Formalizing Application Integration Patterns

Daniel Ritter, Stefanie Rinderle-Ma
University of Vienna
{firstname.lastname}@univie.ac.at

Marco Montali, Andrey Rivkin, Aman Sinha
University of Bolzano
{lastname}@inf.unibz.it

***Abstract*—Enterprise Integration Patterns (EIPs) and their extensions denote the informally described building blocks of current Enterprise Application Integration (EAI) systems. Although a recent approach strives to provide an EIP formalization based on Colored Petri Nets (CPNs), it does not completely consider EAI requirements, such as complex data, transacted resources and time. In the absence of a comprehensive formal definition, the patterns cannot be formally verified, and thus a formal foundation of EAI is missing. In this work, we leverage the novel db-net approach that finds a better balance between the data and process-related aspects than CPNs and we extend it according to the EAI requirements that we systematically collect on a pattern level. Then we discuss pattern realizations, and evaluate our approach for comprehensiveness, test correctness, and show its applicability.**

I. Introduction

With the growing number of cloud and mobile applications, the importance of Enterprise Application Integration (EAI) [14] immensely increases. Integration scenarios – essentially compositions of Enterprise Integration Patterns (EIPs) [10] and their recent extensions [19], [20] – describe typical concepts in designing messaging systems as used for EAI (e. g., the communication between these applications). The current EAI system vendors use many of the EIPs as part of their proprietary integration scenario modeling languages [19]. However, these languages are not ground on any formalism and, hence, may produce models that are subject to design flaws such as functional errors, missing or incomplete functionalities. Currently, detection and analysis of these flaws are by large performed manually. Hence, EIPs can rather be considered as a set of informal design solutions than a formal language for modeling and verifying correctness of integration patterns, thus leaving the EAI vendors with their own proprietary semantics.

Our recent survey [19] concludes that there was only one attempt towards formalization of some integration patterns using colored Petri nets (CPNs) [8]. Although the CPN colors abstractly stand for data types and CPN-based approaches naturally support the control flow through control threads (i. e., tokens) progressing through the net, carrying data conforming to colors, they cannot be used to model, query, update, and reason on requirements inherent to the extended EIPs [10], [19], [20] such as persistent data or timings. To overcome these issues, we set three objectives, that allow to (i) formalize and (ii) simulate EIPs, as well as (iii) verify their realizations, and argue that existing approaches do not fully support them.

In this work, we leverage db-nets [16] as a database-centric extension of CPNs (incl. atomic transactions) and extend them

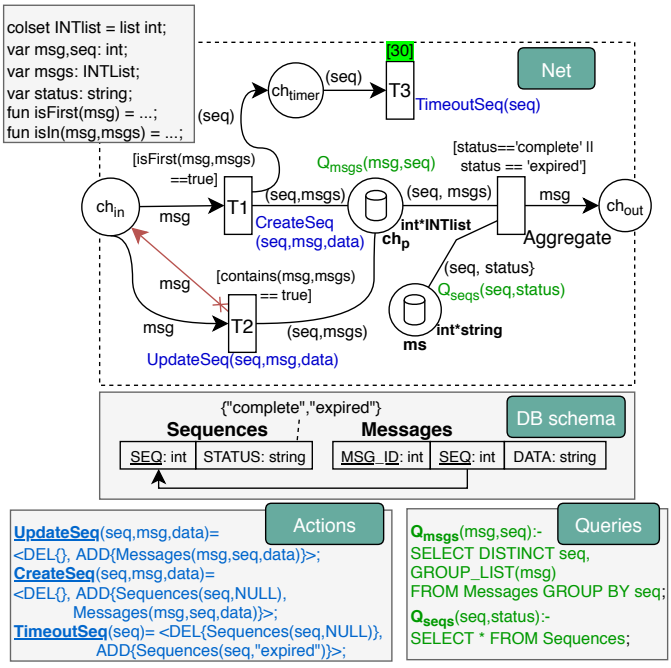


Fig. 1: Aggregator pattern variant as a timed db-net

by further EAI requirements like time. For example, Fig. 1 specifies semantics of a commonly used stateful Aggregator pattern [10] as an extended CPN, which we call *timed db-net*. Here, the aggregator collects messages in a persistent storage that is accessed via a special view place ch_p, and then aggregates them based either on the completion condition (e. g., sequence status is `complete`, modeled via `Aggregate` transition) or on time-out of 30 sec (modeled via transition `T3`). To collect messages and assign them to correct sequences, the net correlates every incoming `msg` token to those in ch_p place, that, in turn, stores pairs of sequences and lists of messages that have been already collected. If the message is the first in a sequence, new entries, one containing information about the message and another containing data about the referenced sequence, are added to tables called `Messages` and `Sequences`, respectively. This is achieved by firing transition `T1` and executing action `CreateSeq` attached to it. Otherwise, a message is inserted into `Messages` by firing `T2` and executing `UpdateSeq`. However, the update by `UpdateSeq` fails, if a message is already in the database or a referenced sequence has already been aggregated due to a timeout (i.e., status is `expired`). In this case the net switches to an alternative roll-back flow (a directed arc from `T2` to ch_{in})

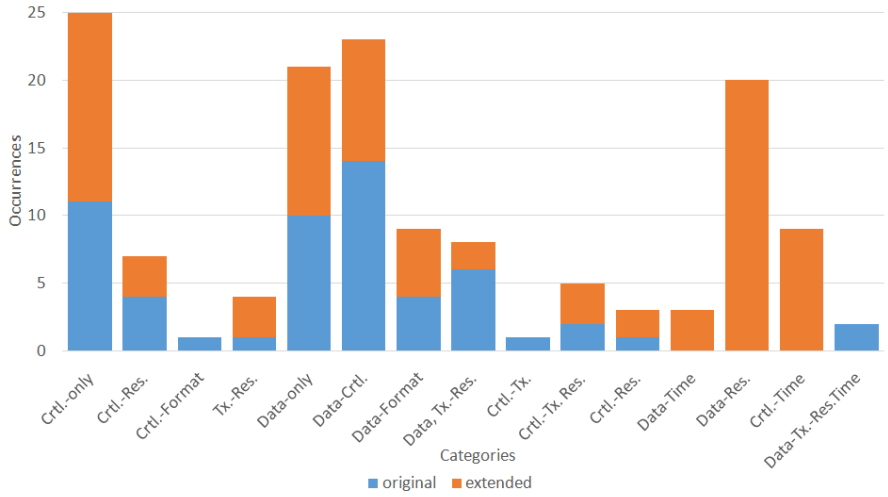


Fig. 2: EIP requirement categories (with Control (Crtl.), Resource (Res.), Transaction (Tx.))

and puts the message back to the message channel ch_{in}.

In summary we address the following research questions:

- Q1 What are relevant EAI requirements for the formal definition of EIPs?
- Q2 Which formalism allows to specify, simulate and verify EIPs under extracted requirements?
- Q3 How to realize the EIPs and real-world integration scenarios?

We approach them following the methodology from [17]. In Sect. II, pattern requirements are harvested from the literature (i. e., existing catalogs with 166 integration patterns) in a quantitative analysis (cf. Q1). Then, in Sect. III, we briefly introduce db-nets and, following objectives (i)–(iii) for single patterns, build on top of the existing CPN approach [8] by adding persistent data and time, consecutively deriving the timed db-net formalism (cf. Q2). Several of the patterns are then realized using timed db-net in Sect. IV[1]. In Sect. V, we elaborate on the comprehensiveness of the formalism in a quantitative study, show a prototypical db-net realization for testing correctness[1], and discuss the general applicability of PNs and, in particular, timed db-nets for the composition of integration patterns (cf. Q3) in a real-world example. We conclude by presenting related work in Section VI and outlining the main results and future research directions in Sect. VII.

