On a Synergy Between Data and Processes

Jan Martijn E.M. van der Werf

Artem Polyvyanyy

Sietse Overbeek

Rick Brouwers

Technical Report UU-CS-2018-004
May 2018

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

On a Synergy Between Data and Processes

Jan Martijn E.M. van der Werf [*] Artem Polyvyanyy [†]

Sietse Overbeek [‡] Rick Brouwers [§]

May 29, 2018

Abstract

The structure of data and processes to manage information are both important aspects of information systems. Nevertheless, most existing modeling languages focus either on one or the other. Languages that focus on data often neglect that data is manipulated by processes, while languages tailored for processes ignore the structure of the data.

In this paper, we present an approach to model and analyze information systems that combines data models with process models by means of an automated theorem prover. In the proposed approach, data models are used to express the structure and constraints of data, while Petri nets with identifiers define the processes together with a specification on how transitions manipulate the data. Through the use of an automated theorem prover, a transition is checked whether its manipulation is valid given the constraints imposed by the data model. Our approach allows specifying an information system over a spectrum of balances between data and process constraints. This spectrum is exemplified with an educational institute as a running example. The approach is supported with a proof-of-concept implementation using CPN-tools and the automated theorem prover E.

1 Introduction

Finding the right balance between data and process constraints 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 data modeling notations introduce concepts to model and verify domain constraints, but neglect that information is populated through processes. Similarly, process-oriented languages, like BPMN [6], contain dedicated constructs to represent data, e.g., documents and messages, and data flows connecting these to the activities. But, these constructs are often of little help when specifying some non-trivial constraints imposed by the domain.

[*]Utrecht University, The Netherlands j.m.e.m.vanderwerf@uu.nl

[†]The University of Melbourne, Australia, artem.polyvyanyy@unimelb.edu.au

[‡]Utrecht University, The Netherlands s.j.overbeek@uu.nl

[§]Utrecht University, The Netherlands r.a.c.m.brouwers@students.uu.nl

Similar symptoms are observed when it comes to model analysis, as hardly any analysis technique is grounded in both data and processes. A classical property in process analysis is *soundness* [1]. For a sound process model it is guaranteed, for example, that it is always possible to complete every started process instance, and there are no pending activities once a process instance completes. A classical property in data analysis is *consistency*, which states that no manipulation on a data population should violate the constraints set by the model. These two properties together demonstrate the need for a synergy between data and processes, as processes of an information system should be sound and all the data manipulations caused by the processes should be consistent.

When it comes to modeling an information system, some constraints are better suited for capturing in a process model, whereas the others fit better in the data model. For example, based on a given balance, different technological choices can be made. If most constraints of the system are data-driven, an active database [16] would be a good choice for the underlying technology, whereas if most constraints are process-driven, one can opt for a Business Process Management System (BPMS) as an underlying technology. Making a well-informed decision, requires the balance between data and process constraints to be explicit.

Several approaches aim to exploit the synergy between processes and data. For example, Colored Petri nets (CPN) can be used to encode cases [8], but adding data in general breaks analysis. Other approaches introduce a formal notation for identifiers that can be utilized to identify business cases, e.g., ν-PN [18, 19]. An extension on this class, are DB-nets [15], which utilizing CPN to encode the link between processes and the underlying data layer.

In practice, most BPMSs, like Bizagi BPM, Bonita BPM and Process Maker [1] follow a different strategy. Instead of relying on a single formalism, processes are modeled in BPMN, and, independently, a data model is developed in a data modeling environment. To connect the processes to the data, processes are attributed with process variables, and activities manipulate these variables and the populated data model using forms, basically following the approach as presented in [11]. In this paper, we follow a similar line of thought. We consider an information system to consist of three artifacts: 1) a *data model* for capturing the domain and its constraints; 2) a *process model* based on Petri nets with identifiers [12]; and 3) a *specification* defining the data manipulations by the activities.

Concretely, this paper makes these three contributions:

1. Proposes an approach for balancing data and process constraints in a model of an information system, and discusses consequences of balancing these constraints;
2. Demonstrates that the proposed approach for modeling information systems remains decidable and, thus, analysis is still possible; and
3. Presents a proof-of-concept implementation of the proposed approach based on CPN-tools and automated theorem prover E, which shows the feasibility of the proposed approach.

