a complete and consistent formal model of our approach in [3] and explain how a scenario-based specification evolves into a system model within the same formalism.

A major difficulty when using scenarios is the step from a system specification to a system *model* with formal *operational semantics* that provides the set of enabled actions which extend a given run s.t. no scenario is violated. Existing operational models for MSCs and LSCs require a translation into another formalism like automata [4], process algebras [5], or state charts [6], or use formal techniques like graph grammars [7]. This makes operational semantics for scenarios surprisingly technical while scenarios and their composition appear to be very intuitive. In the worst case, the modeler cannot relate the operational model to its original scenarios by mere comparison.

A formal model for scenarios with operational semantics within the same formalism would give a more coherent view on the technique and on system models. A candidate formalism are *Petri nets*: They offer an intuitively understandable notation together with a rigorously defined, simple, and well-understood partial-order semantics [8]. The well-developed Petri net structure theory and available verification techniques could be used for analyzing and verifying behavioral properties of the system [9,10]. Established extensions for Petri nets, like colors or time, could easily be transferred to scenarios. Petri net synthesis techniques could help in translating a scenario-based model into a state-based model.

In this paper, we propose a novel formalization of scenarios based on Petri nets that takes existing results, specifically from LSCs into account. We define a new class of Petri nets called *oclets*. An oclet formalizes a scenario as an acyclic, labeled net, that can be read as a partial, partially ordered run. A prefix of the oclet is denoted as a *precondition* for the scenario which must be observed before the entire scenario can occur. We also define *anti-oclets* to denote partial runs which must not occur completely. A *specification* is a set of oclets and anti-oclets.

We provide a *declarative, formal semantics* of oclets to characterize the satisfying, partially ordered runs of an oclet specification. The semantics allows to check whether a given (Petri net) system satisfies the given scenarios. We complement this semantics with an *operational semantics* that turns a specification into an operational model and allows for directly constructing partially ordered runs from oclets. We show that the operational semantics are equivalent to the declarative semantics under a closed world assumption.

The remainder of this paper is organized as follows. In the next section, we informally introduce the concepts of our approach as we revisit the dining philosophers problem and sketch a solution of the problem with scenarios. We formally define the new Petri net class of oclets in Sect. 3, followed by their declarative semantics in Sect. 4. Section 5 is dedicated to the operational semantics of oclets. We wrap up our approach as we solve the problem of the dining philosophers with oclets in Sect. 6. We compare our approach with related work in Sect. 7 and conclude in Sect. 8.

2   Specifying with Scenarios – An Informal Introduction

Before we begin with formal definitions, we explain the concepts and the underlying intuition of our approach by the help of the well-known *dining philosophers*; see [9] for instance. We first illustrate the philosophers problem on a Petri net model of the system, and then informally sketch a solution with scenarios. We revisit and solve the problem with our model in Section 6. We assume the reader to be familiar with basic notions of Petri nets.

2.1   The Dining Philosophers Problem

The Petri net system (N_{phil}^3, m_{phil}^3) in Fig. 1 models three philosophers each taking his forks at once; this stricter variant will be sufficient to illustrate our ideas.

Each circle $\langle th_i, take_i, eat_i, rel_i \rangle, i = 1, 2, 3$ models the behavior of philosopher i going from thinking to eating and back by taking and releasing his left and right fork f_i and $f_{i\oplus 1}$; by $\oplus$ (and $\ominus$ later on), we denote addition (and subtraction) modulo n. The philosophers synchronize on their shared forks: no two neighboring philosophers may eat at the same time. The system exhibits linear runs like the following: (a) $\langle take_1, rel_1, take_1, rel_1, \ldots \rangle$, (b) $\langle take_1, rel_1, take_2, rel_2, take_1, \ldots \rangle$, and (c) $\langle take_1, rel_1, take_2, rel_2, take_3, rel_3, take_1, \ldots \rangle$.

In (a), phil. 1 always takes both of his forks, none of his neighbors eats. In (b), phil 1 and phil 2 alternatingly eat, alternatingly taking the left and the right fork of phil. 3 who never eats. In (c), each philosopher eats. Runs (a) and (b) are *unfair* as transition $take_3$ gets enabled infinitely often but never fires. These unfair runs are undesired in the system, runs like (c) are desired.

The *dining philosophers problem* is to specify a system that distributedly coordinates the execution of the philosophers s.t. the system contains no deadlocks and no unfair runs. Here, we seek for a stricter solution that has only *decent* runs where

Fig. 1. Petri net model N_{phil}^3 of three dining philosophers with its initial marking m_{phil}^3

a philosopher, after having released his forks, refuses to take them again until each neighbor has taken and released the corresponding shared fork [9]. Runs of kind (c) are decent. Figure 2 depicts a decent, partially ordered run of (N_{phil}^3, m_{phil}^3), corresponding to run (c) above.
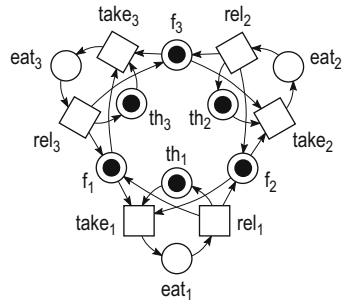
2.2   The Basic Idea

We now want to sketch a solution for the dining philosophers problem with scenarios. Our solution shall have the following properties: (i) A scenario is a well-understandable fragment of a partially ordered run. (ii) System behavior is

composeable from scenarios in an intuitive way. (iii) Anti-scenarios allow speci-
fying forbidden behavior. (iv) Behavioral preconditions of scenarios restrict the
applicability of a scenario to certain situations. (v) The semantics of scenarios
allows testing whether a set of runs satisfies all scenarios. (vi) Finally, satisfy-
ing runs can be constructed from the given scenarios in an operational manner.
We follow this agenda on an informal level in this section and we provide a
corresponding formal model from Sect. 3 onwards.

We begin with the notion of a scenario. In the run π_3 of Fig. 2 we not only find copies of the transitions and places of N^3_{phil}, but even larger, overlapping patterns.

The possibly most obvious pattern, out of which the entire run is composed, is depicted in Fig. 3. It denotes one unrolled execution cycle of philosopher $i \in \{1, 2, 3\}$. The net itself is acyclic but its labels denote that at the end of the execution, the local state $[f_i, th_i, f_{i \oplus 1}]$ is visited again. It specifies a logically self-contained, partial execution of the philosophers system. Such a structure is a *scenario*.

By the symmetry of the philosophers system, *every* partially-ordered run of (N^3_{phil}, m^3_{phil}) consists of overlapping copies of $phil(i), i \in \{1, \ldots, 3\}$ which denote the elementary scenarios of (N^3_{phil}, m^3_{phil}). As we wish to observe these scenarios in the system, we call them *qualified*.

From this observed decomposition, we can infer an appropriate and intuitive *composition* of qualified scenarios: Append a scenario A to another scenario B by merging places at the beginning of A with equally labeled places at the end of B. Likewise, a scenario can be appended to a run: If an initial
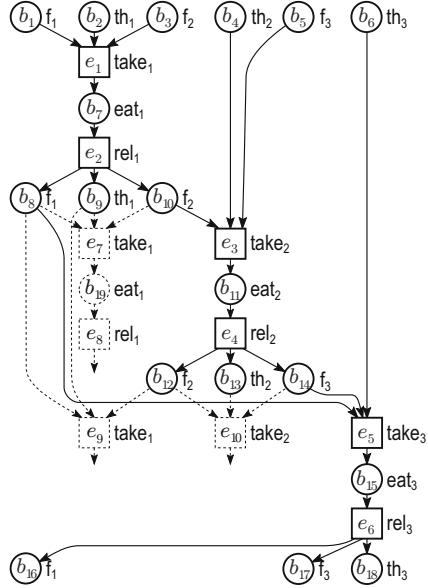

Fig. 2. A decent run π_3 (bold nodes) of the three dining philosophers of Fig. 1
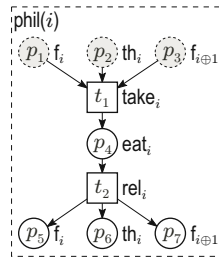
Fig. 3. The elementary behavioral fragment of the philosophers – oclet $phil(i)$

run π_0, consisting of the places $b_1, \ldots, b_6$ of Fig. 2, is given, then π_3 can be constructed by first appending $phil(1)$ followed by $phil(2)$ and $phil(3)$.