II. Formalization Requirements Analysis

In this section, we collect the EAI requirements relevant for the formalization of the EIPs by analyzing the existing pattern catalogs [10], [19], [20] (cf. Q1). Then we briefly discuss which of them can be represented by the means of CPNs or db-nets, and which require further extensions.

A. Pattern Analysis and Categories

The EIP formalization requirements are derived by an analysis of the pattern descriptions based on the integration pattern catalogs from 2004 [10] (as `original`) and recent extensions [19], [20] (as `extended`) that consider emerging EAI scenarios (e. g., cloud, mobile and internet of things). Together the catalogs describe 166 integration-relevant patterns, of which we consider 139 due to their relevance for this work (e. g., excluding abstract concepts like Canonical Data Model [10] or Messaging System [10]). During the analysis, we manually collected characteristics from the textual pattern descriptions and created new categories, if not existent.

The reoccurring characteristics found in this work allow for a categorization of patterns as summarized in Fig. 2 to systematically pinpoint relevant EAI requirements into general categories (with more than one pattern). Most of the patterns require (combinations of) *Data* flow, *Control* (Crtl.) flow, and *(Transacted) Resource* ((Tx.) Res.) access. While the control flow denotes the routing of a message from pattern to pattern via channels (i. e., ordered execution), the data flow describes the access of the actual message by patterns (incl. message content, headers, attachments). Notably, most of the patterns can be classified as control (Crtl.-only; e. g., Wire Tap [10]) and data only (Data-only; e. g., Splitter [10]) or as their combination (Data-Crtl.; e. g., Message Filter [10]), which stresses on the importance of data-aspects of the routing and transformation patterns. In addition, resources denote data from an external service not in the message (e. g., Data Store [19]). The EIP extensions add new categories like combinations of data and {time, resources} (Data-Time like Message Expiration [10], [19], Data-Res. like Encryptor [19]) and control and time (Crtl.-Time; e. g., Throttler [19]). For instance, the motivating example in Fig. 1 is classified as *Data-Tx.-Res.-Time*. The different categories are disjoint with respect to patterns.

B. From Categories to Requirements

We assume that the control requirement **REQ-0 "Control flow"** is inherently covered by any PN approach, and thus in CPN and db-net. However, there are two particularities in the routing patterns that we capture in requirement **REQ-1 "Msg. channel priority, order"**: (a) the ordered evaluation of Msg. channel conditions or guards of sibling PN transitions, required

[1]Due to brevity, further realizations and more on correctness testing can be found in the non-mandatory supplementary material https://bit.ly/2ItNpTZ.

TABLE I: Formalization Requirements (covered $\surd$, partially ($\surd$), not -)

ID	Requirement	CPN	db-net
REQ-0	Control flow (pipes and filter)	$\surd$	$\surd$
REQ-1	(a) Msg. channel **priority**	($\surd$)	($\surd$)
	(b) Msg. channel **distribution**	-	($\surd$)
REQ-2	**Data, format** incl. message protocol with encoding, security	($\surd$)	$\surd$
REQ-3	(a) **Timeout** on message, operation	-	-
	(b) **Expiry date** on message	-	-
	(c) **Delay** of message, operation	-	-
	(d) **Msg./time ratio**	-	-
REQ-4	(a) **CRUD operations** on (external) resources	-	$\surd$
	(b) **Transaction semantics** on (external) resources (incl. roll-back)	-	$\surd$
REQ-5	**Exceptions, compensation** similar to roll-back in REQ-4	-	$\surd$

for the Content-based Router pattern, (b) the enablement or firing of a PN transition according to a ratio for the realization of a Load Balancer [19]. In both cases, neither execution priorities nor ratios are trivially in CPN or db-net.

Furthermore, there are 77 patterns in the catalogs with data and 10 with message format aspects, which require an expressive CPN token representation (e. g., for encodings, security, complex message protocols), for which we add a second requirement **REQ-2 "data, format"** that has to allow for the formal analysis of the data. Although CPNs and db-nets have to be severely restricted (e. g., finite color domains, pre-defined number of elements) for that, db-nets promise a relational representation that can be formally analyzed [16].

We capture the 11 patterns with time-related requirements as **REQ-3 "time"**: (a) Timeout: numerical representation of fixed, relative time (i. e., no global time); (b) Expiry date: discrete point in time according to a global time (i. e., based on existing message content); (c) Delay: numerical, fixed value time to wait or pause until continued: e. g., often used in a redelivery policy (d) Message/time ratio: number of messages that are sent during a period of time. Consequently, a quantified, fixed time delay or duration semantics is required.

The 49 patterns with resources **REQ-4 "(external) resources"** require: (a) create, retrieve, update, delete (**CRUD**) access to external services or resources, and (b) **transaction** semantics on a pattern level. Similarly, exception semantics are present in 28 patterns as **REQ-5 "exceptions"**, which require **compensations** and other post-error actions. Consequently, a PN definition that allows for reasoning over these timing and structured (persistent) data access is required.

C. Requirements Summary

Table I summarizes the formalization requirements for timed db-nets by setting the coverage of the CPN [8] and db-net [16] approaches into context. While CPNs provide a solid foundation for control (cf. REQ-0) and a simple data flow representation (cf. REQ-2), db-nets extend it towards more complex data structures—message protocols in our case (cf. REQ2), and add CRUD operations (cf. REQ-4(a)), transactional semantics (cf. REQ-4(b)), and exception handling (cf. REQ-5), suitable for working with external, transactional resources. The message channel priorities can be represented both by explicitly modeling them leading to complex models and similar for message channel distributions for db-net. In this work we build upon these approaches by subsequently defining timed db-nets in Sect. III for the time-related requirements (cf. REQ-3(a)–(d)) and provide realizations for message channel priority execution (cf. REQ-1(a); only implicitly in CPN, db-net) and load balancing (cf. REQ-4(b)) in Sect. IV.

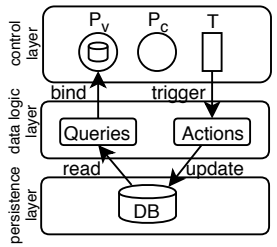


Fig. 3: Db-net layers (similar to [16])

III. Integration Pattern Formalization

We recall db-nets [16] and extend them with time, consequently deriving a new formalism called timed db-nets. We then consider the formal analysis of timed db-nets (cf. Q2).

A. Db-net: Data, transacted resources and compensation

In the context of extending classical Petri nets with complex data, there are plenty of works that go beyond CPNs: data nets [13] and ν-nets [21], Petri nets with nested terms [23], nested relations [9] and XML documents [2]. While all of the approaches treat data subsidiary to the control-flow dimension, the EIPs require data elements attached to tokens being connected to each other by global data models (cf. Sect. II). Consequently, they do not allow for reasoning on persistent, relational data (assuming tree and graph structured formats like XML can be represented by database relations [18]).

