Information Systems Modeling: Language, Verification, and Tool Support

Artem Polyvyanyy[1], Jan Martijn E. M. van der Werf[2], Sietse Overbeek[2], and Rick Brouwers[2]

[1] The University of Melbourne, Parkville, VIC, 3010, Australia
artem.polyvyanyy@unimelb.edu.au
[2] Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands
{j.m.e.m.vanderwerf,s.j.overbeek}@uu.nl,
r.a.c.m.brouwers@students.uu.nl

Abstract. Information and processes are both important aspects of information systems. Nevertheless, most existing languages for modeling information systems focus either on one or the other. Languages that focus on information modeling often neglect the fact that information is manipulated by processes, while languages that focus on processes abstract from the structure of the information. In this paper, we present an approach for modeling and verification of information systems that combines information models and process models using an automated theorem prover. In our approach, set theory and first-order logic are used to express the structure and constraints of information, while Petri nets of a special kind, called Petri nets with identifiers, are used to capture the dynamic aspects of the systems. The proposed approach exhibits a unique balance between expressiveness and formal foundation, as it allows capturing a wide range of information systems, including infinite state systems, while allowing for automated verification, as it ensures the decidability of the reachability problem. The approach was implemented in a publicly available modeling and simulation tool and used in teaching of Information Systems students.

Keywords: IS modeling, verification of IS models, tools for IS modeling.

1 Introduction

An *information system* is an organized collection of concepts and constraints for storing, manipulating, and disseminating information. Finding the right balance between concepts and constraints for specifying static and dynamic aspects is essential when designing an information system. However, existing modeling languages often focus on one of the two aspects, leaving the other to play the second fiddle. Many information modeling notations introduce concepts to capture and verify domain constraints, but neglect that information is populated through processes. Similarly, process modeling languages often contain dedicated constructs to represent information/data, e.g., documents and messages, and data/information flows, but are of limited help when specifying beyond trivial information constraints imposed by the domain.

This work is motivated by the need, as witnessed by research in the last decade [6, 10, 18, 27], for theoretical and practical languages, methods, and tools to effectively integrate processes driven and information managed by information systems, as well as on our experiences in teaching information systems [33]. We propose a language for conceptual modeling of information systems that fulfills these requirements:

R1 Can be used to model concepts and constraints that govern the aspects related to the information that a system can manage, i.e., data, and the semantics of the data, that the processes of the system can manipulate;
R2 Can be used to model concepts and constraints that govern the aspects related to the dynamic behavior that a system can exercise, i.e., processes that manipulate the information managed by the system;
R3 Can be used to specify an aspect of an information system using information and/or process concepts and constraints;
R4 Has a formal foundation that allows automated verification.

The language we propose, called Information Systems Modeling Language (ISML), builds upon established formalisms for modeling process- and information-related concepts and constraints. Other criteria for assessing the quality of conceptual modeling languages, such as clarity, semantic stability, semantic relevance, and abstraction mechanisms (cf. [15]), are not considered in this work. These are addressed in isolation by the languages that constitute our formalism. Studies of manifestations of these criteria for the proposed overarching language are left for future work.

Requirements R1–R4 are standard for IS modeling languages [6, 10, 18, 27]. We use *mathematical modeling* and *formal proof* methods to develop a formalism that instantiates them in a *unique* way, as listed below (this claim is justified in Section 2):

I1 The create, read, update, and delete (CRUD) operations over information facts are supported, along with the expressiveness of the *first-order logic over finite sets with equality* for specifying information constraints;
I2 The process constraints of an information system, captured using *Petri nets with identifiers*, can induce a finite or countably infinite number of reachable states;
I3 An aspect of an information system can be captured using either process only, information only, or a combination of process and information concepts and constraints;
I4 The reachability problem, which given a model of an information system, its initial state, and some other state of the system consists of deciding if the information system can evolve from the initial into the given state, is computable.

These instantiations allow capturing a wide range of systems in a flexible way while ensuring a solid formal foundation. Instantiation I1 ensures standard support for CRUD operations over information facts and the ability to specify arbitrary constraints over them. Instantiation I2 ensures that the dynamic behavior of a captured system can be analyzed based on a wide range of semantics, including the interleaving/noninterleaving and linear/branching time semantics [29]. Consequently, the support of noninterleaving semantics necessitates the support for infinite collections of reachable states, as it often breaks the by-construction-guarantee of a bound on a number of reachable states. Instantiation I3 addresses the standard mechanism for balancing information and process concepts and constraints in models of information systems. We argue that instantiation I4 sets a solid formal foundation. For example, for the well-established formalism of Petri nets for describing distributed systems, many interesting verification problems were demonstrated to be recursively equivalent to the reachability problem [12]; these are the problems of liveness, deadlock-freedom, and several variants of the reachability problem, e.g., submarking reachability, zero reachability, and single-place zero reachability. The in this work presented reachability result is yet to be capitalized on in future studies to extend the repertoire of decidable verification problems for ISML models.

The next section discusses related work. Section 3 presents our modeling language. Section 4 is devoted to the decidability of the reachability problem. Then, Section 5 discusses a proof-of-concept implementation of a tool that supports modeling and simulation of information systems captured using the proposed language and reports on a preliminary evaluation of the approach with a cohort of Information Systems students. The paper closes with conclusions and an outlook at future work.

2 Related Work

To identify existing techniques for modeling information systems, we looked into survey papers on the topic of integrated data and process modeling. The survey papers, in turn, were identified by first using Scopus to find papers with titles that contain strings "data-centric process", "data-aware process", "process-centric data", or "process-aware data", and have the subject area of Computer Science (18 papers), then taking only survey papers (2 papers), and finally including other survey papers that cite any of the papers related to data and process modeling among the initially identified 18 papers. This procedure resulted in four identified survey papers, concretely [6, 10, 18, 27]. In what follows, we discuss those techniques included in the identified surveys that were assessed to deliver the best balance of expressiveness and verifiability. We classify the techniques based on their origin in one of the requirements R1 or R2 from Section 1. The discussions of languages for capturing exclusively process or exclusively information concepts and constraints are omitted, because of the space limitations. Hence, for instance, Entity Relationship diagrams or Object Role Model diagrams, as well as Petri nets, reset nets, or transfer nets, are not discussed.