This way, we can compose all partially ordered runs of $(N_{\mathsf{phil}}^3, m_{\mathsf{phil}}^3)$, even the non-decent ones. For instance, appending $\mathsf{phil}(1)$ to π_0 followed by $\mathsf{phil}(1)$ again adds transition e_7 (take_1) and subsequent nodes. A run that begins with firing take_2 can be composed by first appending $\mathsf{phil}(2)$ to π_0. Intuitively, all these runs *satisfy* the scenarios $\{\mathsf{phil}(1), \ldots, \mathsf{phil}(3)\}$. In the course of this paper, we generalize this appending composition to overlapping scenarios. In either case, a set of scenarios is meaningful only, if the scenarios share some labels.

2.3 Anti-Scenarios Exclude Behavior

We just sketched how composing qualified scenarios yields partially ordered runs. Although each scenario $\mathsf{phil}(i), i = 1, 2, 3$, is qualified, we can construct undesired, non-decent runs as explained. *Anti-scenarios* are an expressive mean to exclude undesired behavior [6].

The non-decent behavior of the philosophers, as defined in Sect. 2.1, can be narrowed down to the anti-scenarios $\mathsf{decentL}(i)$ and $\mathsf{decentR}(i)$ of Fig. 4. Scenario $\mathsf{decentL}(i)$ denotes that after the left fork f_i was released by philosopher i, it is directly taken again by philosopher i; $\mathsf{decentR}(i)$ respectively for the right fork $\mathsf{f}_{i\oplus1}$. A partially ordered run that completely contains an anti-scenario $\mathsf{decentL}(i)$ or $\mathsf{decentR}(i)$ is not decent; such a run *violates* the anti-scenario. The run consisting of the nodes $\{b_1, \ldots, b_{10}, e_1, e_2, e_7, b_{19}\}$ of Fig. 2 violates $\mathsf{decentL}(1)$.

2.4 Behavioral Preconditions

The previous sections introduced the basic concepts of scenarios and their relation to runs. So far, a scenario can be appended to a run as soon as its beginning can be merged with the run. We now introduce a behavioral *precondition* for scenarios.

The grey shaded (dashed) behavior in Fig. 3, 4, and 5 denotes each scenario's *precondition*. It can be a partial marking as in Fig. 3 or a finite, connected history as in Fig. 4 and 5. All other behavior is the *contribution* of the scenario. The interpretation is that a qualified scenario can extend a given run only, if its precondition is satisfied (has been observed) in the run. Conversely, the run must not continue with the contribution of an anti-scenario, if its precondition has been observed.
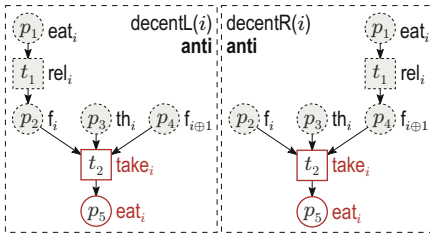

Fig. 4. Anti-oclets $\mathsf{decentL}(i)$ and $\mathsf{decentR}(i)$ specifying the decent use of forks i and $i \oplus 1$
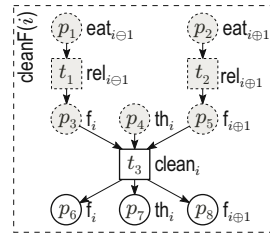
Fig. 5. A qualified scenario with a history-based precondition

Scenario $\mathsf{cleanF}(i)$ in Fig. 5 denotes that phil. i may clean both forks after they have been used and released by his left neighbor $i \ominus 1$ and his right neighbor $i \oplus 1$. The precondition specifies that $\mathsf{clean}(1)$ can be appended to run π_3 of Fig. 2. Directly appending $\mathsf{cleanF}(1)$ in the initial state is not forbidden by qualified scenario $\mathsf{cleanF}(1)$, but such a run cannot be constructed. Thus a qualified scenario reads as "if the precondition holds, the contribution is executable."

2.5   Scenarios Specify Systems, Scenarios Model Systems

Up to now, we only related scenarios to single runs and explained rather vaguely how a set of scenarios relates to the complete behavior of a system. We explain this relation subsequently.

A *specification* is a set of scenarios. In general, a system *satisfies* a specification, if for every prefix of a run π of the system, which allows to append a qualified scenario according to its precondition, the system also has a run π' where the scenario was appended to π. Additionally, no run of the system may violate an anti-scenario of the specification. This definition entails progress for all transitions.

This strict interpretation allows for contradicting specifications: Consider run π_1 that is created by appending $\mathsf{phil}(1)$ to π_0 (consisting of $b_1, \ldots, b_{10}, e_1, e_2$). Qualified scenario $\mathsf{phil}(1)$ requires the presence of a run π_1' constructed by appending $\mathsf{phil}(1)$ again; thus run π_1' contains transition e_7 which violates anti-scenario $\mathsf{decentL}(1)$.

For a more flexible style of scenario-based specifications, we weaken the semantics of qualified scenarios allowing that a qualified scenario is not executed completely if this would violate an anti-scenario. Simply said, we prioritize anti-scenarios over qualified scenarios to solve the contradiction (thus a qualified scenario corresponds to a universal LSC with cold cuts only). With this weaker semantics, a system that executes run π_1, but not π_1', as denoted above, does satisfy the specification $\{\mathsf{phil}(1), \mathsf{decentL}(1)\}$. Apparently, every system satisfying $\bigcup_{i=1}^{3}\{\mathsf{phil}(i), \mathsf{decentL}(i), \mathsf{decentR}(i)\}$ has only decent runs.

In [1], Damm and Harel point out an important issue when interpreting a set of scenarios as there are two principle ways to do so: The *existential* interpretation requires only the possibility to execute the qualified scenarios and forbids anti-scenarios in a system. Any other behavior is allowed. The *universal* interpretation requires that the entire behavior of a system is composed only of the qualified scenarios while disallowing anti-scenarios. Any other behavior that cannot be constructed from the scenarios is forbidden.

A modeler usually begins shaping the system behavior with the existential interpretation in mind. Each new scenario adds a further requirement on the system behavior. Once the scenarios are sufficiently detailed, the modeler changes to the universal interpretation enforcing that the system behaves as specified in the scenarios, only. The universal interpretation turns a set of scenarios into a complete system *model*. It allows to define an operational semantics for scenarios which is not permissible for the existential interpretation.

This concludes the informal introduction of the key concepts for scenario-based specifications and models. In the subsequent sections, we revisit these concepts as we defined the notion of a scenario by generalizing the notion of a local step. This allows us to define an existential, and a universal semantics for scenarios based on Petri nets. The latter constructs runs by appending qualified scenarios while preventing the violation of anti-scenarios as sketched above. We will return to the philosophers example in Sect. 6.

3   Oclets – A Petri Net Model for Scenarios

The next three sections are dedicated to our formal model of scenario-based specifications and their semantics. We begin with structural definitions of the syntax. We assume the reader to be familiar with the basic formal notions of Petri net theory, we recall the most important ones that we need subsequently; for an introduction we refer to [9].

Recalling some basic notions. As usual, we denote a Petri net as $N = (P, T, F)$; we call each place $p \in P$ and each transition $t \in T$ a *node* of N. We will use *labeled* nets, $N = (P, T, F, \ell)$, with a labeling function ℓ assigning each *node* x of N a label $\ell(x)$ from some set $\mathcal{L}$; $\mathcal{L} = \mathcal{T} \uplus \mathcal{P}$ is partitioned into *action labels* $\mathcal{T}$ and *resource labels* $\mathcal{P}$ with $\ell(P) \subseteq \mathcal{P}$ and $\ell(T) \subseteq \mathcal{T}$. We canonically lift any notion on any object to sets and to tuples of these objects.