Db-net. The recent work on db-nets [16] strives to achieve this by combining CPNs with relational databases, separating the (database) persistence layer $\mathcal{P}$ from the PN control layer $\mathcal{N}$ as illustrated in Fig. 3. This is realized by an intermediate data logic layer $\mathcal{P}$ that mediates between the two by supporting the control layer with queries and database operations (e. g., `trigger`, `update`, `read`, `bind`). We select db-nets (see Def. 1) as a foundation of timed db-nets due to their ability to represent relational data (cf. REQ-2: "data", "format"), and the built-in support for transactional CRUD operations (cf. REQ-4) as well as exception handling that require compensations (cf. REQ-5). Moreover, since db-nets are based on CPNs, it is possible to leverage existing simulation techniques of the latter [16].

Definition 1 ([16]): A *db-net* is a tuple $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$, where:

- $\mathfrak{D}$ is a type domain – a finite set of data types $D = \langle \Delta_D, \Gamma_D \rangle$, with the value domain Δ_D and a finite set of predicate symbols Γ_D.
- $\mathcal{P}$ is a $\mathfrak{D}$-typed **persistence layer**, i.e., a pair $\langle \mathcal{R}, E \rangle$, where $\mathcal{R}$ is a $\mathfrak{D}$-typed database schema, and E is a finite set of first-order FO($\mathfrak{D}$) constraints over $\mathcal{R}$.
- $\mathcal{L}$ is a $\mathfrak{D}$-typed **data logic layer** over $\mathcal{P}$, i.e., a pair $\langle Q, A \rangle$, where Q is a finite set of FO($\mathfrak{D}$) queries over $\mathcal{P}$, and A is a finite set of actions over $\mathcal{P}$.

- $\mathcal{N}$ is a $\mathfrak{D}$-typed **control layer** $\mathcal{L}$, i.e., a tuple $(P, T, F_{in}, F_{out}, \texttt{color}, \texttt{query}, \texttt{guard}, \texttt{action})$, where: *(i)* $P = P_c \uplus P_v$ is a finite set of places partitioned into control places P_c and view places P_v, *(ii)* T is a finite set of transitions, *(iii)* F_{in} is an input flow from P to T, a normal output flow F_{out} and a roll-back flow *(iv)* F_{out} and F_{rb} are respectively an output and roll-back flow from T to P_c *(v)* `color` is a color assignment over P (mapping P to a cartesian product of data types), *(vi)* `query` is a query assignment from P_v to Q (mapping the results of Q as tokens of P_v, *(vii)* `guard` is a transition guard assignment over T (mapping each transition to a formula over its input inscriptions), and *(viii)* `action` is an action assignment from T to A (mapping some transitions to actions triggering updates over the persistence layer). □

Input and output/roll-back flows contain inscriptions that match the components of colored tokens present in the input and output/roll-back places of a transition. Such inscriptions consists of tuples of (typed) variables, which can then be mentioned in the transition guard, and also in the action assignment (to bind the updates induced by the action to the values chosen to match the inscriptions). Given a transition t, we denote by $InVars(t)$ the set of variables mentioned in its input flows, by $OutVars(t)$ the set of variables mentioned in its output flows, and by $OutVars(t)$ the set of variables occurring in the action assignment of t (if any). Fresh variables $FreshVars(t) = OutVars(t) \setminus InVars(t)$ denote those output variables that do not match any corresponding input variables, and are consequently interpreted as external inputs.

Thereby the control layer can be seen as a CPN extended with database queries, assigned to special view places, and special database update operations attached to transitions. The data logic layer binds and then executes the queries and actions on the persistence layer. The terms message and (db-net, CPN) token will be used synonymously hereinafter.

Db-net execution semantics. Briefly, the execution semantics of a db-net in Def. 1 accounts for the progression of a database instance compliant with the persistence layer $\mathcal{P}$ and the evolution of a marking over the control layer $\mathcal{N}$, mediated by the data logic layer $\mathcal{L}$. While the marking over the control layer determines the transitions to be fired, it triggers updates of the database instance. In particular, the distributed tokens have to carry data compatible with the color of the places and the marking of a view place P_v must correspond to the associated queries over the underlying database instance. The markings follow the active domain semantics of database systems (i. e., D-active domain, with D $\in \mathfrak{D}$) [16]. Furthermore, the db-net persistence and control layers are stateful. During the execution, in each moment (called *snapshot*) the persistence layer is associated to a database instance I, and the control layer is associated to a marking m aligned with I via `query` (for what concerns the content of view places). The corresponding snapshot is then simply the pair $\langle I, m \rangle$.

Similar to CPNs, the firing of a transition t in a snapshot is defined by a binding that maps the value domains of the different layers, if several properties are guaranteed, e. g., the guard attached to t is satisfied. This enables the transition, which then has the following effects: all matching tokens in control places P_c are consumed; then the action instance `action` — induced by the firing — is applied on the current database instance in an atomic transaction (and rolled-back, if not successful); and accordingly, tokens on output places F_{out} or roll-back places F_{rb} (i. e., , those connected via roll-back flow) are produced. Details are given in [16].

All in all, the complete execution semantics of a db-net is captured by an infinite-state transition systems where each transition represents the firing of a transition in the control layer of the net with a given binding, and each state is a snapshot. Notice that, due to the presence of unbounded colors and of the underlying persistence layer, the transition system may contain infinitely many different states even if the control layer is bounded in the classical Petri net sense. However, this infinity can be tamed using faithful abstraction techniques [16].

Example. The aggregator in Fig. 1 requires a view place P_v (denoted by ⓔ) for storing and updating the message sequences as well as roll-back place F_{rb} (ch_{in}) connected through ⟷). The graphical notation is in line with [16].

Notably, the db-net definition and execution semantics do not address timing or ordering aspects. Hence, we subsequently extend db-nets accordingly or find suitable realizations.

B. Extending db-nets with time

While the implicit temporal support in PNs (i. e., adding places representing the current time) is rather cumbersome [**?**], the temporal semantics of adding timestamps to tokens [**?**], timed places [22], arcs [12] and transitions [28] are well studied and naturally capture different treatments of time in dynamic systems. The temporal requirements in REQ-3 demand a quantified, fixed or discrete time representation by timed transitions or places, representing the delay induced by a transition firing. This is currently missing in db-nets. So, in the spectrum of timed extensions to PNs, we extend the db-net control layer $\mathcal{N}$ with a temporal semantics that has two advantages: it captures the aforementioned requirements, and at the same time it is well-behaved in terms of formal analysis.

We start by explaining the intuition behind the approach, and then provide the corresponding formalization. We assume that there is a global, continuous notion of time. The firing of a transition is instantaneous, but can only occur in certain moments of time, while it is inhibited in others, even in presence of the required input tokens. Every *control token*, that is, token assigned to a control place, carries a (local) *age*, indicating how much time the token is spending in that control place. This means that when a token enters into a place, it is assigned an age of 0. The age then increments as the time flows and the token stays put in the same place. View places continuously access the underlying persistence layer, and consequently their (virtual) tokens do not age. Each transition is assigned to a pair of non-negative (possibly identical) rational numbers, respectively describing the minimum and maximum age that input tokens should have when they are used to fire the transition. Thus, such numbers identify a relative time window that delays the possibility of firing.

