The Information Systems Modeling Suite

Modeling the Interplay Between Information and Processes

Jan Martijn E. M. van der Werf[1] and Artem Polyvyanyy[2]

[1] Department of Information and Computing Science
Utrecht University, The Netherlands
j.m.e.m.vanderwerf@uu.nl
[2] School of Computing and Information Systems
The University of Melbourne, Parkville, VIC, 3010, Australia
artem.polyvyanyy@unimelb.edu.au

Abstract. According to our recent proposal, an information system is a combination of a process model captured as a Petri Net with Identifiers, an information model specified in the first-order logic over finite sets with equality, and a specification of how the transitions in the net manipulate information facts. The Information Systems Modeling (ISM) Suite is an integrated environment for developing, simulating, and analyzing models of information systems, released under an open-source license. This paper presents the basic features of the ISM Suite.

Keywords: Information systems, modeling, simulating, tools

1 Introduction

An *information system* (IS) is an integrated system of components that aim to collect, store, organize, manipulate, process, and disseminate data, information, and knowledge, often in the form of digital products. Finding the right balance between static and dynamic aspects is essential when designing an IS. As shown in [8], existing modeling languages often focus only on one of the two aspects. Therefore, we introduced in [8] the Information Systems Modeling Language (ISML), which focuses on expressing both these aspects and their interplay. An ISML model captures information aspects of using first-order logic with finite sets and equality and describes dynamic aspects that govern the information using Petri nets with Identifiers (PNIDs). Due to this symbiosis, ISML models feature:

1. CRUD operations over information facts and arbitrary information constraints;
2. Process dependencies that extend to infinite-state processes; and
3. Formal foundation which enables automated verification, e.g., one can decide if the system can evolve from its initial state into some other state of interest.

This approach to modeling and analyzing information systems promises advantages in educating future Information Systems professionals. Some benefits reported by students include the immediate experience of the consequences of design decisions and traceability between abstract and implemented concepts [11].

This paper focuses on presenting the Information Systems Modeling Suite (ISM Suite), an integrated collection of programs and tools for designing, executing, testing, simulating, and analyzing models of information systems captured in ISML. The next section describes a motivating scenario of a small yet comprehensive information system. Then, Section 3 exemplifies ISML by giving an in-depth presentation of an ISML model that captures the motivating scenario. Section 4 presents the ISM Suite and its features. The paper closes with conclusions.

2 Motivating Example

Consider the following running example, which we employ throughout the paper. A small organization uses a process-aware information system to support the management of its purchase orders. The organization works with a bidding system for its suppliers. If the organization wants to order a Product, it requests a bid from each of its Suppliers. At least two Suppliers need to respond with their bids, before the organization can select the best bid, and order the Product at the chosen Supplier.

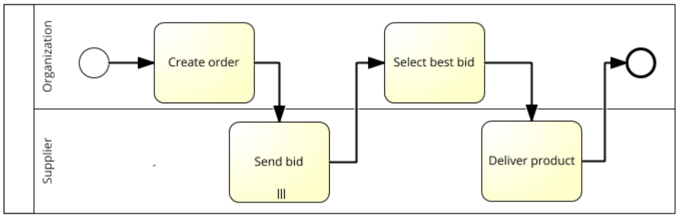


Fig. 1. BPMN model of the Purchase Order system.

The BPMN model that describes the dynamics of the IS is depicted in Fig. 1. First, an employee at the organization creates an Order. Next, several Suppliers *bid* for the Order, implemented via a "multi-instance activity" [2]. Finally, the best Supplier is selected and *receives* the Order, after which the Supplier *delivers* the ordered Product.

The information model of the IS is shown as an Object-Role Model (ORM) [3] in Fig. 2. According to this model, Suppliers *supply* Products. An Order always *contains* exactly one Product (c_1, c_4). Supplier can *bid* on an Order. From the *bids*, a Supplier is selected, and *receives* the Order (c_5). At most one Supplier *receives* the Order (c_2). Eventually, the selected Supplier can *deliver* the Order (c_6). Note that at most one Supplier can *deliver* the Order (c_3).