We write ${}^\bullet x$ for the *preset*, and $x^\bullet$ for the *postset* of a node x of N. A net N is *acyclic* if the flow-relation F has no directed cycles, i.e. the transitive closure of F contains no pair (x, x); we write $\leq_N$ for the reflexive-transitive closure of F. The *minimal* nodes of a set $Y \subseteq P \cup T$ is the set $\min_N Y := \{y \in Y \mid {}^\bullet y \cap Y = \emptyset\}$; *maximal* nodes of Y are $\max_N Y := \{y \in Y \mid y^\bullet \cap Y = \emptyset\}$. The set of transitively reachable predecessors of Y is the set $\lfloor Y \rfloor_N := \{x \mid \exists y \in Y, x \leq_N y\}$; Y is *causally closed* iff $\lfloor Y \rfloor_N \subseteq Y$. The transitively reachable successors are $\lceil Y \rceil_N := \{x \mid \exists y \in Y, y \leq_N x\}$.

A Petri net $\pi = (B, E, F)$ is a *causal net* iff (1) π is acyclic, (2) for each node x of π, $\lfloor \{x\} \rfloor$ is finite, and (3) each place $b \in B$ has at most one pretransition, $|{}^\bullet b| \leq 1$ and at most one posttransition $|b^\bullet| \leq 1$. A labeled causal net that formalizes a *partially ordered run* of a Petri net system as a Petri net again is called *process* (of the system) [8]. We use these three terms synonymously. The net π_3 of Fig. 2 (bold nodes) is a process of $(N^3_{\mathsf{phil}}, m^3_{\mathsf{phil}})$.

The elements of B and E are called *conditions* and *events*, respectively. For the systems considered in this paper, no event of a process has two equally labeled preconditions and no two equally labeled postconditions; further each event of a process has a non-empty preset. The following notions will help us to argue about the structure of processes:

Definition 1 (Induced subnet). *Let $N = (P, T, F, \ell)$ and $M = (P', T', F', \ell')$ be nets. N is a* subnet *of M, $N \subseteq M$, iff $P \subseteq P'$, $T \subseteq T'$, $F \subseteq F'$, and $\ell(x) = \ell'(x)$ for all $x \in P \cup T$. Let $Y \subseteq (P \cup T)$. By $N[Y]$ we denote the Y-induced* subnet *$(P \cap Y, T \cap Y, F|_{(Y \times Y)}, \ell|_Y)$ of N.*

Definition 2 (Complete prefix, ends-with). *A causal net $\pi = (B, E, F)$ is a* prefix *of a causal net $\rho = (B', E', F')$, $\pi \longmapsto \rho$, iff $\pi \subseteq \rho$, $\lfloor B \cup E \rfloor_\rho \subseteq B \cup E$, and $\lceil B \cup E \rceil_\rho = B' \cup E'$.*

Prefix π of ρ is complete *(wrt. postconditions) iff $(e, b) \in F'$ implies $(e, b) \in F$ for each $e \in E$. A set R of causal nets is* prefix-closed *iff each complete prefix of each net of R is also in R. The net ρ* ends with *π, $\rho \longrightarrow\!\!\!| \, \pi$ iff $\pi \subseteq \rho$ and $\max_\pi(B \cup E) \subseteq \max_\rho(B' \cup E')$.*

The structure of scenarios. The aim of our formal model is to describe and construct a system's processes from smaller processes, i.e. the system's scenarios. The simplest kind of scenario is a process given by a single event with its pre- and postconditions; it denotes an *occurrence* of a single transition. We formalize such a scenario as an *atomic oclet*: The event's set of preconditions forms the oclet's *precondition*, the remainder of the process is the oclet's *contribution*. Figure 6 depicts the atomic oclet that denotes the occurrence of transition take$_1$ of Fig. 1.

The theory that we present subsequently generalizes atomic oclets by extending precondition and contribution. In an oclet the occurrence of a transition t can depend on more than its preplaces being marked. Instead, an oclet's precondition can denote a history of transition occurrences which finally produce the tokens on $^\bullet t$. We denote only those predecessors that are *necessary* to fire t, i.e. not all postconditions of

Fig. 6. An atomic oclet

t's predecessors must be included. Figure 5 depicts such an oclet. In the same way, we allow more events and conditions for building larger contributions of an oclet, but here we require each event's pre- and postconditions to be complete; Fig. 3 depicts this case. Altogether this results in the following formal definition of scenarios constituting the class of *oclets*.
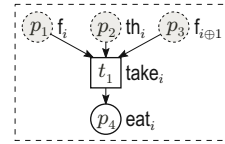
Definition 3 (Oclet). *An* oclet *$o = (P, T, F, \ell, pre)$ is a labeled, finite causal net (P, T, F, ℓ), where each $t \in T$ has no equally labeled preplaces and no equally labeled postplaces, and a* precondition *$pre \subseteq P \cup T$ that induces a complete prefix $o[pre]$ of o s.t. each $x \in \max_o pre$ has a successor in o.*

We call the set $(P \cup T) \setminus pre$ the *contribution* of o, which is non-empty by Def. 3. The nets of the Figures 3, 4, and 5 are oclets. We graphically denote the net structure of an oclet as usual, surrounded by a dashed box; a grey shading (and dashed lines) distinguishes the precondition from the contribution.

By definition, the precondition of every oclet is a complete prefix of the entire oclet, its maximal nodes are places, and each oclet ends with its contribution. Thus, the precondition can be evaluated in a state and the contribution begins with a transition. Further, the precondition is a *history* of the contribution. Otherwise, we would require the contribution to observe behavior on which it does not causally depend.

An *oclet specification* is a set of oclets partitioned into qualified oclets and anti-oclets.

Definition 4 (Oclet specification). *An oclet specification $O = (Q, A)$ consists of two finite, disjoint sets Q and A of oclets where for each $o \in A$ holds $|T_o \setminus pre_o| = 1$. We call Q qualified oclets and A anti-oclets.*

For instance, $\mathsf{Phil}_3^{\mathsf{dec}} := (\{\mathsf{phil}(1), \mathsf{phil}(2), \mathsf{phil}(3)\}, \bigcup_{i=1}^{3}\{\mathsf{decentL}(i), \mathsf{decentR}(i)\})$ is an oclet specification, see Fig. 3 and 4.

For the scope of this paper, we will impose a rather natural consistency condition on an oclet specification O: Let t_1 and t_2 be two distinct transitions from the contributions of two oclets of O. If t_1 and t_2 have the same labels, then for every preplace (postplace) p_1 of t_1 exists an equally labeled preplace (postplace) p_2 of t_2, and vice versa. If this property holds for any two transitions of any two oclets in O, then O is *label-consistent*.

Label-consistency ensures that every two oclet transitions with equal labels specify (maybe different) occurrences of the same "system transition". The specification $\mathsf{Phil}_3^{\mathsf{dec}}$ is label-consistent. We do not impose consistency for transition of the precondition as these do not specify a contribution but an observation of behavior prior to a contribution. Here, partial correspondence is sufficient.

4   Formal Semantics of Oclets

We just defined the syntax of scenarios as oclets. In this section, we will define their semantics in terms of sets of satisfying runs, i.e., labeled causal nets. The semantics of a qualified oclet o shall read as "if pre_o holds, then the entire oclet o can occur". The semantics of an anti-oclet is much simpler: the entire anti-oclet does not occur in any run.

Some useful terminology. The decisive concept to relate an oclet to a run, and hence, to define the semantics of oclets is what we call an *embedding*. An oclet can occur at several places in a run, that is, a run can have several subnets that are isomorphic to an oclet. For clearly distinguishing between an oclet and its occurrences, we say that an oclet is *embedded* in a run if the run contains a subnet that is isomorphic to the oclet; the corresponding subnet isomorphism is an embedding of the oclet into the run. For technical reasons, we formally define these terms for induced subnets.