Definition 2: A *timed db-net* is a tuple $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau \rangle$ where $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N} \rangle$ is a db-net whose control layer $\mathcal{N}$ contains a set T of transitions, while $\tau : T \to \mathbb{Q}^{\geq 0} \times (\mathbb{Q}^{\geq 0} \cup \{\infty\})$ is a *timed transition function* that maps each transition $t \in T$ to a pair of values $\tau(t) = \langle v_1, v_2 \rangle$, such that: *(i)* v_1 is a non-negative rational number; *(ii)* v_2 is either a non-negative rational number, or the special constant ∞; *(iii)* either $v_2 = \infty$, or $v_1 \leq v_2$. □

The default choice for τ is to map transitions to the pair $\langle 0, \infty \rangle$, which correspond to a standard db-net transition.

Given a transition t, we adopt the following graphical conventions: *(i)* if $\tau(t) = \langle 0, \infty \rangle$, then no temporal label is shown for t; *(ii)* if $\tau(t)$ is of the form $\langle v, v \rangle$, we attach label "$[v]$" to t; *(iii)* if $\tau(t)$ is of the form $\langle v_1, v_2 \rangle$ with $v_1 \neq v_2$, we attach label "$[v_1, v_2]$" to t.

Example. The aggregator in Fig. 1 defines a timed transition 30, as Timer, on the creation of a new sequence `seq` in a $\mathfrak{D}$-typed message or token, which times out after 30 time units, here seconds, after the sequence has been created. Then it enables the `Aggregate` transition, by updating the sequence's status on the database to `expired` using action `TimeoutSeq`. Obviously, the intended behaviour of the overall net is properly captured when the Timer *actually fires* upon enablement.

Timed db-net execution semantics. The execution semantics of timed db-nets builds upon the one for standard db-nets. The management of bindings, guards, and database updates via actions, is kept unaltered, while additional conditions on the flow of time and the temporal enablement of transitions considering delays have to be properly tackled. On the one hand, as customary in temporal extensions of Petri nets, we account for the flow of time by indicating that a given amount of time is passing while the net is not firing any transition. This results in incrementing the delay of tokens accordingly. On the other hand, the enablement of transitions has to be revised by checking whether the ages of tokens that the transition intends to consume match the delay window attached to the transition.

Technically, we proceed as follows. We introduce a special domain $\mathbb{A}$ in $\mathfrak{D}$ capturing the age of tokens. We assume that *every* control place p is typed with a color that contains, as a last component, $\mathbb{A}$. Consequently, a marking now has also to assign an age to each token. In addition, we also assume that all arc inscriptions carry the age of matching tokens.

With this intuition at hand, we reconstruct the formal definition of enablement and firing given in [16], as follows (see in particular the last two conditions).

Definition 3: Let B be a timed db-net $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}, \tau \rangle$, and t a transition in $\mathcal{N}$ with $\tau(t) = \langle v_1, v_2 \rangle$. Furthermore, let σ be a binding for t, i.e., a substitution $\sigma : \mathit{Vars}(t) \to \Delta_{\mathfrak{D}}$, where $\mathit{Vars}(t) = \mathit{InVars}(t) \cup \mathit{OutVars}(t)$. A transition $t \in T$ is *enabled* in a B-snapshot $\langle I, m \rangle$ with binding σ, if:

- For every place $p \in \mathcal{P}$, $m(p)$ provides enough tokens matching those required by inscription $w = F_{in}(\langle p, t \rangle)$, once w is grounded by σ, i. e., $\sigma^{\oplus}(w) \subseteq m(p)$;
- The instantiated guard $\mathtt{guard}(t)\sigma$ evaluates to true.
- σ is injective over $\mathit{FreshVars}(t)$, thus guaranteeing that fresh variables are assigned to pairwise distinct values of σ, and for every fresh variable $v \in \mathit{FreshVars}(t)$, $\sigma(v) \notin (\mathit{Adom}_{\mathtt{type}(v)}(I) \cup \mathit{Adom}_{\mathtt{type}(v)}(m))$.[2]
- For each age variable $y \in \mathit{OutVars}(t)$, we have that $\sigma(y) = 0$ (i.e., newly produced tokens get an age of 0). □

Thanks to the fact that ages of tokens are properly checked and updated by the transition binding when defining enablement, we have that the definition of firing for timed db-nets is *identical* to that of standard db-nets.

Definition 4 ([16]): Let B be a timed db-net $\langle \mathfrak{D}, \mathcal{P}, \mathcal{L}, \mathcal{N}\tau \rangle$, and $s_1 = \langle I_1, m_1 \rangle, s_2 = \langle I_2, m_2 \rangle$ be two B-snapshots. Fix a transition t of $\mathcal{N}$ and a binding σ such that t is enabled in s_1 with σ (cf. Def. 3). Let $I_3 = \mathit{apply}(\mathit{action}_{\sigma}(t), I_1)$ be the database instance resulting from the application of the action attached to t on database instance I_1 with binding σ for the action parameters. For a control place p, let $w_{in}(p,t) = F_{in}(\langle p, t \rangle)$, and $w_{out}(p,t) = F_{out}(\langle p, t \rangle)$ if I_3 is compliant with $\mathcal{P}$, or $w_{out}(p,t) = F_{rb}(\langle p, t \rangle)$ otherwise. We say that t *fires* in s_1 with binding σ producing s_2, written $s_1[t, \sigma\rangle s_2$, if:

- if I_3 is compliant with $\mathcal{P}$, then I_2=I_3, otherwise I_2=I_1;
- for each control place p, m_2 corresponds to m_1 with the following changes: $\sigma^{\oplus}(w_{in}(p,t))$ tokens are removed from p, and $\sigma^{\oplus}(w_{out}(p,t))$ are added to p. In formulae: $m_2(p_c) = (m_1(p_c) - \sigma^{\oplus}(w_{in}(p,t))) + \sigma^{\oplus}(w_{out}(p,t))$ □

The execution semantics of a timed db-net then follows the standard construction, modulo two refined aspects. First, the original (non-timed) definition of enablement in [16] is substituted with its refined version in Def. 3. Second, when constructing the transition system, we also add the (infinitely many) transitions dealing with the flow of time. This is done by simply imposing that every B-snapshot $\langle I, m \rangle$ is connected to every B-snapshot of the form $\langle I', m' \rangle$ where $I' = I$ (i.e., the database instances are identical) and m' is identical to m but for the ages of tokens, which all get increased of the same, fixed amount $x \in \mathbb{Q}$ of time.

In the following, given two B-snapshots s and s', we write $s \to s'$ if there exists a direct transition from s to s' in the transition system that captures the execution semantics of B starting from a given initial snapshot s_0. We also write $s \xrightarrow{*} s'$ if there is a sequence of transitions leading from s to s'.

Example. To finish the aggregator definition, when the persisted sequence in the aggregator is complete or the sequence times out, then the enabled `Aggregate` transition fires by reading the sequence number `seq` and snapshot of the sequences' messages, and moving an aggregate msg' to ch_{out} Notably, the `Aggregate` transition is invariant to which of the two causes led to the completion of the sequence.

C. Checking Reachability over Timed Db-nets

Checking fundamental correctness properties such as *safety/reachability*is of particular importance for timed db-nets, in the light of the subsequent discussion in Sect. V-B on reachable goal states. We consider here, in particular, the following relevant REACH-TEMPLATE problem:

Input: *(i)* a timed db-net B with set P_c of control places, *(ii)* an initial B-snapshot s_0, *(iii)* a set $P_{empty} \subseteq P_c$ of

[2] $\mathit{Adom}_D(X)$ is the set of values of type D explicitly contained in X.

empty control places, *(iv)* a set $P_{filled} \subseteq P_c$ of *nonempty control places* such that $P_{empty} \cap P_{filled} = \emptyset$.