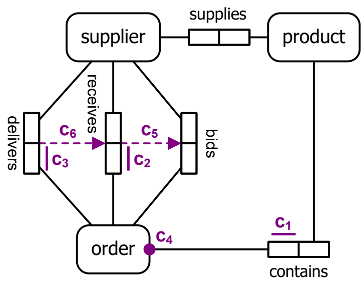


Fig. 2. Information model of the Purchase Order system.

The organization faces a problem with the IS: *no progress can be made for many orders*. Both the process model and the information model do not expose any error: the process model is sound and the constraints of the information model do not contradict. However, it is the interplay of these models that causes the problem: if at most one Supplier *supplies* a requested Product, the process cannot progress. In the remainder of this paper, we show how our tool, the Information Systems Modeling Suite (ISM Suite), can assist the modeler to analyze the interplay between the process and data.

3 Information Systems Modeling Language

Modeling a process-aware information system requires at least three aspects (cf. [1,7,8]): an *information model* for the structure of the information, a *process model* describing the possible activities and their order, and a *specification* that defines how activities manipulate the information model. To model such systems, we proposed the Information Systems Modeling Language (ISML) in [8]. In the remainder of this section, we exemplify ISML using the motivating scenario from Section 2.

3.1 Information Modeling

An *information model* consists of a set of entity types, relations – characterized by finite sequences of entity types – and a set of constraints, specified in first-order logic. Given a sequence or tuple σ, by $\sigma(i)$, $i \in \mathbb{N}$, we denote the element at position i in σ. Let $\mathcal{I}$ and Λ be a universe of *identifiers* and a universe of *labels*, respectively.

Definition 1 (Information model)
An *information model* is a 5-tuple (E, R, ρ, η, 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;
- $\eta : E \to R$ is an *entity relation definition function* that maps every entity type onto a relation type such that η is injective and for every $e \in E$ it holds that $\rho(\eta(e)) = \langle e \rangle$, $\eta(e)$ is called the *entity relation* of e;
- Ψ is a collection of statements in first-order logic that for every $r \in R$, $n = |\rho(r)|$, (i) can use predicate r over the domain $\mathcal{I}^n$, called the *relation predicate* of r, and (ii) contains the statement:

$$\forall\, i \in \mathcal{I}^n : \left(r(i(1), \ldots, i(n)) \Rightarrow \left(\bigwedge_{k=1}^{n} \eta(\rho(r)(k))\,(i(k)) \right) \right),$$

called the *relation predicate statement* of r. ⌟

Definition 1 *refines* the corresponding definition presented in [8]. It introduces function η that explicitly defines entity relations. In addition, it requires that the predicates induced by the relation types, i.e., relation predicates, are named after the corresponding relation types, i.e., for every relation $r \in R$ statements in Ψ can use predicate r of the corresponding arity. Finally, for every relation type, it introduces one relation predicate statement to the theory that establishes dependencies between the truth values of the relation predicates. Note that a relation predicate statement of an entity relation is a tautology. The information model of the Purchase Order system is shown in Example 1.