Data-aware process models. The core formalism for describing data-aware processes is arguably colored Petri nets (CPNs). CPNs extend classical Petri nets by equipping each token with a data value, or color, which can be of an arbitrarily complex type [19]. For CPNs, reachability is undecidable unless the finiteness of the color domain is imposed. In [1], CPNs were used for modeling process-aware information systems. This instantiation of CPNs allows token manipulations to be captured as arbitrary programs, which benefits expressiveness but hinders analysis, as reachability stays undecidable.

In a Petri net with data, every token carries a data value and executions of transitions depend on and augment values associated with tokens. If data values are tested only for equality, like in the case of ν-PNs, the reachability problem is undecidable [28]. However, coverability, termination, and some boundedness problems are decidable for ν-PNs. The coverability, termination, and boundedness are decidable if in addition to the equality testing data values are drawn from a totally ordered domain [21]. However, the reachability problem remains undecidable even under this additional constraint [20].

In [9], the authors propose another model, called RAW-SYS, that combines Petri nets with relational databases. A RAW-SYS may induce an infinite state transition system, which complicates the analysis. In fact, the authors indeed conclude that, unless one limits the number of objects that can co-exist in a reachable state, the reachability problem is undecidable. Note that we do not impose this requirement on our models.

In [8], the authors integrate Petri nets, first-order logic, and specifications of how nets update data populations. Although closely related, this approach is limited to workflows represented as classical Petri nets only. The authors do not report any results on the decidability of verification problems for the proposed integrated modeling approach.

In [24], the authors take inspiration from [9] and propose a three-layered model of DB-nets. In a DB-net, the persistence layer maintains data values in a relational database, the control layer uses a variant of colored Petri nets to describe processes, and the data logic layer provides methods to extract and augment data values. The authors demonstrate that for a special class of bounded DB-nets that use string and real data types and may (despite the name) induce infinite collections of reachable states, the problem of reachability of a nonempty place is decidable.

Process-aware data models. A business artifact describes information about a business entity that evolves over time according to a well-defined lifecycle [26]. In [14], the authors study systems of artifacts that exhibit collective behaviors captured as Kripke structures and demonstrate that certain CTL properties are decidable for these systems when values of attributes and variables (scalars) that encode information managed by the system range over bounded or unbounded (but ordered) domains. Note that a Kripke structure cannot be used to encode the noninterleaving semantics of a system and can only be used to describe a finite number of reachable states. In [5], the authors study the problems of verifying whether an execution of a system can complete, the existence of an execution that leads to a dead-end, and redundancy of an attribute in a given system. These problems are shown to be decidable only under various restrictions, such as abstracting from actual attribute values or imposing restrictions on information manipulations, e.g., a value of an attribute is allowed to be modified at most once. The behavior of the overall system is captured as a set of declarative constraints that describe a collection of allowed executions and interpreted using the interleaving semantics. Differently from the approaches in [5,14], our formalism does not impose restrictions over domains or structure of values used to encode information facts.

In [25], the authors propose to use state transition systems to capture life cycles of data objects. The life cycles of such data objects are then linked according to the relationships between the objects. Consequently, such an integrated system is capable of describing only a finite number of states. A similar approach is followed in artifact-centric modeling [13]. Each artifact has a life cycle, represented by a state machine that manipulates a data model via OCL. Verification may not always terminate, and, as shown in [7], verification is only possible in limited cases.

In [11], the authors present some decidability results on verification of a rich artifact model that surpasses the previous work from IBM on artifact systems at expressiveness. However, the results are obtained under eight restrictions, which limit the management of data and recursive computation. The authors demonstrate that lifting any of the eight restrictions leads to undecidability of the verification. In [4], the authors formalize artifact systems as multi-agent systems and study the decidability of the problem of verifying some temporal logic properties. The authors state the undecidability of the problem for the general class of systems and derive at the decidability result for the subclass of systems whose behavior does not depend on the data values in reachable states.

A relational transducer [3,30] based on Gurevich's Abstract State Machine (ASM) is a relational database along with an ASM that governs management of the database. The problems of verifying temporal properties, log equivalence, and finite log validation are undecidable for transducers [30]. Some decidability results were obtained by a priori limiting the number of possible relations in each transducer state, limiting the number of database changes, and restricting the behavior of the transducers [3,30]. Active XML

(AXML) is an extension of XML with embedded service calls [2]. Some decidability results for AXML models for verifying data and process related properties were shown to be decidable under several restrictions, e.g., by ensuring a static bound on the total number of function calls in an execution of the system.

In [16], the authors address verification of data-centric dynamic systems (DCDSs). In a DCDS data is maintained in a relational database, while the process is captured as a set of condition-action rules that govern updates in the database. A DCDS can induce infinite collections of reachable states. As shown in [16], verification, in terms of some temporal logic properties, is undecidable in general and becomes decidable under constraints over data values in the reachable states.

Summary. None of the existing formalisms for describing process and information aspects of information systems is capable of describing an infinite amount of states while imposing no bounds on the values of governed information facts and enjoying the decidability of the reachability problem. Hence the work at hand to address the gap. In addition, our formalism supports CRUD operations over information facts and can be interpreted using interleaving/noninterleaving and linear/branching time semantics.

3 Information Systems Modeling Language

The language we propose has three constituents: an information model to describe the domain, a Petri net with identifiers to describe dynamic processes, and a specification defining how the processes manipulate information. In the remainder, we use the following running example to demonstrate the proposed language.[1]

Running Example. The educational institute "Private Teaching Institute" (PTI) offers different education tracks, such as Information Sciences and Computer Science. Each track at PTI has a small team, called the track management team, and a small student administration for all tracks together. For each track, different courses can be followed. Every person is entitled to register for a track. Once registered, and accepted by the track management, a person becomes a student of that track. A student accepted for a track must create a study plan, consisting of the courses she wants to follow. This plan has to be approved by the track management. Students enroll for courses. A student of a track is allowed to follow up to two courses concurrently. A lecturer decides whether a student fails or passes the course. In case a student fails, she is allowed to retake the course, until she passes it. Once the student passed all courses approved upon in the study plan, the student can request a diploma for that track. The track management verifies the certificates and the plan, after which they award the diploma.