Output: *yes* if and only if there exists a finite sequence of B-snapshots of the form $s_0 \to \ldots \to s_n = \langle I_n, m_n \rangle$ such that for every place $p_e \in P_{empty}$, we have $|m_n(p_e)| = 0$, and for every place $p_f \in P_{filled}$, we have $|m_n(p_f) > 0$.

The possibility of checking some places to be empty in the target snapshot is especially relevant in the presence of timed transitions, in particular to require that they do not contain "old" tokens that were not consumed during the corresponding delay window. For example, by considering the Timer transition in Fig. 1, asking for the $\texttt{ch}_{\texttt{timer}}$ place to be empty guarantees that the timer indeed triggered whenever enabled.

Since timed db-nets build upon db-nets, reachability is highly undecidable, even for nets that do not employ timed transitions, have empty data logic and persistence layers, and only employ simple string colors. As pointed out in [16], this setting is in fact already expressive enough to capture ν-nets [13], [21], for which reachability is undecidable. Similar undecidability results can be obtained by restricting even more the control layer, but allowing for the insertion and deletion of arbitrarily many tuples in the underlying persistence layer.

However, when controlling the size of information maintained by the control and persistence layers in each single snapshot, reachability and also more sophisticated forms of temporal model checking become decidable for db-nets using string and real data types. In particular, decidability is obtained for db-nets that enjoy the three different types of boundedness. A db-net B with initial snapshot s_0 is:

- **width-bounded** if there is $b \in \mathbb{N}$ s.t., for every B-snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, then the number of values assigned by m to the places of B is bounded by b;
- **depth-bounded** if there is $b \in \mathbb{N}$ s.t., for every B-snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, then the number of tokens assigned by m to the places of B is bounded by b;
- **state-bounded** if there is $b \in \mathbb{N}$ s.t., for every B-snapshot $\langle I, m \rangle$, if $s_0 \xrightarrow{*} \langle I, m \rangle$, we have $|\cup_{D \in \mathfrak{D}} Adom_D(I)| \leq b$.

We say that a (timed) db-net is *bounded* if it is at once width-, depth-, and state-bounded.

Note that the decidability of reachability for bounded db-nets does not imply decidability of reachability for bounded timed db-nets. In fact, ages in timed db-nets are subject to arithmetic operations that are not tackled in db-nets. However, we can prove decidability by resorting to a *separation* argument: the two dimensions of infinity respectively related to the infinity of the data domains and of the flow of time can in fact be tamed orthogonally to each other. In particular, we get the following.

Theorem 1: The REACH-TEMPLATE problem is decidable for bounded timed db-nets with initial snapshot.

Proof 1 (Proof sketch): Consider a bounded timed db-net B with initial snapshot s_0, empty control places P_{empty}, and filled control places P_{filled}. Using the faithful data abstraction techniques presented in [16, Thm 2], one obtains a corresponding timed db-net B' enjoying two key properties. First, B' is bisimilar to B, with a data-aware notion of bisimulation that takes into account both the temporal dynamics induced by the net, as well as the correspondence between data elements. Such a notion of bisimulation captures reachability as defined above, and consequently REACH-TEMPLATE(B,s_0,P_{empty},P_{filled}) returns *yes* if and only if REACH-TEMPLATE(B',s_0,P_{empty},P_{filled}) does so. Second, the only source of infinity when characterizing the execution semantics of B' comes from the temporal aspects, and in particular the unboundedness of token ages. This means that B' can be considered as a "standard" temporal variant of a CPN with bounded colors that, in turn, boils down to a temporal variant of (uncolored) PN. In particular, one can easily see that B' corresponds to a special class of bounded ATPN [12], where whenever B' contains a transition t with $\tau(t) = \langle v_1, v_2 \rangle$, its corresponding ATPN labels each arc entering t with interval $[v_1, v_2]$. Consequently, the infinity of B' can be tamed using standard techniques known for bounded ATPNs, which indeed enjoy decidability of reachability [4], [1]. □

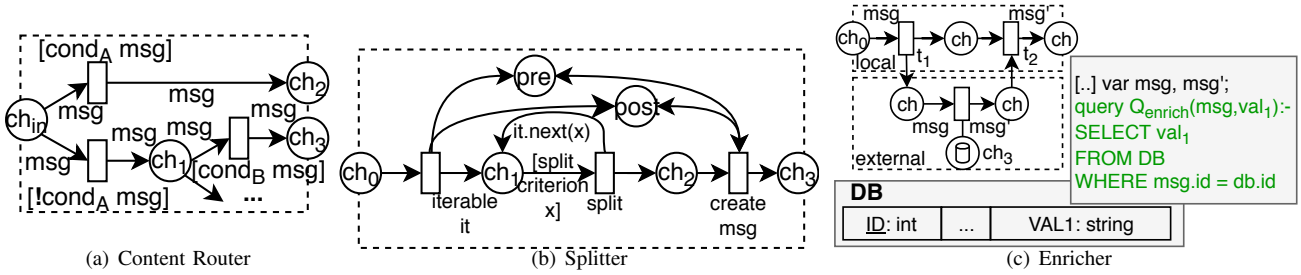
IV. Timed db-net Pattern Realizations

A pattern realization denotes a representation of the pattern in timed db-net (e. g., the aggregator in Fig. 1). In this section, we discuss pattern realizations for requirements REQ-1 (a)+(b) from Sect. II followed by realizations of those patterns required in the case study in Sect. V-C.

A. Patterns with Msg. Channel Ordering

The realization of a Content-based Router [10] with conditions $cond_A$, $cond_B$ that have to be evaluated strictly in-order (cf. REQ-1(a)) is shown in Fig. 4(a). Although the router could be realized more elegantly by using priority functions similar to [27], we explicitly realized them with pair-wise negated timed db-net transition guards: first $cond_A$ is evaluated and the message moved to ch_2, if it matches, else its negation is chosen. In case the latter matches, the second condition is checked, and then its negation, and so on. When non of the pairs alone matches, a non-guarded default transition fires (not shown). This explicit realization covers the router's semantics, however, requires $(k \times 2) + 1$ transitions (i. e., condition, negation, and only one default), with the number of conditions k.

Similarly, the load balancer [19] could be realized by using stochastic PNs [27], [3] or at least an extension of the transition db-net guard definition that sample probability values from a probability distribution (e. g., [11]). While this would extend the db-net persistence layer, for the resulting net the current decidability results would no longer hold, thus making the formal verification undecidable. Hence, we decided to leverage the db-net persistence layer to represent a `Balancing Ratio`, as shown in **??**, to persistently manage the ratios by a simple balancing scheme that uses the db-net transition guards. A message msg in channel ch_1 leads to a lookup of the current ratio by the two enabled transitions Inc_t_1 and Inc_t_2 and an evaluation of their transition formulas, e. g., for $Inc\_t_1(to_{ch_2}) := \frac{to_{ch_2}}{to_{ch_3}} \geq 0.7$. If the formula holds, the transition fires by moving the message to its output place as well as updating the table by a sequence of $Inc\_t.del = \{NumberCh_i(x)\}$, $Inc\_t.add = \{NumberCh_i(x+1)\}$, and thus re-evaluating the next message on the updated state of



(a) Content Router (b) Splitter (c) Enricher

Fig. 4: Timed db-net realizations

the persistence layer. Since the existing proofs do not cover arithmetic formulae such as the increment $x + 1$ [16], it has to be realized as incremented DB sequence.

