On DB-nets and their Applications

Marco Montali and Andrey Rivkin

Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
montali,rivkin@inf.unibz.it

1 Introduction

The recent developments in the Business Process Management (BPM) community demonstrate a paradigmatic shift in the way complex systems are perceived [3, 10, 2]. Now, the "language" for describing such systems should not only consider both processes and (master) data dimensions, but also should be expressive enough to talk about their interplay.

The recently introduced formalism of DB-nets [8] is an example of such language. DB-nets provide a new conceptual way of modelling complex dynamic systems that equally account for the aforementioned dimensions, and where the data dimension considers both local and persistent data. To correctly represent process and data dimensions in one model, DB-nets combine two conventional approaches such as coloured Petri nets (CPNs) with name creation and management, and relational databases. More specifically, in a DB-net: *(i)* master data are represented using full-fledged relational databases with constraints; *(ii)* the process logic as well as local data are captured using a variant of CPNs extended with special places whose content corresponds to a view on top of the underlying database; *(iii)* the task logic conceptually defines how the underlying database is updated. In this short paper we briefly introduce the formalism of DB-nets, showcase its currently existing applications and briefly discuss its future perspectives.

2 The Formalism

Here we provide a simplified definition of a DB-net by formalizing intro thee conceptual layers the three abstractions described above. For a more detailed definition refer to [8].

Definition 1. *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 pairwise disjoint* data types $\mathcal{D} = \langle \Delta_{\mathcal{D}}, \Gamma_{\mathcal{D}} \rangle$*, where* $\Delta_{\mathcal{D}}$ *is a* value domain*, and* $\Gamma_{\mathcal{D}}$ *is a finite set of domain-specific (rigid)* predicates.
- $\mathcal{P}$ *is a* $\mathfrak{D}$*-typed persistence layer – a pair* $\langle \mathcal{R}, \mathcal{E} \rangle$ *where: (i)* $\mathcal{R}$ *is a* $\mathfrak{D}$*-typed database schema, i.e., a set of* $\mathfrak{D}$*-typed relation schemas* $R(\mathcal{D}_1, \ldots, \mathcal{D}_n)$*, with* $\mathcal{D}_i \in \mathfrak{D}$ *for* $i \in \{1, \ldots, n\}$*; (ii)* $\mathcal{E}$ *is a finite set* $\{\Phi_1, \ldots, \Phi_k\}$ *of* $\mathtt{FO}(\mathfrak{D})$[1] *sentences (or queries) over* $\mathcal{R}$*, modelling* constraints over $\mathcal{R}$*.*

[1] First-order (FO) logic extended with data types.

- $\mathcal{L}$ *is a* $\mathfrak{D}$*-typed data logic layer over* $\mathcal{P}$ *– a pair* $\langle \mathcal{Q}, \mathcal{A} \rangle$*, where* $\mathcal{Q}$ *is a finite set of* $\texttt{FO}(\mathfrak{D})$ *queries over* $\mathcal{R}$*, and* $\mathcal{A}$ *is a finite set of parametric atomic actions specifying which facts to delete from (and/or to add to) the persistent storage.*[2]
- $\mathcal{N}$ *is a* $\mathfrak{D}$*-typed control layer – a tuple* $\langle P, T, F_{in}, F_{out}, F_{rb}, \texttt{color}, \texttt{query}, \texttt{guard}, \texttt{act} \rangle$*, such that:*
  - $P = P_c \cup P_v$ *is a finite set of* places *partitioned into control places* P_c *and view places* P_v *(decorated as*  *, connect to transitions only with read arcs).*
  - T *is a finite set of* transitions*, such that* $T \cap P = \emptyset$*.*
  - F_{in} *is an input flow from* P *to* T *assigning multisets of inscriptions (over* $\mathfrak{D}$*-typed variables) to input arcs.*[3]
  - F_{out} *and* F_{rb} *are respectively an* output *and* rollback flows *from transitions* T *to places* P *assigning multisets of inscriptions to output arcs.*
  - `color` *is a color type assignment over* P*, mapping each place* $p \in P$ *to a cartesian product of* $\mathfrak{D}$*-types.*
  - `query` *is a query assignment mapping each view place* $p \in P_v$ *to a query* $Q \in \mathcal{Q}$*, s.t. the color of* p *component-wise matches with the types of the free variables in* Q*.*
  - `guard` *is a transition guard assignment over* T *assigning to each transition* $t \in T$ *a* $\mathfrak{D}$*-typed guard* φ *(i.e., a quantifier- and relation-free* $\texttt{FO}(\mathfrak{D})$ *formula), that is defined over* t*'s input inscriptions.*
  - `act` *is a partial function assigning actions from* $\mathcal{A}$ *to transitions from* T*.*

While the input flow contains inscriptions that match the components of colored tokens present in the input places, the output/roll-back flow can also contain constants and special kind of variables called *fresh*, allowing to generate data values not already present in the net, nor in the underlying database instance. Elements of inscription tuples can be referenced in transition guards and action assignments (for instantiating action formal parameters with inscription bindings).