3.1 Information Models

Many languages are available that satisfy the goal of requirement R1 to govern information and its manipulations, such as ERDs, UML class diagrams, and ORM diagrams. Each notation comes with its constructs and ways to express constraints. Yet, all these notations are similar in that they are founded in set theory and first-order logic. In ISML, we do not advocate the use of specific notations, but rather focus on the underlying principles. An *information model* consists of a set of possible entity types and

[1] Related materials can be found at: http://informationsystem.org/ismsuite/.

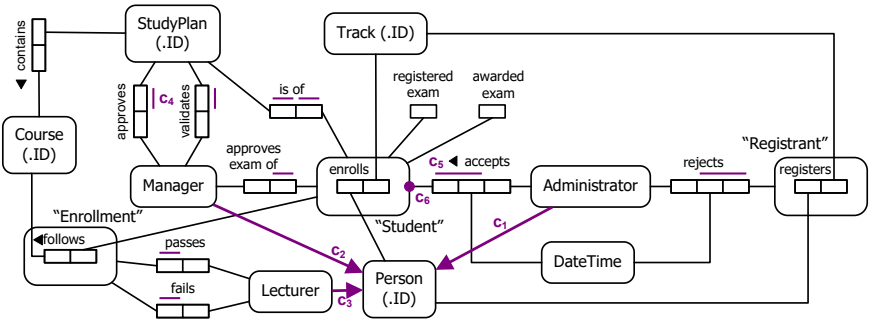


Fig. 1. An information model of the running example in ORM notation.

relations, which are characterized by finite sequences of entity types, together with a set of conditions, specified in first-order logic on finite sets with equality.

Let $\mathcal{I}$ and Λ be a universe of *identifiers* and a universe of *labels*, respectively.

Definition 3.1 (Information model)
An *information model* is a 4-tuple (E, R, ρ, Ψ), where:

- $E \subseteq \mathcal{P}(\mathcal{I})$ is a finite set of *entity types*;
- $R \subseteq \Lambda$ is a finite set of *relation types*;
- $\rho : R \to E^*$ is a *relation definition function* that maps every relation type onto a finite sequence of entity types for which it holds that for every $e \in E$ there exists $r \in R$, called the *entity relation* of e, such that $\rho(r) = \langle e \rangle$; and
- Ψ is a collection of *constraints* defined as a formal theory of the first-order logic statements over a collection of predicates that for every $r \in R$ contains a predicate with the domain $\prod_{i=1}^{|\rho(r)|} \rho(r)(i)$. ⌟

An information model of our running example in ORM notation is shown in Fig. 1. Boxes with rounded corners denote entity types, while rectangles stand for relation types, or facts using the ORM terminology. The diagram allows for traceability between the visual notation and the formalism. Note that in classical ORM, the running example would normally be captured using value types and objectified fact types, refer to [15].

Each entity and relation type of an information model is identified by a label and a corresponding sequence of entity types. For example, entity type `Person` is characterized by entity relation $Person$ and the sequence of entity types $\rho(Person) = \langle \texttt{Person} \rangle$. To indicate that a person can enroll into a track, one can define relation *enrolls* such that $\rho(enrolls) = \langle \texttt{Person}, \texttt{Track} \rangle$. Fig. 2 gives the relation definition function of the running example (without the entity relations).

An information model can be instantiated with entities and relations, called *facts*, which define its *population*. Every population is typed, i.e., every relation obeys its definition given by the relation definition function.

Definition 3.2 (Population, Fact)
A *population* of an information model (E, R, ρ, Ψ) is a function $\pi : R \to \mathcal{P}(\bigcup_{n \in \mathbb{N}} \mathcal{I}^n)$ such that every element in the population is correctly typed, i.e., for every $r \in R$ it holds that $\pi(r) \in \mathcal{P}(\prod_{i=1}^{|\rho(r)|} \rho(r)(i))$.[2] An element in $\pi(r)$ is called a *fact*. ⌟

[2] $\mathbb{N}$ denotes the set of all natural numbers, i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$, set $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$.

$\rho(\mathit{registers}) = \langle \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{rejects}) = \langle \texttt{Administrator}, \texttt{DateTime}, \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{accepts}) = \langle \texttt{Administrator}, \texttt{DateTime}, \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{enrolls}) = \langle \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{isOf}) = \langle \texttt{StudyPlan}, \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{contains}) = \langle \texttt{StudyPlan}, \texttt{Course} \rangle$
$\rho(\mathit{approves}) = \langle \texttt{Manager}, \texttt{StudyPlan} \rangle$

$\rho(\mathit{follows}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course} \rangle$
$\rho(\mathit{fails}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course}, \texttt{Lecturer} \rangle$
$\rho(\mathit{passes}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course}, \texttt{Lecturer} \rangle$
$\rho(\mathit{registeredExam}) = \langle \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{validates}) = \langle \texttt{Manager}, \texttt{StudyPlan} \rangle$
$\rho(\mathit{approvesExamOf}) = \langle \texttt{Manager}, \texttt{Person}, \texttt{Track} \rangle$
$\rho(\mathit{awardedExam}) = \langle \texttt{Person}, \texttt{Track} \rangle$

Fig. 2. Relation definition function for our running example, also visualized in Fig. 1.

Domain constraints are captured as first-order logic statements that define the formal theory of the information model. Based on the structure, one can distinguish various classes of constraints. In the context of the running example, we discuss several classes of constraints. In the remainder, let (E, R, ρ, Ψ) be an information model.

Subtyping Each entity of one type ($X \in E$) belongs to another type ($Y \in E$) iff $\forall_{x \in \mathcal{I}} : [x \in X \Rightarrow x \in Y]$. In Fig. 1, arrows c_1, c_2, and c_3 capture subtype constraints.