Example 1 (Information model of the Purchase Order system)
The information model of the Purchase Order system as depicted in Fig. 2 is captured as the 5-tuple (E, R, ρ, η, Ψ), where:
- $E = \{E_s, E_p, E_o\}$;
- $R = \{\mathit{supplier}, \mathit{product}, \mathit{order}, \mathit{supplies}, \mathit{receives}, \mathit{bids}, \mathit{delivers}, \mathit{contains}\}$;
- $\rho = \{(\mathit{supplier}, \langle E_s\rangle), (\mathit{product}, \langle E_p\rangle), (\mathit{order}, \langle E_o\rangle), (\mathit{supplies}, \langle E_s, E_p\rangle), (\mathit{receives}, \langle E_s, E_o\rangle), (\mathit{bids}, \langle E_s, E_o\rangle), (\mathit{delivers}, \langle E_s, E_o\rangle), (\mathit{contains}, \langle E_o, E_p\rangle)\}$;
- $\eta = \{(E_s, \mathit{supplier}), (E_p, \mathit{product}), (E_o, \mathit{order})\}$; and
- $\{\psi_1, \ldots, \psi_{11}\} \subset \Psi$, such that:
  - $\psi_1 \Leftrightarrow \forall i \in \mathcal{I} : (\mathit{supplier}(i) \Rightarrow (\neg \mathit{product}(i) \wedge \neg \mathit{order}(i)))$;
  - $\psi_2 \Leftrightarrow \forall i \in \mathcal{I} : (\mathit{product}(i) \Rightarrow (\neg \mathit{supplier}(i) \wedge \neg \mathit{order}(i)))$;
  - $\psi_3 \Leftrightarrow \forall i \in \mathcal{I} : (\mathit{order}(i) \Rightarrow (\neg \mathit{supplier}(i) \wedge \neg \mathit{product}(i)))$;
  - $\psi_4 \Leftrightarrow \forall o \in E_o \forall p_1 \in E_p \forall p_2 \in E_p : ((\mathit{contains}(o, p_1) \wedge \mathit{contains}(o, p_2)) \Rightarrow (p_1 = p_2))$;
  - $\psi_5 \Leftrightarrow \forall s_1 \in E_s \forall s_2 \in E_s \forall o \in E_o : ((\mathit{receives}(s_1, o) \wedge \mathit{receives}(s_2, o)) \Rightarrow (s_1 = s_2))$;
  - $\psi_6 \Leftrightarrow \forall s_1 \in E_s \forall s_2 \in E_s \forall o \in E_o : ((\mathit{delivers}(s_1, o) \wedge \mathit{delivers}(s_2, o)) \Rightarrow (s_1 = s_2))$;
  - $\psi_7 \Leftrightarrow \forall o \in E_o \exists p \in E_p : \mathit{contains}(o, p)$;
  - $\psi_8 \Leftrightarrow \forall s \in E_s \forall o \in E_o : (\mathit{receives}(s, o) \Rightarrow \mathit{bids}(s, o))$;
  - $\psi_9 \Leftrightarrow \forall s \in E_s \forall o \in E_o : (\mathit{delivers}(s, o) \Rightarrow \mathit{receives}(s, o))$;
  - $\psi_{10} \Leftrightarrow \forall s \in E_s \forall o \in E_o : (\exists p \in E_p : (\mathit{bids}(s, o) \Rightarrow (\mathit{supplies}(s, p) \wedge \mathit{contains}(o, p))))$;
  - $\psi_{11} \Leftrightarrow \forall s_1 \in E_s \forall o \in E_o : (\mathit{bids}(s_1, o) \Rightarrow (\exists s_2 \in E_s : (s_1 \neq s_2 \wedge \mathit{bids}(s_2, o))))$. ⌟

To visualize an information model, we use the Object-Role Modeling (ORM) notation. For each Entity Type $e \in E$, we draw an ORM Entity Type, for each relation types that are not an entity relation, we draw an ORM Fact Type with r as its caption. Some constraints are supported in ORM notation. For example, uniqueness constraints (e.g. ψ_4, visualized as c_1 in Fig. 2), mandatory constraints (e.g. ψ_7 visualized as c_4), and subset constraints (e.g. ψ_8 visualized as c_5).

An information model can be populated with facts. If a population satisfies all constraints, it is called valid.

Definition 2 ((Valid) Population, Fact)
A *population* of information model (E, R, ρ, η, 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 $\pi(r) \in \mathcal{P}\Big(\prod_{i=1}^{|\rho(r)|} \rho(r)(i)\Big)$. An element in $\pi(r)$ is called a *fact*. The population is *valid*, denoted by $\pi \vDash \Delta$, only if $\pi \vDash \Psi$, given that for every $r \in R$, $n = |\rho(r)|$, it holds that $\forall (i_1, \ldots, i_n) \in \mathcal{I}^n : (r(i_1, \ldots, i_n) \Leftrightarrow (i_1, \ldots, i_n) \in \pi(r))$; otherwise π is *invalid*, denoted by $\pi \nvDash \Delta$. ⌟

To manipulate populations, we define two operations for inserting and removing a fact from a relation. Operations can be concatenated, resulting in a transaction.