Example 1. To demonstrate a simple DB-net model, let us consider a simplified accommodation booking process in a travel e-commerce website. Using the website, a user should be able to search for various options by specifying a city and a period of stay. As soon as a suitable option is found, she is offered to complete a booking form. Upon its completion, selected accommodation is getting booked. Note that the website supports multiple user sessions running at the same time. This can create a situation in which two users are completing forms for the same accommodation option, but one of them is faster. The slower user then loses her chance to get accommodation and has to search for another one.

The persistence layer stores background data as well as data that persist across cases. In our scenario the website database comprises two relation schemas: *Available*(ID : **int**, city : **string**, period : **string**) lists available accommodation options, whereas *Booked*(ID : **int**, city : **string**, period : **string**, data : **string**) lists the booked ones. Each relation is equipped with a primary key constraint defined on ID attributes.

We use view places to expose a portion of the persistence layer in the control layer, so that each token in every view place represents one of the answers produced by the query attached to the place. Such tokens are not directly consumed, but only read by transitions, so as to match

[2] As in STRIPS, we assume that when the same fact is asserted to be added and deleted during the same step, the higher priority is given to the addition.

[3] An inscription is a tuple $\langle e_1, ..., e_n \rangle$ of $\mathfrak{D}$-typed elements, where each e_i can be either a variable or a constant.

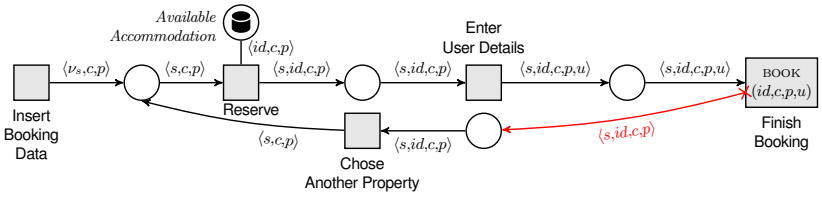

Fig. 1. The control layer of a DB-net for online booking. Here, ν_s is a fresh variable corresponding to a newly created session on the website, whereas c and p are two unbounded variables simulating user input for selected city and period of stay. The rollback output arc (corresponds to the rollback flow) is in red and decorated with an "x".

the input inscriptions with query answers. In our scenario we would like to have access to available accommodations from the website database. To this end, we use a query that is formally defined as $\mathtt{Q}_{\mathtt{ava}}(id,c,p)$:-$Available(id,c,p)$. Its SQL counterpart is **SELECT** id, c, p **FROM** *Available*. This query is then assigned to view place *Available Accommodation* in Figure 1.

A transition in the control layer may bind its input inscriptions to the parameters of an action attached to the transition itself, thus providing a mechanism to trigger a database update upon transition firing (and, maybe, consequently change the content of view places). Here, the data logic layer provides a functionality for booking accommodation for specified period p using action BOOK(id,c,p,u) (with four formal parameters) that, upon execution, removes chosen accommodation with identifier id from the *Available* table, and then adds a new entry with the same id and customer data u to the *Booked* table. Formally it is specified using the following notation: BOOK$\cdot\mathtt{del} = \{Available(id,c,p)\}$ and BOOK$\cdot\mathtt{add} = \{Booked(id,c,p,u)\}$. In Figure 1, BOOK assigned to transition Finish Booking graphically appears in the grey transition box.

Note that Finish Booking has one rollback arc connected to it. This arc essentially models the aforementioned case of at least two users trying to book the same accommodation. Indeed, when consequently firing Finish Booking with two tokens carrying identical identifiers that correspond to the same accommodation option, the second triggered update of BOOK will violate the primary key constraint of *Booked*, and the net will follow the compensation flow.

Execution semantics. Let us briefly recall the execution semantics of DB-nets, that has to simultaneously capture the progression of both persistence and control layers. To this end, at each point in time, the persistence and control layers are respectively associated with database instance $\mathcal{I}$ and marking m, in which content of view places must be compatible with that of $\mathcal{I}$ (i.e., it is aligned via queries assigned to view places). We shall refer to this as a DB-net snapshot, denoted as $\langle \mathcal{I},m \rangle$. Next we informally define a transition enablement and a transition firing in a given snapshot.