Definition 5 (Embedding). *Let N and M bet two labeled Petri nets, let $X_N \subseteq P_N \cup T_N$ and $X_M \subseteq P_M \cup T_M$. A mapping $\alpha : X_N \to X_M$ is an embedding of $N[X_N]$ in $M[X_M]$ iff for each node $x \in X_N$ holds $\ell_N(x) = \ell_M(\alpha(x))$ and for each edge $(x_1, x_2) \in F_N$ exists an edge $(\alpha(x_1), \alpha(x_2)) \in F_M$.*

As an example, consider oclet $\mathsf{phil}(2)$ of Fig. 3 and the process π_3 of Fig. 2. The mapping α_1 with $\alpha_1 = [p_1 \mapsto b_{10}, p_2 \mapsto b_4, p_3 \mapsto b_5]$ is an embedding of $pre_{\mathsf{phil}(2)}$ into π_3. To simplify notation, when referring to an induced subgraph, e.g. $\mathsf{phil}(2)[pre_{\mathsf{phil}(2)}]$, we only write its inducing nodes, e.g. $pre_{\mathsf{phil}(2)}$, if the net is obvious from the context and confusion is safely avoided.

Our next step will be to relate a run to a set of runs which we call its *continuations*. This effectively means that the run gets various new labeled events,

conditions, and arcs. We want to distinguish all these continuations only up to *isomorphism*. That means, in the remainder of the paper, we will treat isomorphic Petri nets as equal, specifically regarding containment in sets, etc. This treatment comes natural if we bear the graphical interpretation of nets in mind. It has been shown earlier, e.g. in [10], how this treatment of isomorphic nets can be reduced to strict mathematical identity by choosing canonic identities for nodes of nets.

Continuing a run with an oclet. We now have all notions to formalize the semantics of an oclet. We first define how a prefix of one oclet relates to one run and how this run can be continued with the oclet. We then lift this notion to a set of runs that satisfies one oclet and finally define the semantics of an oclet specification, i.e. sets of oclets.

We introduce some notation for describing where (a prefix of) an oclet is embedded in a run. Let o be an oclet and let $X \subseteq (P_o \cup T_o)$ be the nodes of a complete prefix $o[X]$ of o; let π be a labeled causal net.

1. The prefix $o[X]$ *holds at the end* of π by embedding α, denoted $(\pi, \alpha) \models o[X]$, iff α embeds $o[X]$ at the end of π, formally $\pi \rightarrowtail \alpha(o[X])$, see Def. 2.
2. The prefix $o[X]$ *holds in* (the past of) π by α, denoted $(\pi, \alpha) \models \diamond\, o[X]$ iff α embeds $o[X]$ in π.
3. We write $\pi \models \varphi$ for $\exists \alpha : (\pi, \alpha) \models \varphi$ and $\pi \models \neg \varphi$ for $\neg \exists \alpha : (\pi, \alpha) \models \varphi$.

For instance, $\pi \models \neg \diamond\, o[X]$ expresses that $o[X]$ does not hold anywhere in π. Although our notation takes inspiration from temporal logic, we do not build such a logic here. Nevertheless, it is easy to prove that $(\pi, \alpha) \models \diamond\, o[X]$ holds iff there exists a prefix π' of π with $(\pi', \alpha) \models o[X]$. In our example, the precondition of phil(1) of Fig. 3 holds at the end of run π_3 in Fig. 2 while the precondition of decentL(1) of Fig. 4 does not hold in π_3.

A process in which the precondition of a qualified oclet holds naturally suggests to *continue* this run by appending the complete oclet at its end. We are not only interested in this largest continuation but also in all intermediate continuations. Of course, a run is a complete prefix of each of its continuations.

Definition 6 (Continuation). *Let π, π' be labeled causal nets, let o be an oclet. π' is a continuation of π with o, $\pi \xrightarrow{o} \pi'$ iff π is a prefix of π' and there exists a complete prefix $o[X']$ of o with $pre_o \subseteq X' \subseteq P_o \cup T_o$ and embeddings α and α' with $(\pi, \alpha) \models o[pre_o]$, $(\pi', \alpha') \models o[X']$ s.t. all new nodes come from X' only: $\alpha'|_{pre_o} = \alpha$ and $\alpha'(X' \setminus pre_o) = (B' \cup E') \setminus (B \cup E)$.*

The set of prefixes of o induces the set of continuations of π. Thus, a continuation of π appends some nodes of o s.t. a larger prefix of o holds. In our example, consider the run π_0 (consisting of $b_1, \ldots, b_6$) and the run π_1 (consisting of $b_1, \ldots, b_{10}, e_1, e_2$) in Fig. 2. Run π_1 is the largest continuation of π_0 with oclet phil(1).

A set of runs R is *closed under continuations* with o iff for each $\pi \in R$, each continuation π', $\pi \xrightarrow{o} \pi'$, is a run in R.

Definition 7 (Semantics of an oclet). *A set of labeled causal nets R satisfies an oclet o, $R \models o$ iff R is prefix-closed and closed under continuations with o. R satisfies* the *negation of o, $R \models \neg o$ iff R is prefix-closed and o does not hold in any run in R: $\forall \pi \in R : \pi \models \neg \diamondsuit\, o$.*

A strict interpretation of an oclet specification $O = (Q, A)$ would be $R \models O$ iff $R \models o$ for all $o \in Q$, and $R \models \neg o$ for all $o \in A$. This would allow for contradicting specifications with no satisfying run as explained in Sect. 2.5.

Existential semantics of an oclet specification. We motivated in Sect. 2.5, that we are interested in a weaker semantics of an oclet specification that requires the satisfaction of a qualified oclet in a run only up to the point where an anti-oclet would be violated.

To achieve this, we cannot require that a set of runs is closed under all continuations; we have to exclude those continuations that would violate an anti-oclet. The formalization is straight forward: Let π be a run, let o be a qualified oclet, and let o' be an anti-oclet. A continuation $\pi \xrightarrow{o} \pi'$ does not violate o' iff $\pi' \models \neg \diamondsuit\, o'$ holds. For a second anti-oclet o'', the continuation $\pi \xrightarrow{o} \pi'$ also does not violate o'' iff $\pi' \models \neg \diamondsuit\, o'$ and $\pi' \models \neg \diamondsuit\, o''$ holds. Thus we can generalize this notion of non-violating continuations to a set of anti-oclets.

Definition 8 (Non-violating continuation). *Let o be an oclet and let A be a set of anti-oclets. Let π, π' be labeled causal nets; π' is a* non-violating *continuation of π with o wrt. A iff $\pi \xrightarrow{o} \pi' \wedge \forall o' \in A : \pi' \models \neg \diamondsuit\, o'$. We write $\pi \xrightarrow{o \wedge \neg A} \pi'$ in this case.*

A non-violating continuation wrt. A does not embed any anti-oclet $o \in A$. Because we exclude only those continuations that do violate an anti-oclet in A, the set of non-violating continuations is maximal. We may now close a set of processes in the right way by only considering the non-violating continuations.

A set of runs R is *closed under non-violating continuations* with o wrt. A iff for each $\pi \in R$ each non-violating continuation of π with o wrt. A is a run in R. With this notion, lifting semantics of an oclet to a set of oclets yields the formal *existential semantics* of oclet specifications.

Definition 9 (Semantics of an oclet wrt. anti-oclets). *A set of labeled causal nets R satisfies an oclet o wrt. a set of anti-oclets A, $R \models (o \wedge \neg A)$, iff R is prefix-closed and closed under non-violating continuations with o wrt. A.*

Definition 10 (Existential semantics). *A set of labeled causal nets R satisfies an oclet specification (Q, A), $R \models (Q, A)$ iff $R \models (o \wedge \neg A)$ for each $o \in Q$ and $R \models \neg o$ for each $o \in A$.*

With Def. 10, we can use qualified oclets and anti-oclets to formally specify the behavior of systems. A model, say a Petri net system, satisfies an oclet specification if the system's processes satisfy the intuitively understandable meaning of "if the precondition holds, the entire scenario must be executable as long as no anti-scenario is violated". In any other respect, the behavior of the system can be arbitrary.