Uniqueness A combination of elements in a tuple is unique within a population. In Fig. 1, c_5 specifies that the last three elements of a tuple in *accepts* are unique: $\forall_{x,y,z,u,v \in \mathcal{I}} : [((x, z, u, v) \in \mathit{accepts} \wedge (y, z, u, v) \in \mathit{accepts}) \Rightarrow x = y]$.

Mandatory An element or fact must take part in another fact. For example, constraint c_6, denoted by a small filled circle in Fig. 1, specifies that *enrolls* must appear in *accepts*: $\forall_{x,y \in \mathcal{I}} : [\exists_{u,v \in \mathcal{I}} : [(x, y) \in \mathit{enrolls} \Rightarrow (u, v, x, y) \in \mathit{accepts}]]$.

Domain-specific constraints that do not fall into predefined categories, like those listed above, for which typically no corresponding graphical notations exist. For example, administrators are not allowed to cheat: $\forall_{x,y,z \in \mathcal{I}} : [(x, y, x, z) \notin \mathit{accepts}]$, i.e., an administrator cannot accept herself for a track.

A population may invalidate the constraints. Thus, we say that a population π is *valid* if it satisfies all the constraints of the information model, denoted by $\pi \models \Psi$; otherwise the population is *invalid*. By $\Pi(D)$ and $\Lambda(D)$ we denote the set of all possible populations of information model D and the set of all possible valid populations of D, respectively.

The population of an information system changes frequently. Entities and facts can be added, deleted, or updated. We define two operations for manipulating populations: inserting entities into a relation and removing entities from a relation. Note that an update can be interpreted as a delete followed by an insert.

Definition 3.3 (Transaction)
Let $D = (E, R, \rho, \Psi)$ be an information model. Let $r \in R$ be a relation, let $v \in \prod_{i=1}^{|\rho(r)|} \rho(r)(i)$ be a fact, and let $\pi \in \Pi(D)$ be a population. An *operation* o is a tuple $o \in \mathcal{O}(D)$ with $\mathcal{O}(D) = \big(R \times \{\oplus, \ominus\} \times \bigcup_{n \in \mathbb{N}} \mathcal{I}^n\big)$.

- Operation $o = (r, \oplus, v)$ *inserts* fact v into r in π, i.e., it results in population $\pi' \in \Pi(D)$, denoted by $(D : \pi \xrightarrow{r \oplus v} \pi')$, iff $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \cup \{v\})\}$.
- Operation $o = (r, \ominus, v)$ *deletes* fact v from r in π, i.e., it results in population $\pi' \in \Pi(D)$, denoted by $(D : \pi \xrightarrow{r \ominus v} \pi')$, iff $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \setminus \{v\})\}$.

A *transaction* $s \in (\mathcal{O}(D))^*$ is a finite sequence of operations, such that every subsequent operation is performed in a population resulting from the previous operation. A transaction is *valid* if the starting and resulting populations are valid. ⌟

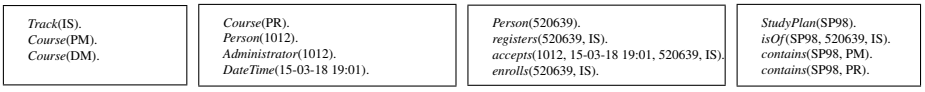


Fig. 3. A valid population for the running example.

When the context is clear, i.e., the scope of the information model and its current population are known, we write $\mathrm{insert}(r, v)$ and $\mathrm{delete}(r, v)$ instead of $(D : \pi \xrightarrow{r \oplus v} \pi')$ and $(D : \pi \xrightarrow{r \ominus v} \pi')$, respectively. A valid population of the information model of Fig. 2 is depicted in Fig. 3. Suppose student 520639 is working on her study plan. Updating the course Programming (PR) into Data Modeling (DM) can be expressed as follows:

$$\langle \mathrm{delete}(\mathit{contains}, (\mathrm{SP98}, \mathrm{PR})), \mathrm{insert}(\mathit{contains}, (\mathrm{SP98}, \mathrm{DM})) \rangle.$$

As the initial population is valid, and the result of executing the transaction will not violate any constraint, the above transaction is valid.

3.2 Process Models

Many different approaches for modeling processes exist that satisfy requirement R2. Each comes with its own notation and applications. For requirement R4 and its instantiation I4, a formal grounding of process modeling is required. Similar to [1], we utilize Petri nets to model processes. Notice that many languages can be translated into Petri nets, thus allowing tools to rely on a grounding formalism, while the modeler is using their own preferred modeling language. The Petri net in Fig. 4 reflects the process model of the running example.

Many analysis techniques for processes ignore data, i.e., tokens in places resembling the state of the process are considered to be indistinguishable. However, this results in an over-approximation of the possible firings, as shown with the following example: Starting with two tokens in place i, resembling two students, the model can eventually mark place *max concurrent courses* with four tokens, refer to Fig. 4; note that places with tokens encode all the corresponding entity instances currently kept in the population of the information model. Now, each student can start following one course by firing *register course*. As two tokens remain in place *max concurrent courses*, transition *register exam* remains enabled. However, if considering the students in isolation, this transition would not have been enabled.

The literature describes several approaches to address requirement R4, refer to Section 2. In ν-PN, tokens carry identifiers, while markings map places to bags of identifiers, indicating how many tokens in each place carry the same identifier.

In this paper, we extend the idea of tokens carrying identifiers to vectors of identifiers, to obtain Petri nets with identifiers (PNIDs). Vectors of identifiers have the advantage that a single token can represent multiple entities at the same time. In this way, a token may resemble a (composed) fact from a population of an information model.

In a PNID, each arc is labeled with a vector of variables. Similar to ν-PN, a valuation instantiates the variables to identifiers. The size of the vector on the arc is implied by the *cardinality* of the place it is connected to. Tokens carrying vectors of size 0 represent classical – black – tokens. If for a transition some variable only appears on outgoing arcs, it is called a *creator variable*. Let Σ denote a universe of variables.