B. Selected Patterns and Processing Semantics

An (iterative) Splitter [10], required for the case study, is a complex routing pattern that is able to construct new messages for each element in iterable of an incoming message <pre><iterable><post> of the form [<pre>] <iterable: $elem_i$> [<post>] with optional pre and post parts. The splitter can be realized completely in CPN as shown in Fig. 4(b). The entering message payloads in ch_0 are separated into its parts: pre, post and it. While the first two are remembered during the processing, the iterable it is iteratively split into its parts $it.next$ according to a criterion x and a message is created on ch_3 using the pre and post information.

The Content Enricher [10] in Fig. 4(c) either enriches the tokens from ch_0 locally or through request-reply transitions t_1, t_2 to an external service or resource. While the stateless enriching part is in CPN, the query on a stateful resource ch_3 requires db-net semantics (cf. REQ-4(a,b)). In addition to the specific pattern requirements, the message processing semantics of the EIPs describes one message (or token) at a time. Since PN and db-net transitions are able to process several tokens in the inbound control place P_c (i. e., from the inbound flow F_{in} Def. 1), the transition capacity has to be restricted to one.

C. Discussion

The db-net foundation implicitly covers REQs-2,4 in form of a relational formalization with database transactions. Together with the realizations of the Router, the Load Balancer (cf. REQ-1(a), (b)) and the Aggregator Fig. 1 (cf. REQs-3(a), REQ-4 and REQ-5) we showed realizations for all of the requirements from Sect. II. The expiry of tokens, depending on time information within the message, is in CPNs and db-net, when modeled as part of the token's color set and transition guards (similar to [24]). As the ratio function of the balancer, the transition timeouts (cf. REQ-3(a)) and delays (cf. REQ-3(c)) are in timed db-net. Similarly, the msg/time ratio (cf. REQ-3(d)) can be represented (cf. supplementary material).

V. Evaluation

in this section, we quantitatively evaluate the comprehensiveness of the timed db-net realizations against the real-world integration scenarios (including pattern composition cases), show how to show how to test the correctness of the derived EIP formalism for the requirements from Sect. II-B, and discuss their application to one of the scenarios (cf. Q3).

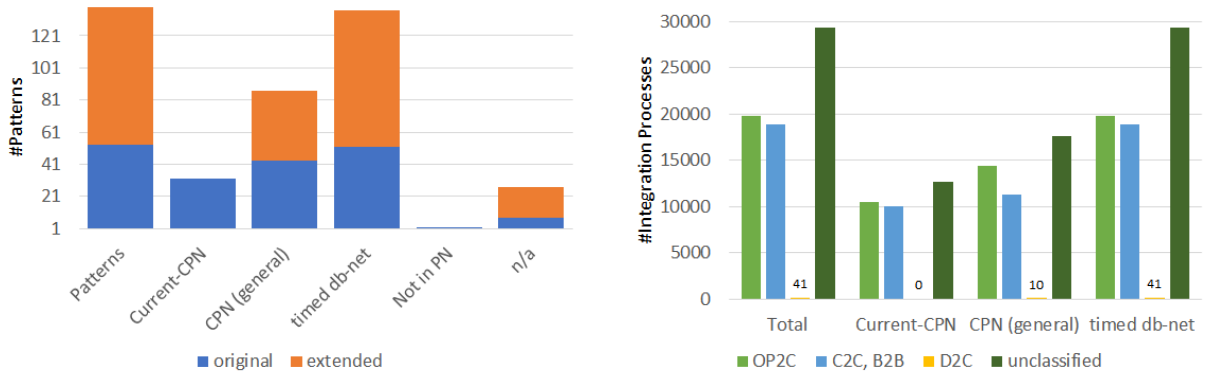
A. Comprehensiveness of timed db-net

The comprehensiveness of timed db-nets is evaluated with respect to coverage of the patterns in the catalogs depicted in Fig. 5(a). Here we compare the applicability of the existing CPN-based formalization [8] (`Current-CPN`), CPNs in general (`CPN (general)`) and timed db-nets (`timed db-net`). While the formalization proposed in [8] covers only some of the EIP from [10], many more EIPs as well as the recently extended patterns can be represented by CPNs. Now, with the timed db-net formalism proposed in this work one can nearly formalize all of the EIPs. However, there are two patterns that have requirements that cannot be represented, using Petri net classes discussed in this work, without further investigation (i. e., Dynamic Router, Durable Subscriber) due to their state or transition generating requirements.

While the first analysis targeted the pattern coverage of the formalisms, we now consider their relevance by the coverage of real-world integration scenarios. For this we implemented a Content Monitor pattern [19], which allows for the analysis of the actually deployed integration scenarios, e. g., running on SAP Cloud Platform Integration (SAP CPI)[3]. Figure 5(b) shows the coverage of the formalisms grouped by the following integration scenario domains, taken from [19]: On-Premise to Cloud (OP2C: known as hybrid integration), Cloud to Cloud or Business Network (C2C, B2B: native cloud applications), and Device to Cloud (D2C: incl. Mobile, IoT and Personal Computing) integration. Briefly, the results show that the current approach [8] is only partially sufficient to cover the OP2C, C2C and B2B scenarios. With a more general CPN approach, more than 70% of more conventional OP2C communication patterns can be covered. The more recent and complex cloud, business network and device integration requires timed db-net to a larger extend, which covers all analyzed scenarios.

Conclusions. (1) timed db-net is sufficient to represent most of the EIPs; (2) place or transition generating EIPs are not covered by considered PN classes; (3) hybrid integration requires less complex semantics and thus largely in CPN; (4) timed db-nets cover all of the current integration scenarios in SAP CPI.

[3]SAP CPI, visisted 05/2018: https://api.sap.com/shell/integration.



(a) Pattern coverage by formalization

(b) Realization by integration solution

Fig. 5: Timed db-net comprehensiveness

B. Simulation: Pattern Correctness Testing

With the preliminary model checking results, we prototypically implemented the db-net formalism so as to experimentally test the correctness of the pattern realizations through simulation. For this, we leverage results from Sect. III, which allow to trace the markings on the control and execution trace on the persistence layer to decide on the correctness of the realized pattern. This is due to the timed db-net execution semantics (cf. Def. 4) producing several B-snapshots $s_1, .., s_n$ during the execution of the pattern from an input B-snapshot $s_1 = \langle I_1, m_1 \rangle$ with database instance $s_n = \langle I_n, m_n \rangle$ to a final snapshot s_n, denoted by $s_1[t, \sigma\rangle s_2$,..,$s_i[t, \sigma\rangle s_n$, with transition $t \in T$ and binding σ. In case of a database instance I_j is not compliant with $\mathcal{P}$, then the execution stops and leaves the timed db-net in an intermediate state $I_j = I_{i-1}$, otherwise I_n is a valid final state. When considering the pattern's inner workings as unknown, with the data exchanged through input $ch_1, .., ch_i$, output $ch_{n-m}, .., ch_n$, and intermediate places ch_j in $\mathcal{N}$, together with the corresponding database instances $I_1, .., I_i$, $I_{n-m}, .., I_n$, and I_j, the control and data flows can be validated, by checking whether the pattern model produces a correct sequence of firing leading to B-snapshot with the correct final database instance and marking.