Definition 3 (Transaction)
Let $\mathcal{D} = (E, R, \rho, \eta, \Psi)$ be an information model and let π be a population of $\mathcal{D}$. An *operation* is a tuple $o \in \mathcal{O}(\mathcal{D})$, where $\mathcal{O}(\mathcal{D}) = (R \times \{\oplus, \ominus\} \times \bigcup_{n \in \mathbb{N}} \mathcal{I}^n)$.

- Operation $o = (r, \oplus, v)$ *inserts* fact v of type r into π iff $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \cup \{v\})\}$ is a valid population of $\mathcal{D}$, denoted by $(\mathcal{D} : \pi \xrightarrow{r \oplus v} \pi')$.
- Operation $o = (r, \ominus, v)$ *removes* fact v of type r from π iff $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \setminus \{v\})\}$ is a valid population of $\mathcal{D}$, denoted by $(\mathcal{D} : \pi \xrightarrow{r \ominus v} \pi')$.

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

Example 2 (Transaction in the Purchase Order system)
A transaction in the Purchase Order system to add some Order o_1 for Product p can be expressed as follows: $\langle(\text{order}, \oplus, (o_1)); (\text{contains}, \oplus, (o_1, p))\rangle$. For any valid population π such that $o_1 \notin \pi(\text{order})$, this transaction is valid. ⌟

3.2 Process Modeling with Petri Nets with Identifiers

For modeling the activities and their order we use Petri nets with Identifiers (PNID) [4,8]. PNIDs can be seen as an extension of ν-Nets [9]. In a PNID, tokens carry a vector of identifiers. These vectors have the advantage that a single token can represent multiple entities at the same time. In this way, a token may represent a (composed) fact from a population of an information model. Each place is typed with a vector of identifiers, i.e., all tokens in a place have the same vector length, called the *cardinality*. Arcs are annotated with vectors of variables. Its size is implied by the cardinality of the place it is connected to. Let Σ denote the universe of variables.

Definition 4 (Petri net with Identifiers)
A *Petri net with Identifiers* (PNID) N is a 5-tuple (P, T, F, α, β), where:
- P and T are two disjoint sets of *places* and *transitions*, resp., i.e., $P \cap T = \emptyset$;
- $F : ((P \times T) \cup (T \times P)) \to \mathbb{N}^0$ is the *flow function*;
- $\alpha\colon P \to \mathbb{N}^0$ defines the *cardinality* of a place, i.e., the length of the vector of identifiers 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., $\beta \in \prod_{\{f \mid F(f) > 0\}\}} V_f$, where $V_{(p,t)} = V_{(t,p)} = \Sigma^{\alpha(p)}$ for $p \in P, t \in T$.

Its set of all possible markings is defined as $\mathcal{M}(N) = \prod_{p \in P}(C(p) \to \mathbb{N}^0)$. The pair (N, m) is a *marked PNID* if $m \in \mathcal{M}(N)$. ⌟

To fire a transition, the variables on its arcs need to be *valuated* to match identfiers the tokens carry. A valuation maps each variable to an identifier. New identifiers can be created if a transition contains variables that only occur on outgoing arcs. The valuation guarantees that variables occurring on an outgoing arc each receive a new, fresh identifier. For the full semantics of PNIDs, we refer the reader to [4,8].

Example 3 (PNID of the Purchase Order system)
The PNID of the Purchase Order system is depicted in Fig. 3. The multiple-instance activity is translated in a standard pattern in which two suppliers can be asked simultaneously. The net starts in the marking with three tokens in place r, and three tokens in place s. These tokens resemble products and suppliers, resp., i.e., $m_0 = \{r \mapsto$