Definition 3.4 (Petri net with identifiers)
A *Petri net with identifiers* (PNID) is a 5-tuple (P, T, F, α, β), where:

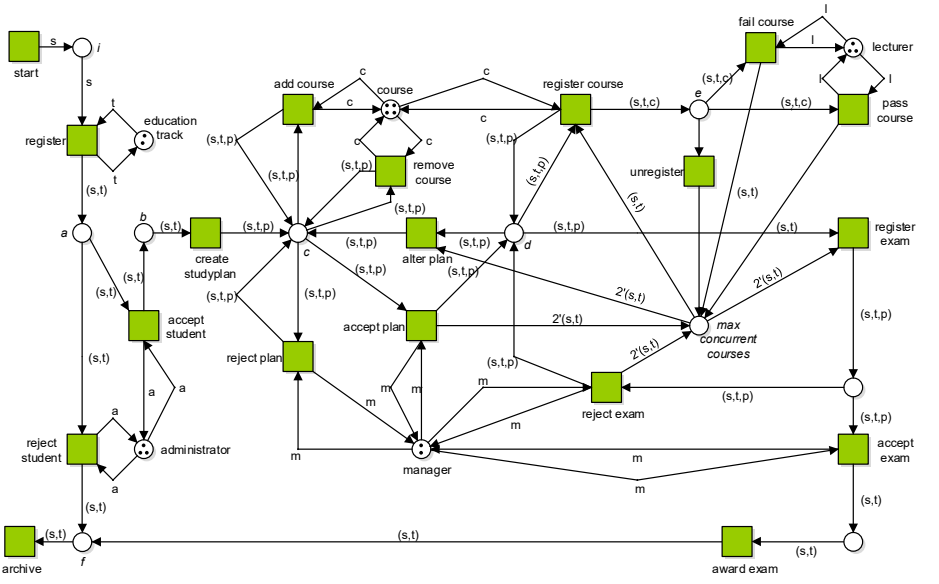


Fig. 4. PNID that describes processes students follow at PTI.

- (P, T, F) is a Petri net, with a set of places P, a set of transitions T, such that $P \cap T = \emptyset$, and a flow function $F : ((P \times T) \cup (T \times P)) \to \mathbb{N}^0$; if for $n, m \in P \cup T$, $F(n, m) > 0$, an arc is drawn from n to m;
- $\alpha \colon P \to \mathbb{N}^0$ defines the *cardinality* of a place, i.e., the length of the vector carried on the tokens residing at that place; its *color* is defined by $C(p) = \mathcal{I}^{\alpha(p)}$;
- β defines the *variable vector* for each arc, i.e., $\beta \in \prod_{f \in F} V_f$, where $V_{(p,t)} = V_{(t,p)} = \Sigma^{\alpha(p)}$ for $p \in P, t \in T$. ⌟

A marking of a PNID defines for each place the amount of tokens per vector identifier.

Definition 3.5 (Marking)
Given a PNID $N = (P, T, F, \alpha, \beta)$, its set of all possible markings is defined as $\mathcal{M}(N) = \prod_{p \in P} C(p) \to \mathbb{N}^0$. For $m \in \mathcal{M}(N)$, pair (N, m) is a *marked PNID*. ⌟

A transition is enabled if a *valuation* of variables to identifiers exists, such that each input place contains sufficient tokens with vectors of identifiers induced by the instantiated variable vector of the corresponding arc. The same valuation is used to determine which vectors of identifiers are produced in the output places. Note that a transition can only create new identifiers through variables that do not occur on its input arcs.

Definition 3.6 (Transition firing in a PNID)
Let (N, m) be a marked PNID with $N = (P, T, F, \alpha, \beta)$. Let *valuation* $\nu : \Sigma \to \mathcal{I}$ be an injective function that maps each variable to an identifier. A transition $t \in T$ is enabled in (N, m) if $[\nu(\beta((p, t)))^{F(p,t)}] \leq m(p)$ for all places $p \in P$. Its firing, denoted by $(N : m \xrightarrow{(t,\nu)} m')$, results in a marking m' with $m'(p) + [\nu(\beta((p, t)))^{F(p,t)}] = m(p) + [\nu(\beta((t, p)))^{F(t,p)}]$, for all places $p \in P$. ⌟

Details on the semantics of PNIDs are in [34]. Consider again the net in Fig. 4. This net is a PNID. Transition *start* creates a token with a single identifier, representing a person

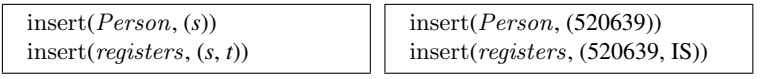


Fig. 5. Abstract transaction for transition *register*, and its instantiation with valuation $\{s \mapsto 520639, t \mapsto \text{IS}\}$.

entering PTI. Place *education track* contains all the tracks PTI offers. Firing transition *register*, models that some person s chooses a track t and registers for that track. The result is a token with a vector containing two identifiers: one for the person and one for the track. A token in place d resembles a student with an accepted plan. Similarly, place e represents students following a course, and carries three identifiers: the student (person and track), and the course.

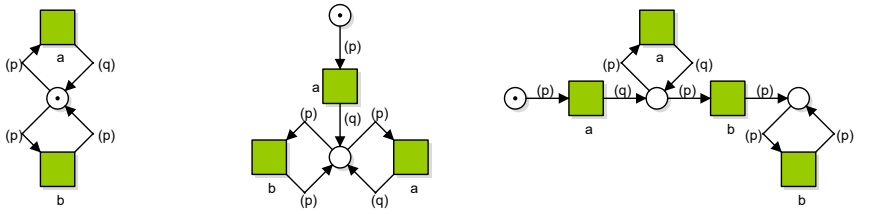
3.3 Information Systems Modeling Language

A transition firing can resemble some fact manipulation in the information model. Its firing requires a valuation that determines which identities can be used. For example, transition *register* resembles adding a fact to the population: $\text{insert}(register, (p, t))$, for some person p and track t. The intent of requirement R2 is to make this relation explicit. In our proposed formalism, each transition is specified with an *abstract transaction* that describes how the transition manipulates the population of the information model. Similar to transition firings in PNID, valuations are used to compute the transaction by instantiating the abstract transaction. For example, transition *accept student* from Fig. 4 can have the abstract transaction depicted in Fig. 5, that inserts two facts into a population: one to add the student as a person, and one to relate the person to the track.