Prototype. To perform simulation, we developed prototypical modeling and simulation extensions for CPN Tools v4.0. As compared to other PN tools like Renew v2.5 [5], CPN tools supports third-party extensions that can address the persistence and data logic layers of db-nets. Moreover, CPN Tools handles sophisticated simulation tasks over models that use the deployed extensions. To support db-nets, our extension supports view places, SQL queries and actions, and realizes the full execution semantics of db-nets on top of a PostgreSQL database.

Simulation. Due to space constraints, we illustrate the correctness testing of the aggregator pattern realization in Fig. 1 through simulation in our CPN Tool extension, shown in Fig. 6 (available for download: https://bit.ly/2IGSoBW). Note that the timed completion condition is neglected due to differing temporal semantics in the tool. For the simulation, we added a table `Test_Messages` with four test

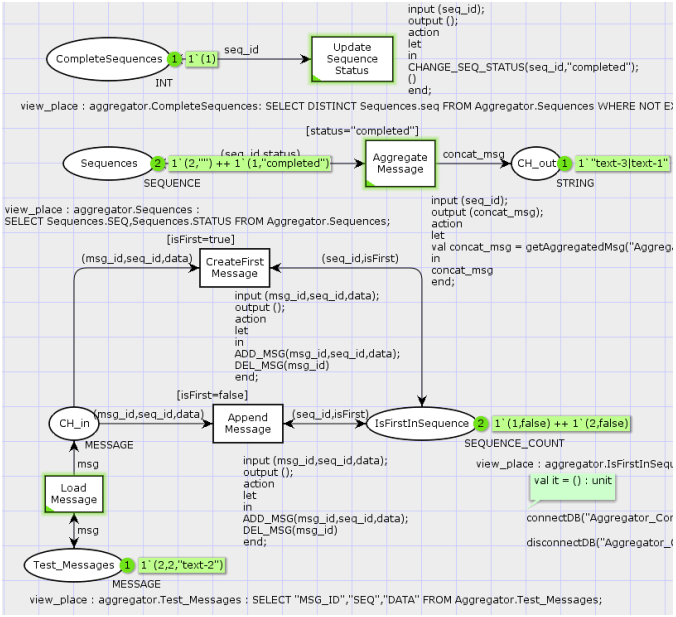


Fig. 6: Aggregator simulation (as CPN Tool extension)

messages, i.e., {`(1,1,'text-1')`, `(2,2,'text-2')`, `(3,1,'text-3')`, `(4,2,'text-4')`}, with ids from 1–4, two sequences {1, 2} and a textual payload. The completion condition is configured to aggregate after two messages of the same sequence and the aggregation function concatenates the message payloads separated by '|'. The expected result in the output place `CH_out` for the first sequence is one message with both payloads aggregated `(1,'text-3|text-1')`.

Now, when establishing a connection to the database and to the CPN tools extension server, the data from the connected database tables are queried and the net is initialized with the data from the database in form of tokens. Thereby, enabled transitions are highlighted by a green frame, indicating that they are ready to fire. When the transition is (manually) fired, the token is moved and the database state is read or updated accordingly, which potentially enables further transitions. With that, we simulated the aggregator realization in Fig. 6 for the two test sequences, until one sequence was complete, resulting

to the expected outcome in `CH_out` and the database.
Conclusions. (5) The CPN Tools extension allows for EIP simulation and correctness testing; (6) model checking implementations beyond correctness testing are desirable.

C. Applicability: Predictive Maintenance and Service (PDMS)

In the context of digital transformation, an automated maintenance of industrial machinery is imperative and requires the communication between the machines, the machine controller and ERP systems that orchestrate maintenance and service tasks. Integrated maintenance is realized by one of the analyzed D2C scenarios in Sect. V-A, which helps to avoid production outages and to track the maintenance progress. Thereby, notifications are usually issued in a PDMS solution as shown in Fig. 7 from SAP CPI, represented in BPMN according to [20].

Due to the lack of space, we simplified the scenario, e. g., only one machine `Machine A` sends alerts to PDMS, indicating that a follow-on action is required. The PDMS system creates alerts for the different machines (note the query Q_{alert} that returns the device id and the critical value act_val) and forwards them to a mediator, connecting the PDMS to the ERP system. Before the ERP notification can be created, additional data from the machines are queried based on the split and single alerts, and then enriched with information provided by query Q_{get} that adds the feature type $feat\_type$. The information of the single alerts is used to predict the impact and then aggregated to be sent to ERP. In case the notification has been created successfully in ERP, the PDMS gets notified including the service task ID and thus stops sending the alert (not shown). For this study, we manually encoded the BPMN scenario into a timed db-net as shown in Fig. 8. While the splitter is close to the current CPN solution in [8], the content enricher (incl. the query on the machine's state) and aggregators require timed db-net. Nevertheless, timed db-nets allow to represent patterns not covered before (e. g., the stateful aggregator with a timeout or the content enricher with external resources) and check their soundness and correctness. Note that for more complex scenarios the timed db-net representation might become very complex, e. g., compared to a BPMN representation, and thus might be more suitable as formalism and not as modeling language for the average user (e. g., integration developer).
Conclusions. (7) timed db-net representations allow for an explicit modeling of all data aspects in complex data-aware scenarios (e. g., roll-back, queries); (8) the formalism's technical complexity might prevent a usage as a modeling language.

D. Discussion

With the timed db-net formalization, it is possible to model and reason about EAI requirements like data, transacted resources and time (cf. conclusions (1), (5)), going beyond the simple hybrid integration scenarios (cf. conclusion (3)). Thereby the pattern realizations are self-contained, can be composed into complex integration scenarios (e. g., Fig. 8) and analyzed (cf. conclusions (4)), while leaving the extension of our tool prototype to model checking as future work (cf. conclusion (6)). The composition is facilitated through "sharing" control places, preventing unwanted side-effects between patterns.

However, there are some limitations that we briefly discuss next. PN classes considered in this work fall short when it comes to generation of places or arcs (cf. conclusion (2)). E.g., Dynamic Router requires a proper representation of dynamically added or removed channels, while Durable Subscriber is based on a changing infrastructure: if one receiver is off due to maintenance, it can still receive messages, which PNs cannot model. Further, the deep insights into data-aware patterns and scenarios lead to the trade-off between sufficient information and model complexity (cf. conclusion (7)). The complexity of PN models compared to their BPMN counterparts in Fig. 7 might not allow for modeling by non-technical users (cf. conclusion (8)). Hence, we propose modeling in a less technical modeling notation, which can be then encoded into PN models, e. g., for verification.

VI. Related Work

We found [19] that the only existing formalization of EIPs is provided in [8] using CPNs. In particular, [8] defines messages as coloured tokens and uses PN transition guards as conditions. However, it does not cover all requirements we singled out, and hence we employ db-nets [16] as an extension of CPNs that covers all but two of the EIPs as discussed before. In the business process domain PNs were sucessfully used to model and reason about workflow nets [25] and some resource- [15] and data-aware [6]) extensions, without however tackling EIP requirements. Although our predictive maintenance scenario has been captured in BPMN [20], we do not consider BPMN as a suitable formalism for our objectives (i)–(iii). We rather build on a formalization by PNs, which were actually also employed to define the BPMN control-flow semantics [7].