5   Operational Semantics of Oclets

In the previous section, we established the existential, formal semantics of oclets for specifying systems. In this section, we turn an oclet specification into an *oclet system* having only the behavior that is specified by its oclets.

We define the *universal* semantics of oclets which is the behavior that can be constructed from a given set of oclets only. The universal semantics is operational as it defines exactly the actions that extend a given run. Such a semantics needs a specific point to begin with the construction; we define an oclet system.

Definition 11 (Oclet system). *Let O be a label-consistent oclet specification, and π_0 be a process. Then $\Omega = (O, \pi_0)$ is an oclet system.*

Like for oclet specifications, we require that equally labeled transitions have equally label preplaces and equally labeled postplaces; see Sect. 3. For example, oclet specification $(\bigcup_{i=1}^{3}\{\mathsf{phil}(i)\}, \bigcup_{i=1}^{3}\{\mathsf{decentL}(i), \mathsf{decentR}(i)\}) =: \mathsf{Phil}_3^{\mathsf{dec}}$ yields the oclet system $\Omega_3^{\mathsf{dec}} = (\mathsf{Phil}_3^{\mathsf{dec}}, \pi_0)$ with π_0 having only the conditions $B_{\pi_0} := \{b_1, \ldots, b_6\}$ of Fig. 2; π_0 is called *initial* process.

Considering transition t_1 of $\mathsf{phil}(1)$, we see that the strict past of t_1, i.e. $\lfloor {}^\bullet t_1 \rfloor$, can be embedded in π_0 while $\lfloor t_1 \rfloor$ cannot be embedded. That is $\pi_0 \models \lfloor {}^\bullet t_1 \rfloor$. Simply said, t_1 is *enabled* in π_0. Transition t_2 of $\mathsf{decentL}(1)$ is not enabled in π_0 because there is no embedding of the entire $\lfloor {}^\bullet t_2 \rfloor$ at the end of π_0.

Definition 12 (Enabled transition). *Let o be an oclet, let $t \in T_o \setminus pre_o$, and let π be a labeled causal net. Transition t is* enabled *in π iff $\pi \models \lfloor {}^\bullet t \rfloor$.*

This definition of enabling a transition generalizes the definition for classical nets: In order to embed the preset of a transition in a process, all its predecessors must be embeddable. Thus in order to enable a transition of an oclet, the transition's history must have occurred. Transition t_2 of $\mathsf{decentL}(1)$ is enabled in π_1 (having nodes $\{b_1, \ldots, b_{10}, e_1, e_2\}$); the corresponding embedding α_2 yields $\alpha_2({}^\bullet t_2) = \{b_8, b_9, b_{10}\}$.

Intuitively, *firing* an enabled transition t means to continue a process π with transition t and its postset. We formalize this pattern as an *extension* of π: It consists of a new event e_t that consumes from those conditions of π that correspond to t's preplaces and produces on new conditions that correspond to t's postplaces.

Definition 13 (Extension). *Let t be a transition of an oclet o, $t \in T_o \setminus pre_o$ that is enabled in π by α: $(\pi, \alpha) \models \lfloor {}^\bullet t \rfloor$. An* extension *of π by t at α is a net fragment $E_t^\alpha := (B_t, \{e_t\}, F_t^\alpha, \ell_t)$ with $B_t = \{b_p^* \mid p \in t^\bullet\}$, $B_t \cap B_\pi = \emptyset$, $e_t \notin E_\pi$,*

- $F_t^\alpha = \{(b, e_t) \mid b \in \alpha({}^\bullet t)\} \cup \{(e_t, b_p^*) \mid b_p^* \in B_t\}$, *and*
- $\ell_t(e_t) = \ell_o(t)$ *and* $\ell_t(b_p^*) = \ell_o(p)$ *for all $p \in t^\bullet$.*

In our example, the extension that corresponds to t_1 (take_1) of $\mathsf{phil}(1)$ in π_0 consists of nodes e_1 and b_7 and all incoming arcs, especially (b_1, e_1), etc. in Fig. 2. An extension is not a Petri net, because its arcs refer to nodes that are

not part of the extension. We therefore call it a net fragment. To fire a transition in a process, append the corresponding extension to the process.

In our operational semantics, firing a transition must not violate an anti-oclet. Observe that transition t_1 of phil(1) is enabled in π_1 as well; the corresponding embedding α_3 yields $\alpha_3(^\bullet t_1) = \{b_8, b_9, b_{10}\} = \alpha_2(^\bullet t_2)$ where t_2 is the contributing transition of anti-oclet decentL(1). Both transitions have the same label; firing t_1 of phil(1) in π_1 would violate decentL(1).

In general, a transition t *would violate* an anti-oclet o_a in a process π iff t is enabled in π by α and o_a has an equally-labeled transition $s \in (T_a \setminus pre_a)$ that is enabled in π by α_a s.t. t and s denote the same occurrence: $\alpha(^\bullet t) = \alpha_a(^\bullet s)$. Firing a violating transition is forbidden. This interpretation yields the *processes* of an oclet system.

Definition 14 (Processes of an oclet system). *Let $\Omega = ((Q, A), \pi_0)$ be an oclet system. The set $Proc(\Omega)$ of all* processes of Ω *is the least set that satisfies:*

1. *π_0 is a process of Ω iff $\pi_0 \models \neg o_a$ for all $o_a \in A$.*
2. *Let $\pi \in Proc(\Omega)$. Let $o \in Q$ and let $t \in (T_o \setminus pre_o)$ be a transition that is enabled in π and that would not violate any $o_a \in A$.*
   The net $(\pi \oplus E_t^\alpha) := (B_\pi \cup B_t, E_\pi \cup \{e_t\}, F_\pi \cup F_t^\alpha, \ell_\pi \cup \ell_t)$ is a process of Ω.

Definition 14 completes the formal semantics of oclets. The entire process π_3 of Fig. 2 is a process of Ω_3^{dec}.

In the remainder of this section, we show that these definitions make sense. We prove that oclet systems are at least as expressive as elementary net systems. We finally show that existential and universal oclet semantics are consistent: the processes of an oclet system (universal semantics) satisfy its own specification (existential semantics). But first of all we show that all processes of an oclet system are labeled causal nets, i.e. that Def. 14 is formally sound.

Lemma 1. *Let Ω be an oclet system. If $\pi \in Proc(\Omega)$ is a process, and E_t^α is an extension of π, then $\pi \oplus E_t^\alpha \in Proc(\Omega)$ is a labeled causal net.*

Proof. Let Ω, π, and E_t^α be as assumed. Let o be the oclet of Ω with $t \in T_o$.

$\rho := \pi \oplus E_t^\alpha$ is a Petri net: π is a net. $B_\rho \cap E_\rho = (B_\pi \cup B_t) \cap (E_\pi \cup \{e_t\}) = \emptyset$ by Def. 13 and 14. It is easy to see that extending F_π with F_t^α preserves the bipartite structure of nets and that the union of ℓ_π and ℓ_t is well-defined.

ρ is a causal net: (1) π and E_t^α are acylic and disjoint. Further, E_t^α has no arc (x, y) with y in π. Thus there exists no arc from E_t^α into π to close a cycle in ρ. (2) In $\pi \oplus E_t^\alpha$, each node has only finitely many predecessors because π is a causal net and E_t^α is finite. (3) Transition t is enabled in π (by Def. 13) which implies $\alpha(^\bullet t) \subseteq \max_\pi(B_\pi \cup E_\pi)$ (by Def. 12). Thus each $b \in \alpha(^\bullet t)$ has no successor in π. In ρ, each $b \in \alpha(^\bullet t)$ has only one successor: e_t (by Def. 13 and 14). The new postconditions $B_t \subseteq \max_\rho(E_\rho \cup B_\rho)$ have no successor and only one predecessor: e_t (by Def. 13). Thus π is a labeled causal net as defined in Sect. 3.   □

The operational semantics of oclets is complete wrt. the partial order semantics of Petri net systems.

Theorem 1. *Let $N = (P, T, F, m_0)$ be a Petri net system with initial marking m_0. Then there exists an oclet system Ω_N s.t. the set of processes of N and the set of processes of Ω_N are equal.*

Proof. We show completeness by defining an algorithm that constructs for each N an oclet system Ω_N that has exactly the same processes as N.