Definition 3.7 (Abstract transaction)
Let $D = (E, R, \rho, \Psi)$ be a data model. An *abstract transaction* is a sequence of *abstract operations* $o \in \left(R \times \{\oplus, \ominus\} \times \bigcup_{n \in \mathbb{N}} (\Sigma \cup \mathcal{I})^n\right)^*$, using variables from Σ and identifiers $\mathcal{I}$. An abstract transaction o is instantiated using a *valuation* $\nu : \Sigma \to \mathcal{I}$, denoted by $\nu(o)$, which results in a transaction by replacing all variables by their valuation. The set of all abstract transactions for data model D is denoted by $\mathcal{T}(D)$. ⌟

Starting with a valid population, a transaction should not invalidate the population. Hence, we only allow transitions to fire if both the transition is enabled and its corresponding transaction is valid in the current population. This forms the basis of an ISM, whereas ISML consists of three languages for specifying information models, PNIDs, and specifications which define abstract transactions of the transitions of PNIDs.

Definition 3.8 (Information System Model, Semantics)
An *Information System Model* (ISM) is a tuple $IS = (D, N, S)$, where $D = (E, R, \rho, \Psi)$ is an information model, $N = (P, T, F, \alpha, \beta)$ is a PNID, and $S : T \to \mathcal{T}(D)$ is a specification. A *state* of an information system is a pair (π, m), with population $\pi \in \Lambda(D)$ and marking $m \in \mathcal{M}(N)$. Given markings $m, m' \in \mathcal{M}(N)$ and valid populations $\pi, \pi' \in \Lambda(D)$, transition $t \in T$ with valuation ν is enabled in (π, m) iff $(D : \pi \xrightarrow{\nu(S(t))} \pi')$ and $(N : m \xrightarrow{(t,\nu)} m')$. Its firing results in the new state (π', m'), and is denoted by $(IS : (\pi, m) \xrightarrow{(t,\nu)} (\pi', m'))$. A state (π_n, m_n) is said to be *reachable* from (π_0, m_0) if intermediate states (π_i, m_i) and transitions t_i with valuations ν_i exist such that $(IS : (\pi_i, m_i) \xrightarrow{(t_i,\nu_i)} (\pi_{i+1}, m_{i+1}))$ for all $0 \leq i < n$. ⌟



(a) Model N_1 (b) Model N_2 (c) Model N_3

Fig. 6. Three process models of ISMs that capture the same behavior (subject to the information model); ISMs that rely on models N_1 and N_3 are information- and process-driven, respectively.

A spectrum of information system models. Domain constraints can be expressed in the information model or in the process model, or in both. As an example, consider the models in Fig. 6. Suppose we have information model D with relation types defined by $\rho(P) = \langle \mathtt{P} \rangle$ and $\rho(Q) = \langle \mathtt{P}, \mathtt{P} \rangle$. Let the information model be constrained by $\forall_{x,y \in \mathcal{I}} : [(x,y) \in Q \implies ((x) \in P \wedge (y) \in P)]$. Let the specification map all the transitions in Fig. 6 labeled a to the transaction $\langle \mathrm{delete}(P,(p)), \mathrm{insert}(P,(q)) \rangle$, and all the transitions labeled b to the transaction $\langle \mathrm{insert}(Q,(p,p)) \rangle$. The three process models result in the following ISMs: $IS_1 = (D, N_1, S)$, $IS_2 = (D, N_2, S)$, and $IS_3 = (D, N_3, S)$. Suppose, we start from the empty population. In all three ISMs, transition b is only enabled after a transition with label a has fired at least once; it requires a fact $(x) \in P$, which initially does not hold. Similarly, suppose we have a population with fact $(x) \in P$. Firing a transition with label b results in the population with facts $(x) \in P$ and $(x,x) \in Q$. Removing fact $(x) \in P$ is not allowed anymore, as this will violate the constraint. Hence, transition a is never enabled once transition b fired. Consequently, given the initial empty population, all three ISMs model exactly the same behavior.

The above example shows that different ISMs can describe exactly the same behavior. Model N_1 does not impose any order on the process. Hence, always if a transaction in N_1 is valid, the corresponding transition is enabled. We call this behavior *information-driven*. On the other hand, in model N_3 it becomes directly apparent that there is a constraint on the order of firing transitions a and b: always if the transition is enabled in the net, the corresponding transaction is valid. We say such transitions are *process-driven*. Model N_2 is a combination of the two: the top transition a and transition b are both process- and information-driven, whereas the other transition a is only information-driven. These examples show the existence of a spectrum of ISMs:

Definition 3.9 (Information- and process-driven ISMs)
Let $IS = (D, N, S)$ be an ISM with $N = (P, T, F, \alpha, \beta)$. Transition $t \in T$ is called:
- *information-driven* if $(D : \pi \xrightarrow{\nu(S(t))} \pi')$ implies $(IS : (\pi, m) \xrightarrow{(t,\nu)} (\pi', m'))$,
- *process-driven* if $(N : m \xrightarrow{(t,\nu)} m')$ implies $(IS : (\pi, m) \xrightarrow{(t,\nu)} (\pi', m'))$,

for any two markings $m, m' \in \mathcal{M}(N)$, valuation ν, and populations $\pi, \pi' \in \Lambda(D)$. If all transitions in the PNID are information-driven (process-driven), the ISM is called *information-driven* (*process-driven*). ⌟

Most transitions are neither information- nor process-driven. Instead, for each transition, the modeler balances between information and process. As an example, consider transition *register exam* from Fig. 4. Suppose PTI prescribes that registering for an

exam is only allowed if all courses the student listed in the study plan are passed. In the current model, if the student has its study plan accepted, but did not yet follow a single course, transition *register exam* is enabled. Although the above constraint could be modeled in the process model, it adds unnecessary complexity, whereas the constraint is relatively simple to be expressed in first-order logic (see Fig. 7). Therefore it can be added to the information model, rather than to encode it in the process model.