In the paper, a running example will be used to explicate the proposed concepts.

[1]see: `http://www.bizagi.com`, `http://bonitasoft.com/`, and `http://www.processmaker.com/`

Running Example. The educational institute "Private Teaching Institute" (PTI) offers different education tracks, such as information sciences. PTI consists of a small team per track, called the track management, 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, they become a student of that track. Students accepted for the track have to create a study plan, consisting of the courses they want to follow. This plan has to be approved by the track management. Students enrol 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 a course. In case a student fails, she is allowed to retake, until she passes the course. 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.

The remainder of this paper is structured as follows. The next section presents a formalism for capturing data models, instantiations of data models into populations, and principles for manipulating populations of data models. Section 3 introduces Petri nets with identifiers as an example of a process-oriented modeling language. An approach for achieving a synergy between process and data models is proposed in Section 4. This section also demonstrates how the proposed approach allows balancing between data and process constrains in specifications of information systems, and discusses its proof-of-concept implementation. The paper closes with conclusions. Finally, the mathematical notations used in this paper can be found in Appendix A.

2 Data Models

Many languages are available for modeling data, in which concepts are represented together with relationships between these concepts and additional constraints on the data model. Noteworthy examples of such languages are class diagrams in the Unified modeling Language (UML) [7], the Entity-Relationship (ER) language [4], and the Object-Role modeling (ORM) language [9]. When viewing these three data modeling languages altogether it can be concluded that different terminology and graphical notations are used for comparable modeling constructs.

Remarkable differences in the way how constraints are expressed in each of these modeling languages include, e.g., the possibility to express constraints in UML class diagrams by means of the technical Object Constraint Language (OCL) [2] to realize machine-readable data model constraints. The Semantics Of Business Vocabulary And Rules (SBVR) language [3] enables to generate natural language expressions of constraints that are part of an ORM model. The underlying idea of this constraint language is to generate a human-readable verbalization of an ORM model. This verbalization can on its turn be interpreted and validated against a specific universe of discourse of which the ORM model is an abstraction.

[2]see: `http://www.omg.org/spec/OCL/2.2/PDF/`
[3]see: `http://www.omg.org/spec/SBVR/1.0/PDF/`

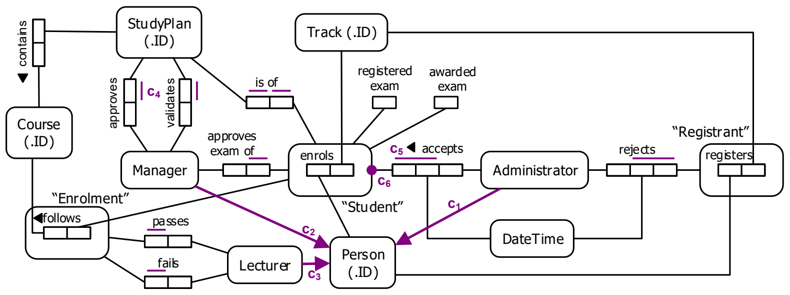

Figure 1: A data model of the running example in ORM notation.

2.1 A Formalization with Set Theory and First-Order Logic

Each of the aforementioned formalisms focuses on different aspects of modeling data. Each formalism comes with its constructs and ways to express constraints. Yet, all these notations are similar in that they are founded in set theory, relational algebra and first-order logic [4, 5]. We do not advocate for the use of specific notations, but rather focus on the underlying principles that drive them. For a given situation, a *data model* describes all allowed populations, i.e., conditions that must be satisfied by a population. A *population* consists of entities (with attributes), and relations between the entities. Based on the relational model of Codd [5], entities and relations are represented using set theory [4]. This forms the basis of our data model definition. Therefore, a data model consists of a set of possible entity types, and labels for entity types and relations, which are characterized by finite sequences of entity types.

Let $\mathcal{I}$ denote the universe, i.e., a countably infinite set, of *identifiers*, and Λ the universe of *labels*, which are disjoint.