Let $t \in T$. We define an atomic oclet o_t that specifies the firing of transition t: $o_t := (P_o, T_o, F_o, \ell_o, pre_o)$ with $P_o = {}^\bullet t \cup t^\bullet$, $T_o = \{t\}$, F_o the restriction of F_N to $(P_o \times T_o) \cup (T_o \times P_o)$, ℓ_o the identity on $P_o \cup T_o$, and $pre_o = {}^\bullet t$; c.f. Fig. 6. The structure o_t is an oclet by Def. 3. Define $\Omega_N := ((\{o_t \mid t \in T\}, \emptyset), \pi_0)$ with initial process π_0 consisting only of the set of conditions $B_{\pi_0} := \{b_1, \ldots, b_k \mid \exists p \in P_N : m_0(p) = k, \ell_{\pi_0}(b_i) = p, i = 1, \ldots, k\}$; Ω_N is an oclet system by Def. 14.

We prove the equivalence of processes by induction on the number of events in them. Firstly, π_0 is the initial process of N iff it is the initial process of Ω_N, which holds by construction. Let π be a labeled causal net containing n events. By inductive assumption, π is a process of N iff it is a process of Ω_N.

By definition of partial order semantics of nets, π reaches the marking m with $m(p) = |\{b \in \max \pi \mid \ell_\pi(x) = p\}|$ for each place p. Let T_m be the set of transitions that are enabled in m. By construction of Ω_N holds $t \in T_m$ iff there exists oclet o_t in Ω_N with transition t that is enabled in π according to Def. 12.

If $T_m = \emptyset$, π cannot be extended by N and by Ω_N. Otherwise, let $t \in T_m$ with ${}^\bullet t = \{p_1, \ldots, p_k\}$ and $t^\bullet = \{q_1, \ldots, q_l\}$; firing t according to the Petri net semantics constructs process ρ by adding a new t-labeled event e with preconditions ${}^\bullet e = \{b_1, \ldots, b_k\} \subseteq \max_\pi(B_\pi \cup E_\pi)$ with $\ell_\rho(b_i) = p_i, i = 1, \ldots, k$ and new postconditions $e^\bullet = \{b_1^*, \ldots, b_l^*\}$ with $\ell_\rho(b_i^*) = q_i, i = 1, \ldots, l$. Because Ω_N has no anti-oclets, there exists an embedding α with $\pi \oplus E_t^\alpha \in Proc(\Omega)$ (Def. 14). By definition of o_t and by Def. 13 holds $\pi \oplus E_t^\alpha = \rho$. Thus ρ, with $n+1$ events, is a process of N iff it is a process of Ω_N. $\square$

Theorem 1 relates oclet systems to classical Petri net systems. The following theorem relates the universal semantics of oclet systems to the existential semantics of oclet specifications.

Theorem 2. *Let $\Omega = (O, \pi_0)$ be an oclet system. $Proc(\Omega) \models O$.*

We prove Thm. 2 by the help of two lemmata. The first technical lemma states that every complete prefix of a continuation is a continuation as well. The second lemma proves that the processes of an oclet system are closed under non-violating continuations.

Lemma 2. *Let π be a process, let o be an oclet, and let $\pi \xrightarrow{o} \pi_2$. Let π_1 be a complete prefix of π_2, i.e. π_1 contains all postconditions of its events (Def. 2), s.t. π is a prefix of π_1. Then $\pi \xrightarrow{o} \pi_1$.*

Proof. We construct the prefix $o[X_1]$ of o that is embedded at the end of π_1 according to Def. 6. The nodes $X_\Delta := (B_2 \cup E_2) \setminus (B_1 \cup E_1)$ are not in π_1.

Because $\pi \xrightarrow{o} \pi_2$, there exists a prefix $o[X_2]$ of o with $pre_o \subseteq X_2$ and embeddings α and α_2 with $(\pi, \alpha) \models o[pre_o]$ and $(\pi_2, \alpha_2) \models o[X_2]$; see Def. 6.

By Def. 5, α_2 is injective. The nodes $X_1 := X_2 \setminus \alpha_2^{-1}(X_\Delta)$ are all oclet nodes that are embedded into π_1: Because π_1 is a complete prefix of π_2 and by α_2 being injective follows $o[X_1]$ is a complete prefix of $o[X_2]$. The restricted embedding $\alpha_1 := \alpha_2|_{X_1}$ embeds $o[X_1]$ at the end of π_1: $(\pi_1, \alpha_1) \models o[X_1]$.

From π being prefix of π_1 follows $(B \cup E) \cap X_\Delta = \emptyset$. Thus $\alpha(pre_o) \cap X_\Delta = \emptyset$ holds, which implies $pre_o \cap \alpha_2^{-1}(X_\Delta) = \emptyset$. Hence $pre_o \subseteq X_2$ and $X_1 = X_2 \setminus \alpha_2^{-1}(X_\Delta)$ imply $pre_o \subseteq X_1$. Thus $\pi \xrightarrow{o} \pi_1$ by Def. 6. $\square$

Lemma 3. *Let $\Omega = ((Q, A), \pi_0)$ be an oclet system. Let $o \in Q$ and let $\pi \in Proc(\Omega)$. Then every process π_2 with $\pi \xrightarrow{o \wedge \neg A} \pi_2$ is a process of Ω.*

Proof. We prove the property by induction on the number n of new events in π_2, $n = |E_{\pi_2} \setminus E_\pi|$. For $n = 0$ we have $\pi_2 = \pi$ by Def. 6; thus $\pi_2 \in Proc(\Omega)$.

Consider $n > 0$: Let π_2 be a non-violating continuation of π; π_2 contains a new event $e \in (E_{\pi_2} \setminus E_\pi)$ that has no successor event, i.e. there ex. no $e' \in E_{\pi_2}, e \neq e'$ with $e \leq_{\pi_2} e'$. Let π_1 be the prefix of π_2 which we obtain by removing e and $e^\bullet$ from π_2. Effectively, we remove events $E^* = \{e\}$, conditions $B^* = e^\bullet$, and arcs $F^* = \{(b, e) \mid b \in {}^\bullet e\} \cup \{(e, b) \mid b \in B^*\}$.

Because $e \notin E_\pi$, π is a prefix of π_1. From Lemma 2 follows that π_1 is a continuation of π. Trivially, π_1 does not violate any anti-oclet of A because π_2 does not. Thus by inductive assumption, $\pi_1 \in Proc(\Omega)$.

We have to show that $\pi_2 \in Proc(\Omega)$. From the definition of continuation (Def. 6) follows that some prefix $o[X_2]$ of o is embedded at the end of π_2 by some embedding α_2. Thus there exists a transition $t \in X_2$ with $\alpha_2(t) = e$ which we removed from π_2 to obtain π_1. We fire t in π_1 to construct π_2:

Because e and $e^\bullet$ were removed, the prefix $o[X_1]$, $X_1 := X_2 \setminus (t \cup t^\bullet)$ of o is embedded at the end of π_1 by $\alpha_1 := \alpha_2|_{X_1}$. Then $(\pi_1, \alpha_1) \models \lfloor {}^\bullet t \rfloor$, i.e. t is enabled in π_1 (Def. 12). From Def. 13 follows that $E_t^{\alpha_1} = (B^*, E^*, F^*, \ell_{\pi_2}|_{B^* \cup E^*})$ as removed from π_2. Thus $\pi_1 \oplus E_t^{\alpha_1} = \pi_2$. By Def. 14, π_2 is a process of Ω. $\square$

With Lem. 3 we have proven that the processes an oclet system are closed under non-violating continuations. The proof also shows how declarative and operational semantics are related to each other by the notion of local steps. The proof of Thm. 2 is now straight forward.

Proof (of Thm. 2). Let $\Omega = (O, \pi_0)$ be an oclet system with $O = (Q, A)$. We have to show $Proc(\Omega) \models O$ according to Def. 10. The set $Proc(\Omega)$ is prefix-closed by construction. From Lemma 3 follows that $Proc(\Omega)$ is closed under non-violating continuations with any qualified oclet of Ω wrt. A. Thus $Proc(\Omega) \models (o \wedge \neg A)$ for each $o \in Q$.