Being aware of how constraints manifest in the different models and their consequences is essential when designing information systems. This is the main idea behind requirement R3 and our instantiation of this requirement with ISMs, i.e., that designers of information systems are aware of which constraints are imposed and how they interplay within the system, as these are a possible cause of mistakes, as experienced by many students [33]. ISML allows modelers to decide how to specify constraints, and to verify the consequences of that decision.

4 Automated Verification

Automated verification assists designers in checking whether their system satisfies expected properties. An important class of properties are reachability related [12]: Given some current state of an IS, it should always be possible to reach some other state of the system. For example, a student that starts studying a track, should always be able to finish the track. This results in the following definition of the reachability problem:

Definition 4.1 (Reachability problem)
Given an initial state (π_0, m_0) of an ISM (D, N, S), the reachability problem consists in deciding whether a state (π, m) is reachable from (π_0, m_0). ⌟

Combining information and process models is almost a guarantee to violate requirement R4 [6]. In general, the reachability problem is undecidable for Petri nets with identifiers, as there is no structure on the countably infinite set of identifiers; this observation is similar to the one for ν-PNs [21, 28]. In ISMs, identifiers represent elements in the information model. Under the assumption that no information model becomes infinite, which is a reasonable assumption [6], there is a bound on the number of elements the identifiers represent. Further assuming that each identifier is generated consecutively, provides an ordering on the identifiers. These two assumptions form the basis of the class of *counter-valuated PNIDs* [17]. In this class, identifiers are mapped on the natural numbers, and an implicit *counter place* is used to generate the next, fresh, identifier for every fresh element in the information model. As the last generated identifier for a given net is always known, the net can be translated into a classical Petri net [34], for which the reachability problem is decidable [12].

Based on the same assumptions, one can conclude that the set of populations for an information model induced from a finite set of elements is finite. In addition, the set of transactions possible on these populations is finite. Consequently, the process of moving between populations can be represented by a deterministic finite automaton. Hence, given an upper bound k on the identifiers, the semantics of the information system model becomes the synchronous product of a classical Petri net, the one constructed from the corresponding counter-valuated PNID, and a deterministic finite automaton, the one obtained from the information model, for which reachability is again decidable [23]. All these observations lead to the main reachability result for ISMs.

```
tff(register_for_exam}, conjecture,
  ! [p: Person, t: Track]:
    ( registeredExam(p,t) =>
    ? [s: StudyPlan]: ( isOf(s,p,t)
      & ! [c: Course]: ( contains(s,c) =>
        ? [l: Lecturer] :
        passes(p,t,c,l) ) )
  ) ).
```

Fig. 7. Constraint in TPTP-format.

```
process Student {
  ...
  transition register(p: Person,
                      t: Track) {
    register p;
    insert (p) into Person;
    insert (p,t) into registers;
  } ... }
```

Fig. 8. Excerpt of the specification.

Theorem 4.2 (Decidability of the reachability problem)
Given an ISM (D, N, S), where N is a counter-valuated PNID, it is decidable whether some state (π, m) is reachable from the initial state (π_0, m_0) of the ISM. ⌟

Proof. (sketch) Let $\bar{N}$ be the classical Petri net derived from the k-bounded net N, where $k \in \mathbb{N}^0$ is the last generated identifier in N, cf. [34]. Let Q be the automaton induced by the up-to-k-bounded populations of D. Then Q is finite and deterministic. As each transition in $\bar{N}$ maps to a transaction in Q via specification S, one can construct the synchronous product of $\bar{N}$ and Q that describes the semantics of the ISM. Hence, the reachability of the k-bounded ISM translates to the reachability of the synchronous product, which is decidable [23]. □

Based on the result in [22], we conclude that the proposed decision procedure requires at least $2^{c \times (2^{k^u \times v} + p \times k^w)}$ space for some constant $c > 0$, where k, as in the above proof sketch, is the identifier in the counter place of N, u is the number of relations in D, v is the length of the shortest relation in D, w is the minimal sum of all incoming and outgoing arcs of some place of N, and p is the number of places in N. Details on obtaining this result and for the rigorous proof of Theorem 4.2 can be found in [34].

5 Tool Support and Initial Evaluation

To show the applicability of ISML, we have implemented our approach in a prototype called ISM Suite.[3] In this prototype, we build upon CPN tools [35] for simulating the PNID, and an own implementation of a theorem prover on finite sets. Constraints of the information model are specified in TPTP syntax [32]; an example constraint is shown in Fig. 7. The specification uses a special format to define transactions of transitions. An excerpt of the specification of the running example is in Fig. 8. The specification language has three constructs that can be used to define transactions: *register*, to register an element in the population, and *insert* and *remove* to add and remove facts, respectively. If an unregistered element is used in a fact, the resulting population is invalid.

All enabled transitions that result in valid populations are listed in the user interface, from which the user can select a transition to fire. For each transition that yields an invalid population, the violated constraints can be requested, to support the designer in better understanding the reasons of the violation.

In [33], we reported on an initial evaluation of the modeling component of ISM Suite with a cohort of Information Systems students in a real teaching and learning

[3] The source code is available from https://github.com/information-systems/ismsuite.

environment; the students used our tool to solve an information system modeling task. The initial results are promising, as evidenced by the collected qualitative comments from the students, refer to [33]. This and subsequent collected feedback will be used to inform evolution of our tool.

6 Conclusions and Future Work

The paper at hand proposes an approach for modeling an information system as an integration of an information model and a process model via a specification on how processes manipulate information. The proposal constitutes a unique instantiation of standard requirements for capturing concepts and constraints of an information system. Using the proposed formalism, one can express an infinite state system that supports CRUD operations over arbitrary finite populations of information facts governed by the constraints expressed in first-order logic with equality. At the same time, the proposed formalism enjoys the decidability of the reachability problem, which sets a solid foundation for verification of formal properties of the described systems.