By analogy with CPNs, the firing of a transition t in a snapshot is defined w.r.t. a so-called binding σ for t that substitutes all variables in the inscriptions on the arcs incident to t and, possibly, formal parameters of an action signature assigned to t with values from $\mathfrak{D}$. However, to properly enable the firing of t, the binding σ must satisfy three conditions: *(i)* there should be enough of tokens that mach inscriptions on the corresponding input arcs; *(ii)* the guard attached to t has to be satisfied; *(iii)* all fresh variables should be substituted with values that are pairwise distinct, and also distinct from all the values present in the current marking, as well as in the current database instance.[4] Now, if a transition is enabled, it can be fired. The firing, instead, has a three-

[4] Fresh variables is a typed analogue of ν-variables of ν-Petri nets [12]. They can appear only in actions as well as inscriptions of output and rollback flows.

fold effect. First, all tokens in control places P_c are consumed according to matching input inscriptions. Second, the instantiated action assigned to t is applied on the current database instance $\mathcal{I}$. Since actions are atomic (i.e., the respect transactional semantics), one proceeds as follows. If the application is successful (i.e., the resulting instance of the persistence storage satisfies the constraints from $\mathcal{E}$), the database instance is updated (*commit*); if not, it is kept unaltered (*rollback*). Third, tokens are populated in target places according to output arc inscriptions and an output flow, that is going to be F_{out} in the case of commit and F_{rb} otherwise. Note that the latter is virtually an example of how a net can alter its behavior based on the manipulations with the persistent data.

All in all, the execution semantics of a DB-net is captured by a possibly infinite-labeled state transition system that accounts for all possible executions starting from their initial markings. While transitions in such LTSs model the effect of firing nets under given bindings, their states are represented with DB-net snapshots.

3 Current and Prospective Applications

DB-nets for EAI. [11] studies an application of DB-nets to Enterprise Application Integration (EAI). EAI defines a set of technologies and services for integrating various applications in an enterprise using compositions of Enterprise Integration Patterns (EIPs) and their extensions. EIPs are adopted by various EAI system vendors in their proprietary integration scenario modelling languages. However, such languages are not grounded in any formalism, and thus may produce integration models that are prone to design flaws. To minimize the manual errors and allow for automatic analysis of the pattern correctness, EIPs should be formalised.

Our work revealed that more versatile modelling formalisms are in high demand and, given growing interest in complex enterprise scenarios in which several inter-related business processes are linked together via shared data objects and events, DB-nets are very appreciated thanks to the conceptual tradeoffs they realize. Moreover, it appeared that DB-nets exhaustively cover all the requirements for EIPs and their extensions mentioned in the most recent classification suggested in [11]. We demonstrated how to model EIPs using DB-nets and showed how such models can become operational in a prototype based on CPN Tools (`http://cpntools.org/`) and its extension library Access/CPN. Unfortunately, our work revealed that the verification of EIP models created in CPN Tools using the state exploration tool falls short, since the latter becomes non-operational in the presence of data generating third party extensions (i.e., those that populate data/tokens into the net model). In order to still guarantee some form of correctness, we opted for the validation via simulation. This, in turn, proved to be quite efficient since CPN Tools offers a range of analytic features (e.g., a generation of simulation performance reports) based on the simulation toolkit.

One of the drawbacks of this approach is that the functionality provided by CPN Tools and Access/CPN is rather limited and hampers fast and agile modelling of data-aware processes. For example, there is no approach that would allow for the on-the-fly specification of data acquisition functions (that, essentially, model data injection via unbounded variables appearing in output flows as well as action formal parameters of DB-nets). They must be implemented per DB-net model directly in the Java code of its extension. In order to overcome such limitations one could use Renew (`http://www.`

`renew.de/`), proviso that modelling and simulation remain the main objectives. Notably, Renew supports high-level Petri nets and provides tighter integration with Java.

Formal Verification. It is easy to see that our formalism is Turing-complete. Nevertheless, given that DB-nets conceptually separate different aspects of a dynamic system, the formalism itself becomes an interesting model for fine-grained studies on how such aspects impact on undecidability and complexity of verification tasks, and how should they be controlled to guarantee decidability/tractability. For example, it is known that (un)decidability of reachability can be affected by the presence of ordered vs. unordered data types as well as (globally) fresh inputs [6], or by the presence of of negation in the queries used to inspect the persistence layer as well as the arity of relation schemas contained in it [1]. Notably, DB-nets do not only provide a comprehensive model to fine-tune all such parameters, but also allow to study how they interact with each other. In the same vein, we consider of particular importance the case where a DB-net obeys to the state-boundedness property. Indeed, under certain boundedness restrictions that apply both to the database and the net (a state-bounded DB-net is still allowed to visit infinitely many different snapshots along its runs), one could show that by following a similar procedure used in [7], decidability results are derivable for model checking properties expressed in first-order variant of μ-calculus. Alternatively, one can leverage results on the verification of infinite-state systems using Satisfiability Modulo Theories (SMT) techniques. While these techniques typically only support verification of (variants of) safety properties, a large amount of available tools can be used for testing DB-net encodings in different FO theories. We are currently working on the realization of both ideas. It would be also interesting to study how to check or guarantee, using modeling strategies, that a DB-net is state-bounded.