It remains to show that no process of Ω violates any anti-oclet $o_v \in A$: Assume there is a run $\pi_v \in Proc(\Omega), \pi_v \neq \pi_0$ and an embedding α_v that violates o_v: $(\pi_v, \alpha_v) \models \Diamond o_v$. Because $Proc(\Omega)$ is prefix-closed, we may assume that $(\pi_v, \alpha_v) \models o_v$ holds. From Def. 14 follows that there exists $\pi \in Proc(\Omega)$, an oclet $o \in Q$, and a transition t of o that is enabled in π by an embedding α with $\pi_v = \pi \oplus E_t^\alpha$. But then, o_v contains transition s with $\ell_{o_v}(s) = \ell_o(t)$ and $\alpha_v({}^\bullet s) = \alpha({}^\bullet t)$. Hence t would violate o_v. This contradicts $\pi_v = \pi \oplus E_t^\alpha \in Proc(\Omega)$ by Def. 14. Thus $Proc(\Omega) \models \neg o$ for each $o \in A$. $\square$

We just have shown that the universal semantics of oclet systems imply the existential semantics of oclet specifications. The semantics are not equivalent in general, as any set of processes, that contain labels that do not occur in the specification, cannot be constructed with the universal semantics.

The behavior that satisfies an oclet specification (Q, A) but that cannot be constructed by an oclet system $\Omega = ((Q, A), \pi_0)$ violates the following *closed-world assumption* of the universal semantics. A set of runs R is *closed* wrt. Ω iff for all $\pi \in R$, π_0 is a complete prefix of π and for each node $x \in (B_\pi \cup E_\pi)$ exists a qualified oclet o of O that contributes this node to π, i.e. there exists $y \in P_o \cup T_o$ and embedding $\alpha : \lfloor y \rfloor \to (B_\pi \cup E_\pi)$ with $\alpha(y) = x$. From the inductive definition of the processes of an oclet system follows that if $Proc(\Omega) \subset R$, then R is not closed wrt. Ω.

6   Modeling with Oclets

In the previous three sections, we defined the formal semantics of oclets. With this semantics, the oclet system $\Omega_3^{\text{dec}} = (\text{Phil}_3^{\text{dec}}, \pi_0)$ with $\text{Phil}_3^{\text{dec}} := (\bigcup_{i=1}^3 \{\text{phil}(i)\}, \bigcup_{i=1}^3 \{\text{decentL}(i), \text{decentR}(i)\})$ of our introductory Sect. 2 has only decent runs by construction. The run π_3 of Fig. 2 is a process of Ω_3^{dec}.

To give a better understanding for the use of oclet systems, we explain two application scenarios for oclets in the section. First, we use oclets to specify the solution of n dining philosophers with only $n - 1$ forks available. Secondly, we sketch how oclets allow to quickly *adapt* system models.

6.1   Scenario-Based System Design with Oclets

We consider a variant of the dining philosophers: The philosophers are still sitting around a table and alternate between thinking and eating by taking and releasing forks they share with their neighbors. Unfortunately, one fork f_i is missing; philosophers i and $i \oplus 1$ cannot eat. The task: *Specify a system of n philosophers with $n - 1$ forks s.t. each philosopher always eventually eats.*

In our solution, the philosophers pass the forks around the table: Every philosopher $i \oplus 1$, who is missing his right fork $i \oplus 2$ may request the left fork of his left neighbor i. Philosopher i grants this request by passing his left fork i to his right neighbor $i \oplus 1$, who put this fork to his right. Now, phil. $i \ominus 1$ is missing his right fork i; he may send a request to his left neighbor. To allow each phil. i to eat at least once before passing his left fork, he may do so only after he has just returned from eating. Oclet exchF(i) of Figure 7 denotes exactly this specification.
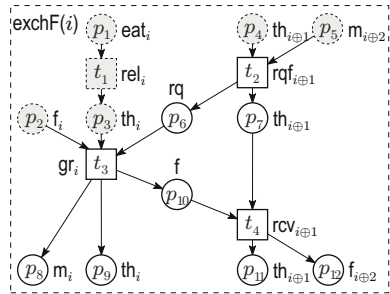

Fig. 7. Oclet exchF(i) specifying the fork-passing protocol

Now, consider the oclet system $\Omega_3^m := (\mathsf{Phil}_3^m, \pi_0^m)$ with oclet specification $\mathsf{Phil}_3^m := (\bigcup_{i=1}^3 \{\mathsf{phil}(i), \mathsf{exchF}(i)\}, \emptyset)$ and initial process π_0^m having initial conditions with labels $\mathsf{f}_1, \mathsf{th}_1, \mathsf{f}_2, \mathsf{th}_2, \mathsf{m}_3, \mathsf{th}_3$. Figure 8 depicts a process π_4^m of Ω_3^m: In π_0^m, transition take_1 of $\mathsf{phil}(1)$ and rqf_2 of $\mathsf{exchF}(1)$ are enabled concurrently. Because we made the enabling of a transition dependent only its own history (instead of the entire precondition of its oclet), it is possible to start executing a scenario even if not the entire precondition was observed. Theorem 2 justifies this behavior. Firing take_1 and rqf_2 yields π_1^m (nodes $b_1, \ldots, b_9, e_1, e_2$).

In π_1^m, only rel_1 of $\mathsf{phil}(1)$ is enabled, yielding π_2^m (nodes $b_1, \ldots, b_{12}$, $e_1, \ldots, e_3$). Transitions take_1 of $\mathsf{phil}(1)$, and gr_1 of $\mathsf{exchF}(1)$ are enabled in conflict: their presets can only be overlappingly embedded. Firing gr_1 and the subsequently enabled rcv_2 of $\mathsf{exchF}(1)$ constructs π_3^m $(b_1, \ldots, b_{17}, e_1, \ldots, e_5)$ where philosopher 2 can now take both forks. Continuing with the construction, we reach π_4^m where now philosopher 2 has to choose whether to grant the request of philosopher 3 or whether to take the forks again.

The system Ω_3^m solves the missing fork problem for 3 philosophers, but has non-decent runs. We can easily refine Phil_3^m to $\mathsf{Phil}_3^{m,d}$ by adding anti-oclets $\mathsf{decentL}(i)$ and $\mathsf{decentR}(i)$ of Fig. 4 for $i = 1, 2, 3$. Process π_4^m is also a process of the refined system $\Omega_3^{m,d} := (\mathsf{Phil}_3^{m,d}, \pi_0^m)$, and cannot be extended by e_9 (take_2) because of t_2 of $\mathsf{decentL}(2)$. The system $\Omega_3^{m,d}$ has only decent runs by construction.

Fig. 8. A process π_4^m of the fork-passing philosophers

This solution has another unfair run in case of more than three philosophers: Assume phil. 2 requests and receives fork f_1 from phil. 1 and puts it as fork f_3 between phil. 2 and phil. 3. Fork f_3 can equally be taken from phils. 2 and 3. Meanwhile, the other philosophers may have kept on passing forks until phil. 4 requests f_3 from 3. If now phil. 3 takes and releases his forks, and then grants the request of 4, phil. 2 was not able to eat with the forks he just has requested. The specification can easily be extended with an anti-oclet to prevent this behavior.