Future work will strengthen the results reported in this paper to allow the adoption of the language by practitioners. The concrete next steps include studies of other verification problems and data flow anomalies [31], studies of the interplay between information and process concepts and constraints, improvement of the tool support, development of methodologies for designing information systems using our formalism, and empirical studies aimed at improving the usability of the approach. Finally, the high lower bound on the space requirement reported at the end of Section 4 justifies that one can use ISML to capture a wide range of systems. It is interesting to study how often do the extremely complex cases manifest in the problems encountered in the real world.

Acknowledgment. Artem Polyvyanyy was partly supported by the Australian Research Council Discovery Project DP180102839.

References

1. W. M. P. van der Aalst and C. Stahl. *Modeling Business Processes—A Petri Net-Oriented Approach*. Cooperative Information Systems series. MIT Press, 2011.
2. S. Abiteboul, L. Segoufin, and V. Vianu. Modeling and verifying active XML artifacts. *IEEE Data Eng. Bull.*, 32(3):10–15, 2009.
3. S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.
4. F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of agent-based artifact systems. *Journal of Artificial Intelligence Research*, 51:333–376, 2014.
5. K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, volume 4714 of *LNCS*. Springer, 2007.
6. D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data-aware process analysis: a database theory perspective. In *PODS*, pages 1–12. ACM, 2013.
7. D. Calvanese, M. Montali, M. Estañol, and E. Teniente. Verifiable UML artifact-centric business process models. In *CIKM*. ACM Press, 2014.
8. G. De Giacomo, X. Oriol, M. Estañol, and E. Teniente. Linking data and BPMN processes to achieve executable models. In *CAiSE*, volume 10253 of *LNCS*. Springer, 2017.
9. 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. AAAI Press, 2017.

10. A. Deutsch, R. Hull, Y. Li, and V. Vianu. Automatic verification of database-centric systems. *SIGLOG News*, 5(2):37–56, 2018.
11. A. Deutsch, Y. Li, and V. Vianu. Verification of hierarchical artifact systems. In *PODS*, pages 179–194. ACM Press, 2016.
12. J.Esparza and M.Nielsen. Decidability issues for Petri nets–a survey. *EATCS Bull.*, 52, 1994.
13. M. Estañol, M.-R. Sancho, and E. Teniente. Verification and validation of UML artifact-centric business process models. In *CAiSE*, volume 9097 of *LNCS*. Springer, 2015.
14. C. E. Gerede and Jianwen Su. Specification and verification of artifact behaviors in business process models. In *ICSOC*, volume 4749 of *LNCS*, pages 181–192. Springer, 2007.
15. T. A. Halpin and Anthony C. Bloesch. Data modeling in UML and ORM: A comparison. *J. Database Manag.*, 10(4):4–13, 1999.
16. B. Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *PODS*. ACM Press, 2013.
17. K.M. van Hee, N. Sidorova, M. Voorhoeve, and J.M.E.M. van der Werf. Generation of Database Transactions with Petri nets. *Fundamenta Informatica*, 93(1–3):171 – 184, 2009.
18. R. Hull, J. Su, and R. Vaculín. Data management perspectives on business process management: tutorial overview. In *SIGMOD*, pages 943–948. ACM, 2013.
19. K. Jensen. *Coloured Petri Nets—Basic Concepts, Analysis Methods and Practical Use Volume 1*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1996.
20. S. Lasota. Decidability border for Petri nets with data: WQO dichotomy conjecture. In *Petri Nets*, volume 9698 of *LNCS*, pages 20–36. Springer, 2016.
21. R. Lazic, T. C. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. *Fundam. Inform.*, 88(3):251–274, 2008.
22. R.J. Lipton. *The reachability problem requires exponential space*. Research report (Department of Computer Science). Yale University, 1976.
23. E.W. Mayr. Persistence of vector replacement systems is decidable. *Acta Inf.*, 15(3), 1981.
24. M. Montali and A. Rivkin. DB-Nets: On the marriage of colored Petri nets and relational databases. In *Petri Nets*, volume 10470 of *LNCS*, pages 91–118. Springer, 2017.
25. D. Müller, M. Reichert, and J. Herbst. Data-driven modeling and coordination of large process structures. In *CoopIS*, volume 4803 of *LNCS*. Springer, 2007.
26. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
27. H.A. Reijers, I.T.P. Vanderfeesten, M.G.A. Plomp, P. Van Gorp, D. Fahland, W.L.M. Van der Crommert, and H.D.D. Garcia. Evaluating data-centric process approaches: Does the human factor factor in? *SoSyM*, 16(3), 2017.
28. F. Rosa-Velardo and D. de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theoretical Computer Science*, 412:4439–4451, 2011.
29. V. Sassone, M. Nielsen, and G. Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1-2):297–348, dec 1996.
30. M. Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 66(1):40–65, 2003.
31. S.X. Sun, J.L. Zhao, J.F. Nunamaker Jr., and O.R.L. Sheng. Formulating the data-flow perspective for business process management. *Inf. Syst. Res.*, 17(4), 2006.
32. G. Sutcliffe, S. Schulz, K. Claessen, and A. van Gelder. Using the TPTP language for writing derivations and finite interpretations. In *Aut. Reason.*, volume 4130 of *LNCS*. Springer, 2013.
33. J.M.E.M. van der Werf and A. Polyvyanyy. An assignment on information system modeling. In *BPM Ed. Symp.*, volume 342 of *LNBIP*. Springer, 2018.
34. J.M.E.M. van der Werf and A. Polyvyanyy. On the decidability of reachability problems for models of information systems. Technical Report UU-CS-2018-005, Utrecht University, 2018.
35. M. Westergaard and L.M. Kristensen. The access/CPN framework: A tool for interacting with the CPN-tools simulator. In *Petri Nets*, volume 5606 of *LNCS*. Springer, 2009.