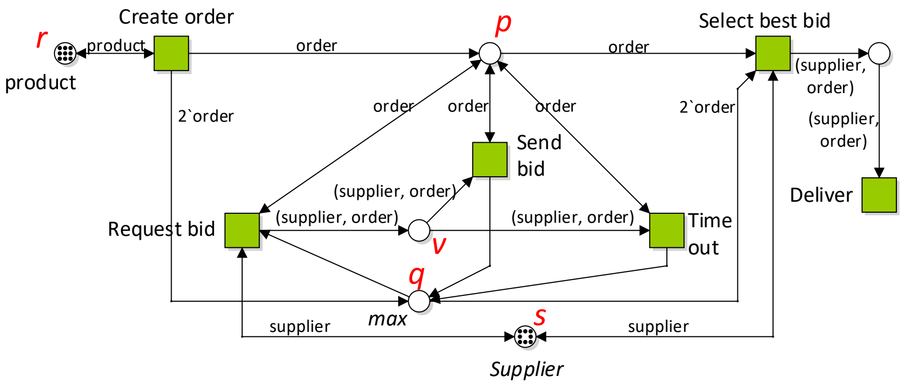


Fig. 3. PNID of the Purchase Order system.

$[p,q,r], s \mapsto [s,t,v]\}$. In this marking, transition Create order is enabled with valuation $\nu_1 = \{\text{order} \mapsto o_1, \text{product} \mapsto p\}$, where o_1 is a fresh identifier. Firing this transition results in marking $m_1 = \{p \mapsto [o_1], q \mapsto [{o_1}^2], r \mapsto [p,q,r], s \mapsto [s,t,v]\}$. ⌟

3.3 Semantics of the Information Systems Modeling Language

Transitions in the process model often resemble events that manipulate the information model. For example, in the Purchase Order system, the transition Create order resembles adding a new order fact to the information model. In ISML, each transition is specified with an *abstract transaction* that describes how the transition manipulates the information model. Similar to transition firing, abstract transactions rely on valuations to be instantiated. The set of all abstract transactions over some information model $\mathcal{D}$ is denoted by $\mathcal{T}(\mathcal{D})$, and extends normal operations by allowing variables in the facts.

Example 4 (Specification of the Purchase Order process)
The Purchase Order system has four transitions with a non-empty abstract transaction. The variables used in the transactions coincide with the variables used on the arcs.

- Transition Create order ensures an order contains exactly one product:
  $S(\text{Create order}) = \langle(\text{order}, \oplus, (\textit{order})); (\text{contains}, \oplus, (\textit{order}, \textit{product}))\rangle$.
- Transition Send bid resembles the activity that a supplier responds to an order:
  $S(\text{Send bid}) = \langle(\text{bids}, \oplus, (\textit{supplier}, \textit{order}))\rangle$
- Transition Select best bid sends the order to the best supplier:
  $S(\text{Select best bid}) = \langle(\text{receives}, \oplus, (\textit{supplier}, \textit{order}))\rangle$
- Transition Deliver models that the order is delivered by the supplier:
  $S(\text{Deliver}) = \langle(\text{deliver}, \oplus, (\textit{supplier}, \textit{order}))\rangle$ ⌟

An information system model has three constituents: an information model $\mathcal{D}$, a PNID N, and a specification S. In an information system, executing a transaction should not result in an invalid population. Therefore, given a state (π, m), a transition can only fire if (i) it is enabled in (N, m), and (ii), its transaction results again in a valid population. For an overview of ISML and its semantics, we refer the reader to [8]. The information system model of the Purchase Order system is given in Example 5.

Example 5 (Information System Model of the Purchase Order system) The information system model for the Purchase Order system uses the information model of Example 1, the PNID depicted in Fig. 3, and the specification in Example 4. Consider the following initial population, with three products and three suppliers:

$\text{product}(p)$	$\text{supplier}(s)$	$\text{supplies}(s,p)$	$\text{supplies}(t,r)$
$\text{product}(q)$	$\text{supplier}(t)$	$\text{supplies}(s,q)$	$\text{supplies}(v,r)$
$\text{product}(r)$	$\text{supplier}(v)$	$\text{supplies}(t,p)$	

Hence, the tokens in place r resemble the products, and the tokens in place s suppliers. Transition Create order is enabled in the PNID with three valuations, one for each product. None of the valuations result in an invalid transaction on the current population. Hence, the transition is enabled in the ISM with all three valuations.
Selecting valuation ν_1 results in transaction: $\langle(\text{order}, \oplus, (o_1)); (\text{contains}, \oplus, (o_1, p))\rangle$. Firing the transition with this valuation results in marking m_1 (see Example 4), and the valid population:

$\text{product}(p)$	$\text{supplier}(s)$	$\text{supplies}(s,p)$	$\text{supplies}(t,r)$	$\text{order}(o_1)$
$\text{product}(q)$	$\text{supplier}(t)$	$\text{supplies}(s,q)$	$\text{supplies}(v,r)$	$\text{contains}(o_1,p)$
$\text{product}(r)$	$\text{supplier}(v)$	$\text{supplies}(t,p)$		

In marking m_1, two transitions are enabled in the PNID: Request bid and Select best bid. Both are enabled with three valuations, one for each supplier. Consider valuation $\nu_2 = \{\text{supplier} \mapsto v, \text{order} \mapsto o_1\}$. For transition Select best bid, this valuation results in the transaction $\langle(\text{receives}, \oplus, (v, o_1))\rangle$. However, this transaction invalidates the constraints ψ_8 and ψ_{11}, as the population contains no bids for order o_1. Hence, this transition is not enabled in the ISM for any of the valuations. Only transition Request bid is enabled. Firing it with valuation ν_2 results in the marking $m_2 = \{p \mapsto [o_1], q \mapsto [o_1], v \mapsto [o_1], r \mapsto [p,q,r], s \mapsto [s,t,v]\}$. Now, transitions Time out and Send bid are enabled in the PNID, with valuation $\nu_3 = \{\text{order} \mapsto o_1\}$. However, the transaction induced by Send bid yields an invalid population, as $\text{supplies}(v,p)$ is not a fact in our population. Hence, only transition Time out is enabled in the current state. ⌟

4 ISM Suite

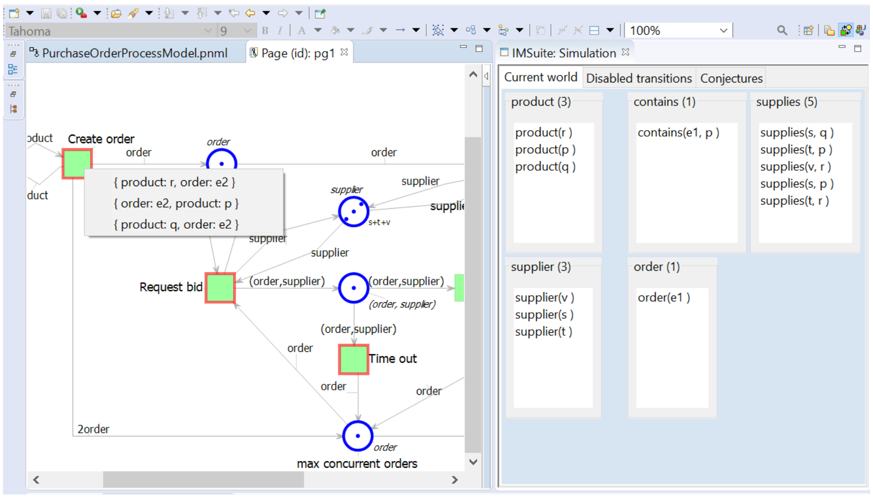
The ISM Suite is implemented as a plug-in for Eclipse and can be installed via a publicly available Update Site.[3] The ISM Suite consists of an editor and simulator for PNIDs, and a simulator for ISMs. It heavily builds upon ePNK [6] and is fully PNML compliant [5]. The ISM Suite has its own perspective in Eclipse. It uses its own prover and simulation libraries, which, together with the source code of the ISM Suite, can be found on Github.[4]

4.1 Editing and Simulating PNIDs

As the ISM Suite process editor is based on ePNK, it opens with a process explorer, which displays the content of the PNML file. Initially, a net is created of the correct type, together with a page. As the editor is PNML compliant, all Petri net content has

[3] See: http://informationsystem.org/ismsuite/

[4] See: https://github.com/information-systems/ISMSuite



{ product: r, order: e2 }
{ order: e2, product: p }
{ product: q, order: e2 }

product (3): product(r), product(p), product(q)

contains (1): contains(e1, p)

supplies (5): supplies(s, q), supplies(t, p), supplies(v, r), supplies(s, p), supplies(t, r)

supplier (3): supplier(v), supplier(s), supplier(t)

order (1): order(e1)

Fig. 4. The ISM Suite. The view on the left depicts the PNID, a popup menu with enabled valuations is shown. The panel on the right shows the current population of the ISM.