Definition 2.1 (Data model)**.** A *data model* is a 4-tuple (T, R, ρ, Ψ), where:

- $T \subseteq \mathcal{P}(\mathcal{I})$ is a finite set of *entity types*,
- $R \subseteq \Lambda$ is a finite set of *relation types*,
- $\rho : R \to T^*$ is a *relation definition function* that maps every relation type onto a finite sequence of entity types for which it holds that for every $t \in T$ there exists $r \in R$, called the *entity relation* of t, such that $\rho(r) = \langle t \rangle$, 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 $\prod_{i=1}^{|\rho(r)|} \rho(r)(i)$. ⌟

A data model of our running example captured using the Object-Role modeling (ORM) notation [9] is shown in Fig. 1. One can use basic types, such as `Integer`, `String` and `Date`, or distinguish domain specific types, such as `Person`, `Track`, and `Course`. In the figure, boxes with rounded corners denote entity types, for example `Manager` and `DateTime`. Rectangles stand for relations (or facts using the ORM terminology). For example, "*contains*" is a binary relation between `StudyPlans` and `Courses`, identified by `StudyPlanIDs` and `CourseIDs`, respectively (thus, the "`.ID`" notation). Note that "*approves exam of*" is a ternary relation, as each "`Student`" is

$\rho(\textit{registers}) = \langle \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{rejects}) = \langle \texttt{Administrator}, \texttt{DateTime}, \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{accepts}) =$

$\langle \texttt{Administrator}, \texttt{DateTime}, \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{enrols}) = \langle \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{isOf}) = \langle \texttt{StudyPlan}, \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{contains}) = \langle \texttt{StudyPlan}, \texttt{Course} \rangle$

$\rho(\textit{approves}) = \langle \texttt{Manager}, \texttt{StudyPlan} \rangle$

$\rho(\textit{follows}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course} \rangle$

$\rho(\textit{fails}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course}, \texttt{Lecturer} \rangle$

$\rho(\textit{passes}) = \langle \texttt{Person}, \texttt{Track}, \texttt{Course}, \texttt{Lecturer} \rangle$

$\rho(\textit{registeredExam}) = \langle \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{validates}) = \langle \texttt{Manager}, \texttt{StudyPlan} \rangle$

$\rho(\textit{approvesExamOf}) = \langle \texttt{Manager}, \texttt{Person}, \texttt{Track} \rangle$

$\rho(\textit{awardedExam}) = \langle \texttt{Person}, \texttt{Track} \rangle$

Figure 2: A relation definition function.

identified by a pair composed of a `Person` and `Track`. For more information on ORM, please refer to [9].

Each entity type and relation type of a data model is identified by a label and a corresponding sequence of entity types. For example, the entity type `Person` is characterized by the 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 '*enrols*' such that $\rho(\textit{enrols}) = \langle \texttt{Person}, \texttt{Track} \rangle$. We shall employ the notation in which labels that stand for entity relations are capitalized and those that stand for the other relations start with lowercase. Fig. 2 gives the relation definition function of the running example (without the entity relations).

A data model may contain various constraints. The constraints are captured as the first-order logic statements that define the formal theory of the data model. Based on the structure, one can distinguish various classes of constraints. Fig. 1 depicts some constraints of the running example in ORM notation. Let (T, R, ρ, Ψ) be a data model. Next, in the context of the running example, we discuss several classes of constraints [4].

An entity of one type can also belong to another type. This can be captured using a *subtype constraint*. Given two entity types $X, Y \in T$, X *is a subtype of* Y if and only if $\forall\, x \in \mathcal{I} : x \in X \Rightarrow x \in Y$. In Fig. 1, arrows c_1, c_2, and c_3 capture subset constraints. For example, c_1 specifies that $\forall\, x \in \mathcal{I} : x \in \textit{Administrator} \Rightarrow x \in \textit{Person}$. Note that the running example permits a person at the same time to be an administrator, a manager, and a lecturer.

Every tuple in a relation is unique, i.e., it occurs at most once in the relation. Sometimes one needs to specify that a combination of a subset of elements in a tuple of a relation is unique. This can be done by means of a *uniqueness constraint*. For example, the constraint $\forall\, x, y, z \in \mathcal{I} : ((x, z) \in r \wedge (y, z) \in r) \Rightarrow x = y$ specifies that there are no two tuples in binary relation r with identical first elements. In Fig. 1, interval c_4 denotes the constraint $\forall\, x, y, z \in \mathcal{I} : ((z, x) \in \textit{approves} \wedge (z, y) \in \textit{approves}) \Rightarrow x = y$. Uniqueness constraints can spawn multiple elements. Interval c_5 specifies constraint $\forall\, x, y, z, u, v \in \mathcal{I} : ((x, z, u, v) \in \textit{accepts} \wedge (y, z, u, v) \in \textit{accepts}) \Rightarrow x = y$, i.e., each combination of the last three elements of a tuple in *accepts* is unique.