Using PNs, [26] defines an alternative approach for representing and reasoning on database transactions using special token vectors with identifiers and inhibitor nets. While this could also be used similar to db-nets, we build our formalism on db-nets due to their more comprehensive focus on (relational) data, operations, and persistent storage. Furthermore, there is work on ITCPN [24] and stochastic PNs [27] that are either too restricted by time intervals with a single global time in case of ITCPN or hard to practically reason as in case of the stochastic nets. However, both works helped during the specification of timed db-net. Stochastic PNs [27] define a priority function, whose execution semantics however does not suffice in representing the required ordering in REQ-1(a).

VII. Conclusion

To formalize the EIPs as the foundations of current EAI systems, this work collects relevant EAI requirements (cf. RQ-1), selects and combines existing PN approaches as timed db-nets (cf. RQ-2), then realizes selected EIPs using the formalism, and briefly sketches how to assess their correctness (cf. RQ-3). The evaluation results into several interesting conclusions, e. g., the suitability of our approach for EIPs and their compositions (cf. conclusions (1), (3)–(5), (7)), a model complexity trade-off

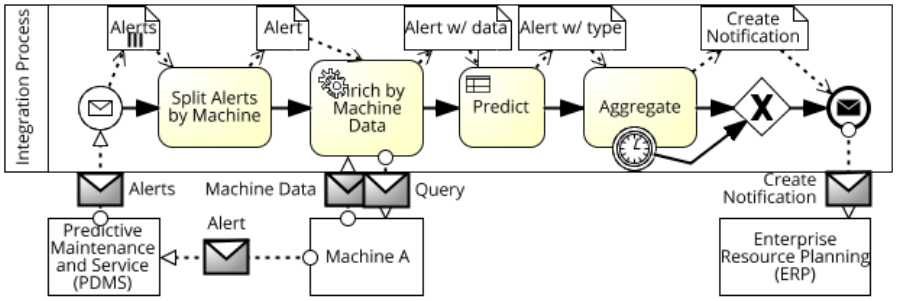


Fig. 7: Predictive Maintenance — Create Notification scenario as modeled by a user

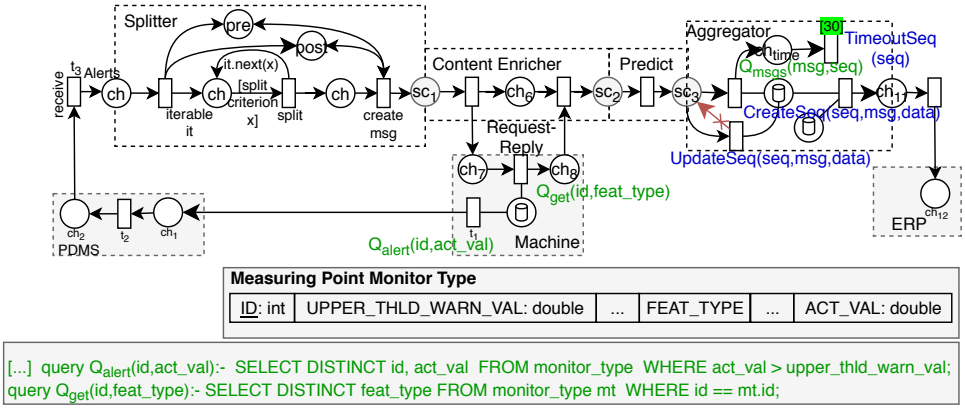


Fig. 8: Create Notification scenario translated into its timed db-nets representation (schematic)

(cf. conclusion (8)), and desirable extensions. Future work will target the extension of our concept and development of a prototype for model checking (cf. conclusion (6)) as well as further investigations of the model complexity trade-off through an automatic translation between user-friendly modeling environments and PNs.

References

[1] S. Akshay, B. Genest, and L. Hélouët. Decidable classes of unbounded petri nets with time and urgency. In *PN*, pages 301–322. Springer, 2016.

[2] E. Badouel, L. Hélouët, and C. Morvan. Petri nets with structured data.

[3] G. Balbo. Introduction to stochastic petri nets. In *FMPA*, volume 2090, pages 84–155. Springer, 2001.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Soft. Eng.*, 17(3), 1991.

[5] L. Cabac, M. Haustermann, and D. Mosteller. Renew 2.5 – towards a comprehensive integrated development environment for petri net-based applications. In F. Kordon and D. Moldt, editors, *Application and Theory of Petri Nets and Concurrency*, pages 101–112. Springer, 2016.

[6] R. De Masellis, C. Di Francescomarino, C. Ghidini, M. Montali, and S. Tessaris. Add data into business process verification: Bridging the gap between theory and practice. In *AAAI*, pages 1091–1099, 2017.

[7] R. M. Dijkman et al. Formal semantics and analysis of bpmn process models using petri nets. *QUT Tech. Rep.*, 2007.

[8] D. Fahland and C. Gierds. Analyzing and completing middleware designs for enterprise integration using coloured petri nets. In *CAiSE*, pages 400–416, 2013.

[9] J. Hidders et al. DFL: A dataflow language based on petri nets and nested relational calculus. *Information Systems*, 33(3):261–284, 2008.

[10] G. Hohpe and B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley, 2004.

[11] Y. Hu, S. Sundara, and J. Srinivasan. Supporting time-constrained sql queries in oracle. In *VLDB*, pages 1207–1218, 2007.

[12] L. Jacobsen, M. Jacobsen, M. H. Møller, and J. Srba. Sofsem. pages 46–72. Springer, 2011.

[13] S. Lasota. Decidability border for petri nets with data: WQO dichotomy conjecture. In *PN*, pages 20–36. Springer, 2016.

[14] D. S. Linthicum. *Enterprise Application Integration*. Addison-Wesley, 2000.

[15] M. Martos-Salgado and F. Rosa-Velardo. Dynamic soundness in resource-constrained workflow nets. In *Formal Techniques for Distributed Systems*, pages 259–273. Springer, 2011.

[16] M. Montali and A. Rivkin. Db-nets: On the marriage of colored petri nets and relational databases. *T. Petri Nets and Other Models of Concurrency*, 12:91–118, 2017.

[17] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *JMIS*, 24(3):45–77, 2007.

[18] D. Ritter. Database processes for application integration. In *BICOD*, pages 49–61. Springer, 2017.

[19] D. Ritter, N. May, and S. Rinderle-Ma. Patterns for emerging application integration scenarios: A survey. *Information Systems*, 67:36 – 57, 2017.

[20] D. Ritter and J. Sosulski. Exception handling in message-based integration systems and modeling using BPMN. *Int. J. Coop. Inf. Syst*, 25(2), 2016.

[21] F. Rosa-Velardo and D. de Frutos-Escrig. Decidability and complexity of petri nets with unordered data. *Theoretical Computer Science*, 412(34):4439–4451, 2011.

[22] J. Sifakis. Use of petri nets for performance evaluation. *Acta Cybernetica*, 4(2):185–202, 1980.

[23] M. Triebel and J. Sürmeli. Homogeneous equations of algebraic petri nets. *arXiv preprint arXiv:1606.05490*, 2016.

[24] W. M. van der Aalst. Interval timed coloured petri nets and their analysis. In *ICATPN*, pages 453–472, 1993.

[25] W. M. Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[26] K. M. Van Hee, N. Sidorova, et al. Generation of database transactions with petri nets. *Fund. Inf.*, 93(1-3):171–184, 2009.

[27] A. Zenie. Colored stochastic petri nets. In *International Workshop on Timed Petri Nets*, pages 262–271, 1985.

[28] W. Zuberek. D-timed petri nets and modeling of timeouts and protocols. *Trans. Soc. Comp. Simul.*, 4(4):331–357, 1987.