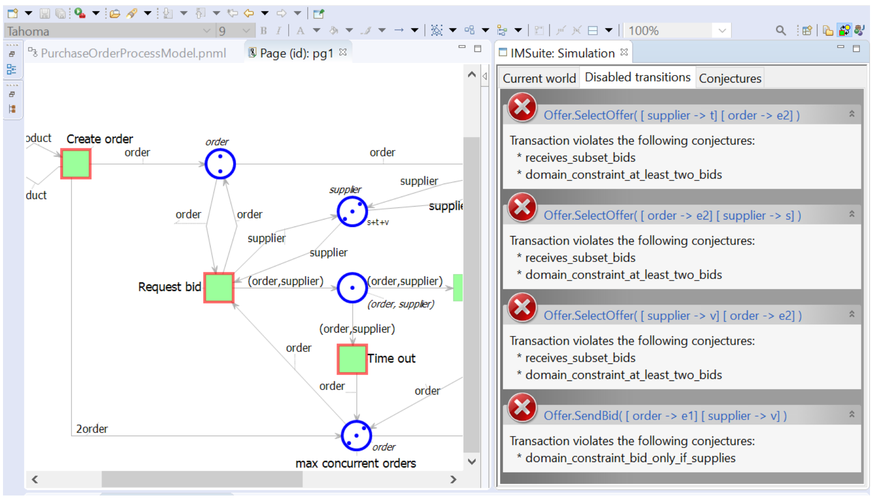
to be created on pages. The editor supports reference places and reference transitions to “divide” the net over several pages. By double-clicking on a page, the graphical editor is started. Places, transitions, and arcs can be inserted on the canvas from the pallet.

By default, all places have the cardinality of 0, i.e., can hold “black” tokens. Instead of working with cardinalities, the ISM Suite is structurally typed, i.e., each constituent has its type, and variables need to be consistently typed. The cardinality is derived from this type. A place can be typed by adding a label to the canvas and connecting it with a “Link label”. The type is specified as a bracketed list of entity types. Each entity type has to start with a lower case. Similarly, the inscription of arcs can be added. Places with a non-empty type are colored yellow in the editor. Tokens can be added by creating another label. Each token is identified by its vector. For example, `3‘()` denotes three black tokens, while `2‘(a,b)+(c,d)` resembles three “colored” tokens: two tokens carry the vector (a, b) and one token carries the vector (c, d).

To simulate the PNID, one can start the simulator via the menu “ISM Suite”. Before the engine is started, the net is validated and type-checked. If the net is invalid, a warning is shown, together with the violations. The simulator is shown in the left panel of Fig. 4. Once the simulator is started, enabled transitions are marked with a thick red outline. On clicking on such transition, a menu is opened with the possible valuations. After selecting a valuation, the transition fires. The tokens residing in a place can be checked by clicking on that place. A menu is opened with a list of all tokens.

4.2 Simulating ISMs

To simulate an information system model, choose the “Start ISM Simulator” option in the menu “ISM Suite”. A dialog is opened that asks for two files: an information model and a specification. The information model needs to be specified in the Typed First-Order Formulae (TFF) format [10]. For example, constraint ψ_{11}, refer to Example 1, at least two bids of different suppliers are required, can be written in TFF as:



Offer.SelectOffer([supplier -> t] [order -> e2])
Transaction violates the following conjectures:
* receives_subset_bids
* domain_constraint_at_least_two_bids

Offer.SelectOffer([order -> e2] [supplier -> s])
Transaction violates the following conjectures:
* receives_subset_bids
* domain_constraint_at_least_two_bids