In [9] we have shown that one can isolate a fragment of DB-nets (with the querying language being restricted to SELECT-FROM-WHERE SQL queries with WHERE clauses using only conjunctions of atomic formulas) for which there exists a bisimilar class of CPNs with prioritized transitions, name creation and management, and provided a translation for constructing the latter. Even though such class of CPNs differs from the more conventional one of Jensen [4] by allowing variables range over infinite domains, one can realize the injection of possibly fresh data values (the way it is done in DB-nets) directly in CPN Tools using the Comms/CPN library. Note that one can then exploit the translation to automatically construct a bisimilar CPN and inspect its state space using CPN Tools, proviso that the generated state space is finite. The finitness can be achieved by implementing a special abstraction technique directly in the net.

Finally, the formalism of DB-nets paves the way towards the formal analysis of additional properties, which only become relevant when CPNs are combined with relational databases. In particular two families of properties could be of high interest. The first is related to rollbacks, so as to check whether it is always (or never) the case that a transition induces a failing action. The second is related to the true concurrency present in a DB-net, which may contain transitions that appear to be concurrent by considering the control layer in isolation, but have instead to be sequenced due to the interplay with the persistence layer (and its constraints).

Modelling and Beyond. From the modeling point of view, DB-nets incorporate all typical abstractions needed in data-aware business processes. And existing tool support makes this formalism even more attractive for scenarios that also require

simulation. For example, considering that a simulation of DB-nets produces a database instance populated by executing the control layer (and thus implicitly reflecting its footprint), the formalism could provide novel insights into the problem of data benchmarking [5], especially in the context of data preparation for process mining. Another interesting scenario has been recently proposed by Lomazova and Carrasquel, in which they aim at using a variant of DB-nets for modelling and validating trading systems. Interestingly, as the basis for their validation approach, they suggest to use simulation together with conformance checking from the domain of process mining. The latter is very challenging as it considers a multi-perspective approach in which one has to take into account the interplay between the net in the control layer (including its local data) and the persistent storage.

4 Conclusions

In this paper we provided a short summary of the formalism of DB-nets. This formalism can be seen as the marriage of colored Petri nets and relational databases, and can be used for modelling, enactment and verification of data-aware processes. We also discussed current and prospective applications. Given the preliminary theoretical results as well as studied use cases, we believe that the formalism could find multiple applications in different modelling and simulation settings, and also could be investigated towards more fine-grained verification scenarios.

References

1. Abdulla, P.A., Aiswarya, C., Atig, M.F., Montali, M., Rezine, O.: Recency-bounded verification of dynamic database-driven systems. In: Proc. of PODS. ACM Press (2016)
2. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data aware process analysis: A database theory perspective. In: Proc. of PODS (2013)
3. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: Proc. of ODBASE. pp. 1152–1163 (2008)
4. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
5. Lanti, D., Rezk, M., Xiao, G., Calvanese, D.: The NPD benchmark: Reality check for OBDA systems. In: Proc. of EDBT. pp. 617–628. OpenProceedings.org (2015)
6. Lasota, S.: Decidability border for petri nets with data: WQO dichotomy conjecture. In: Proc. of PN. LNCS, vol. 9698, pp. 20–36. Springer (2016)
7. Montali, M., Rivkin, A.: Model checking petri nets with names using data-centric dynamic systems. Formal Aspects of Computing pp. 1–27 (2016)
8. Montali, M., Rivkin, A.: DB-Nets: on the marriage of colored Petri Nets and relational databases. Trans. of Petri Nets and other Models of Concurrency **28**(4) (2017)
9. Montali, M., Rivkin, A.: From db-nets to coloured petri nets with priorities. In: Proc. of ICATPN. pp. 449–469 (2019)
10. Reichert, M.: Process and data: Two sides of the same coin? In: Proc. of OTM. pp. 2–19 (2012)
11. Ritter, D., Rinderle-Ma, S., Montali, M., Rivkin, A., Sinha, A.: Formalizing application integration patterns. pp. 11–20 (2018)
12. Rosa-Velardo, F., de Frutos-Escrig, D.: Name creation vs. replication in petri net systems. Fundam. Inform. **88**(3), 329–356 (2008)