6.2   Adapting System Models with Oclets

This rather flexible style of creating oclet specifications also helps when specifying systems that have to be adapted frequently. Processes in disaster response are such as case, where the system model must be adapted to incorporate changes of the real-world processes [3].
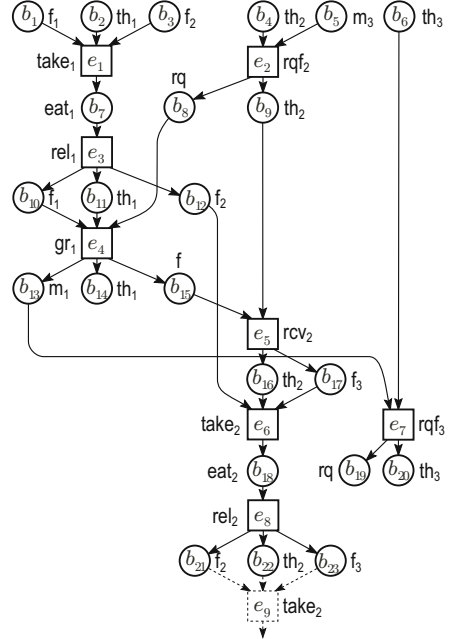
As it is fairly easy to add, remove, and modify single scenarios, the changes are well-conceivable and do not break the model. Because our operational semantics makes no assumptions regarding the initial process, adaptation can be done as follows: Construct a process π of an oclet system (O, π_0) until a problem is encountered. Change specification O by adding, removing, or modifying oclets; $O \rightarrow O'$. Then continue in the system (O', π). Iterate this procedure, possibly beginning again at π_0 or π, until the system is adapted. Our formal semantics guarantees well-defined behavior at any time during adaptation.

7   Related Work

In this section, we compare our approach for scenario-based modeling with Petri nets to existing works.

MSCs formalize scenarios as partial orders on events; several extensions are available. Hierarchical MSCs (HMSCs) and Message Sequence Graphs (MSGs) explicitly denote in a graph how scenarios may be concatenated, for specifying entire systems. Operational semantics of (H)MSCs and MSGs translate a specification into process algebraic expressions [5], automata [4], or employ graph grammars to construct runs [7] from MSCs. These, as well as existing Petri net semantics like [11] do not support anti-scenarios.

LSCs are an extension of MSCs with a formal semantics for overlapping scenarios, anti-scenarios, and modalities for scenarios and events. LSCs are more expressive than oclets; the original LSC semantics is declarative. Operational semantics for LSCs, i.e. LSC play-out, is defined by a translation to state charts [6], or by constructing an automaton from a specification [12]. Unfortunately, this linearizes the partial order explicitly specified in the charts.

Desel et al brought up the approach of scenario-based system design and validation with Petri nets [13,2]. The principle idea is to let the system designer denote desired and undesired behavior as complete (finite) partially ordered runs, i.e. complete scenarios and anti-scenarios. These mediate between a formal specification and a system model (a Petri net): Specification and model are validated against the scenarios, that is, whether each scenario satisfies the specification and whether the system model executes the desired scenarios while disallowing the undesired ones. The modeler iteratively reaches a valid system model; thereby refinement of the system model is a creative step involving human interaction. This step can be supported by folding desired scenarios into an overapproximating Petri net [2]. In [14], Bergenthum et al show how an equivalently implementing Petri net can be synthesized from a complete set of finite desired runs. The approach is extended in [15] where desired behavior is given as a regular expression over finite scenarios.

The oclet model follows this idea of scenario-based system design. Oclets contribute history-based preconditions to scenarios allowing that a scenario specifies behavior "in the middle" of an execution. Thereby the composition of scenarios to complete runs follows from the oclet's inherent precondition requiring no further notion like an expression for composition. Our operational semantics

allows to execute a set of scenarios directly without the need for additional synthesis or transformation. In that respect, our semantics makes a set of scenarios a *complete* system model. Still, a synthesis into Petri nets as in [14,15] allows to use the entire Petri net theory for verification.

The concept of history-dependent firing of transitions has been proposed earlier by defined corresponding transiting guards [16]; oclets provide a graphical syntax for a subclass of these guards. The net composition techniques defined in [17] are a general case of the net composition employed in our model. In the context of adapting system models, existing works in the area of adaptive workflows, see [18] for a survey, use models with sequential semantics or require to denote adaptations in explicit model transformation rules. Graph transformations on nets also require explicit adaptations rules for adaptations, e.g. [19]. In comparison, adaptations of oclet systems can be done from the perspective of desired and undesired scenarios only.

8   Conclusion

We presented a novel formal model for specifying and modeling systems with Petri net scenarios. We defined a specification to be a set of oclets, labeled causal nets with a dedicated precondition; oclets are partitioned into qualified oclets and anti-oclet describing desired and forbidden behavior, respectively.

We defined a declarative semantics that characterizes sets of runs that satisfy a given specification. We then provided an operational semantics to construct a maximal set of satisfying runs; we have shown that any run, that cannot be constructed either violates the specification, or includes an action that is not defined in the specification. We solved the dining philosophers problem in two variants to illustrate how our model can be used for modeling distributed systems. Providing an operational, partial-order Petri net semantics for scenarios *and* anti-scenarios makes our work a contribution in the area of scenario-based techniques.

Our results hint to further research: We already have first results towards constructing the complete finite prefix of a branching process of an oclet system [10] which allows for the verification of oclet systems (the full branching process can already be constructed with the given semantics). These results also hint towards a synthesis of Petri nets from scenarios and anti-scenarios. We also research structural properties of oclet specifications to derive system properties directly from scenarios and intend to introduce modalities known from LSCs such as imperative scenarios and events.

Our approach is implemented in our graphical runtime environment Greta, that is available online at `http://www.service-technology.org/greta/` together with several example specifications.

References

1. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. Form. Methods Syst. Des. 19(1), 45–80 (2001)
2. Desel, J.: From human knowledge to process models. In: UNISCON, pp. 84–95 (2008)
3. Fahland, D., Woith, H.: Towards process models for disaster response. In: Ardagna, D., et al. (eds.) BPM 2008 Workshops. LNBIP, vol. 17, pp. 244–256. Springer, Heidelberg (2008)
4. Mukund, M., Kumar, K.N., Thiagarajan, P.S.: Netcharts: Bridging the gap between HMSCs and executable specifications. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 293–307. Springer, Heidelberg (2003)
5. Mauw, S., Reniers, M.A.: An algebraic semantics of Basic Message Sequence Charts. The Computer Journal 37, 269–277 (1994)
6. Harel, D., Kugler, H.: Synthesizing State-Based Object Systems from LSC Specifications. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 1–33. Springer, Heidelberg (2001)
7. Hélouët, L., Jard, C., Caillaud, B.: An event structure based semantics for high-level message sequence charts. Mathematical. Structures in Comp. Sci. 12(4), 377–402 (2002)
8. Engelfriet, J.: Branching processes of Petri nets. Acta Inf. 28(6), 575–591 (1991)
9. Reisig, W.: Elements Of Distributed Algorithms: Modeling and Analysis with Petri Nets. Springer, Heidelberg (1998)
10. Esparza, J., Heljanko, K.: Unfoldings - A Partial-Order Approach to Model Checking. Springer, Heidelberg (2008)
11. Kluge, O.: Petri nets as a semantic model for Message Sequence Chart specifications. In: INT 2002, Grenoble, France, pp. 138–147 (2002)
12. Harel, D., Kugler, H., Marelly, R., Pnueli, A.: Smart play-out of behavioral requirements. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD 2002. LNCS, vol. 2517, pp. 378–398. Springer, Heidelberg (2002)
13. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and validation with VipTool. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 380–389. Springer, Heidelberg (2003)
14. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundam. Inform. 88(4), 437–468 (2008)
15. Bergenthum, R., Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages with VipTool. In: AWPN 2008, Rostock, Germany, University of Rostock (September 2008)
16. Hee, K., Serebrenik, A., Sidorova, N., Voorhoeve, M., Werf, J.: Modelling with History-Dependent Petri Nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 320–327. Springer, Heidelberg (2007)
17. Barros, J.a.P., Gomes, L.: Net model composition and modification by net operations: a pragmatic approach. In: Proceedings of INDIN 2004, Berlin, Germany (June 2004)
18. Rinderle, S., Reichert, M., Dadam, P.: Evaluation of correctness criteria for dynamic workflow changes. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 41–57. Springer, Heidelberg (2003)
19. Ehrig, H., Hoffmann, K., Padberg, J., Prange, U., Ermel, C.: Independence of net transformations and token firing in reconfigurable place/transition systems. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 104–123. Springer, Heidelberg (2007)