Offer.SelectOffer([supplier -> v] [order -> e2])
Transaction violates the following conjectures:
* receives_subset_bids
* domain_constraint_at_least_two_bids

Offer.SendBid([order -> e1] [supplier -> v])
Transaction violates the following conjectures:
* domain_constraint_bid_only_if_supplies

Fig. 5. The ISM Suite. The panel on the right explains why certain transitions are not enabled.

```
tff(domain_constraint_at_least_two_bids, conjecture,
  ! [S1: universe, O: universe] :
   ( receives(S1,O) => ( ? [S2: universe] : ( (S1 != S2) & bids(S2,O) ) ) ) ).
```

A transaction is specified by a name and a set of typed variables and consists of a sequence of operations. Different from the information model, the ISM Suite does not "know" which entities are present in each entity type. These need to be added explicitly in the transaction. To this end, there are four operations:

– Operation `register p;` adds element p to its entity type set;
– Operation `deregister p;` removes element p from its entity type set;
– Operation `insert (a,...,z) into relation;` inserts fact $(a, \ldots, z)$ to 'relation'; and
– Operation `remove (a,...,z) from relation;` removes fact $(a, \ldots, z)$ from 'relation'.

Transactions are matched with the transitions of the PNID by their identifiers. To create an initial population, upon starting the simulator, all places that contain tokens are also matched with transactions. For places, the variable names have to match the identity types as specified in the place type. For example, to specify the initial population, as presented in Example 5, a place is added with five tokens, and the place is matched with the following transaction:

```
transaction Offer.provides(supplier: universe, product: universe)    {
  register product; register supplier;
  insert (product) into product; insert (supplier) into supplier;
  insert (supplier, product) into supplies;                           }
```

If the model is fully consistent, the simulator is started, as shown in Fig. 4. On the left, the process model is shown, the panel on the right shows the current population. By firing transitions, the population is updated automatically. As shown in the figure, transition Select best bid is not enabled, even though there are sufficient tokens. To study why the transition is not enabled, the panel on the right has a second tab, "Disabled transitions", as shown in Fig. 5. For each transition and valuation that is not enabled, an explanation is given why the transition is not enabled. As shown in this figure, transition Select best bid, is not enabled because it violates two constraints "receives subset bids" and domain constraint "at least two bids".

5 Conclusion

As the interplay between data and processes can be very subtle, it is not sufficient to only study information and process models in isolation. In this paper, we present the Information System Modeling Suite (ISM Suite). It is a tool that helps to study how processes and data are related in an information system. The tool provides editing and simulation facilities for modeling Petri Nets with Identifiers. Furthermore, it allows simulating information system models. It combines process models with an information model in terms of first-order logic constraints, and a simple specification language to define how transitions manipulate the information model. The tool provides visual aids to assist the modeler by explaining why certain transitions are disabled in the ISM.

We envision the ISM Suite as a tool for learning modeling of information systems. In the future, we want to perform several experiments with students to validate the modeling approach, and how it is experienced, similar to [11]. As the ISM Suite currently only supports visual modeling of processes, we plan to extend it with visual facilities for information modeling, together with simulation and analysis options.

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

References

1. 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.
2. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer, 2018.
3. T. A. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers, 2nd edition, 2008.
4. 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.
5. L. M. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Tréves. A primer on the petri net markup language and iso/iec 15909-2. *Petri Net Newsletter*, 76:9–28, 2009.
6. E. Kindler. The ePNK: A generic PNML tool - Users' and Developers' Guide for Version 1.0.0. Technical Report IMM-Technical Report-2012-14, DTU Informatics, 2012.
7. 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.
8. A. Polyvyanyy, J. M. E.M. van der Werf, S. J. Overbeek, and R. A. C. M. Brouwers. Information systems modeling: Language, verification, and tool support. In *CAiSE 2019*, volume 11483 of *LNCS*, pages 194–212. Springer, 2019.
9. F. Rosa-Velardo and D. de Frutos-Escrig. Decidability and complexity of Petri nets with unordered data. *Theoretical Computer Science*, 412:4439–4451, 2011.
10. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.
11. 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.