It may be required that a relation, i.e., all its tuples, "participate" in another relation. This can be expressed by means of a *mandatory constraint*. For example, mandatory

[4]Material for the running example can be found at `http://www.architecturemining/tools/DPS`.

constraint $\forall\, x, y \in \mathcal{I}\, \exists\, z \in \mathcal{I} : (x, y) \in r \Rightarrow (x, y, z) \in s$ specifies that for every pair (x, y) in relation r it holds that there exists a triple (x, y, z), $z \in \mathcal{I}$, in relation s, i.e., it is mandatory for r to participate in the first two positions in s. Thus, constraint c_6, which is the only mandatory constraint in Fig. 1 denoted by a small filled circle, specifies $\forall\, x, y \in \mathcal{I}\, \exists\, u, v \in \mathcal{I} : (x, y) \in \mathit{enrols} \Rightarrow (u, v, x, y) \in \mathit{accepts}$, i.e., every pair in *enrols* must appear at positions three and four of some 4-tuple in *accepts*.

In general, one can specify data model constraints using arbitrary first-order logic expressions. Thus, constraints, in general, may not have the corresponding graphical notation in standard data modeling notations. For example, one can specify that administrators do not cheat using the expression $\forall\, x, y, z \in \mathcal{I} : (x, y, x, z) \notin \mathit{accepts}$, i.e., an administrator cannot accept herself for a track.

The reader familiar with ORM will notice that in Fig. 1 we use ORM for our purpose of encoding the data model of Def. 2.1 of the running example. This is done to allow traceability between visual notation and the formalism. In classical ORM, the running example can be captured slightly differently, e.g., nested `Student`, `Enrolment`, and `Registrant` types would normally be captured as entity types with the compound reference schemes (also called objectified fact types), while `DateTime` would be modeled as a value type.

2.2 Data Model Populations

A data 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. A population may invalidate the constraints. Thus, we say that a population is *valid* if it satisfies all the constraints of the data model; otherwise the population is *invalid*.

Definition 2.2 (Population)**.**
A *population* of a data model (T, R, ρ, Ψ) is a function $\pi : R \rightarrow \mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}^n)$ such that every element in the population is correctly typed, i.e., for every $r \in R$ it holds that $\pi(r) \in \mathcal{P}(\prod_{i=1}^{|\rho(r)|} \rho(r)(i))$. Given a relation r, an element $v \in \pi(r)$ is called a *fact*. ⌟

If a population satisfies the constraints set by the data model, it is called *valid*, denoted by $\pi \models \Psi$; otherwise it is invalid, denoted by $\pi \not\models \Psi$. Given a data model D, by $\Pi(D)$ and $\Lambda(D)$ we denote the set of all possible populations of D and the set of all possible valid populations of D, respectively.

Consider again the data model of Fig. 1 and its relation definition function in Fig. 2. A valid population is: *Track*(IS), *Course*(PM), *Course*(DM), *Course*(PR), *Person*(1012), *Administrator*(1012), *Person*(520639), *registers*(520639, IS), *enrols*(520639, IS), *accepts*(1012, 15-03-18 19:01, 520639, IS), *StudyPlan*(SP98), *isOf*(SP98, 520639, IS), *contains*(SP98, PM), *contains*(SP98, PR). In this population, there is a track Information Sciences (IS), and three courses: Data modeling (DM), Process modeling (PM), and Programming (PR). The population specifies one administrator (Alice) with personnel number 1012, one student (Robin) of the IS track with student number 520639, who was accepted by Alice, and is currently creating a study plan, containing the courses PM and PR. As this population satisfies all the constraints, it is valid.

2.3 Manipulations with Populations

The population of an information system changes frequently, entities and facts are added to, deleted from, or updated in populations. To this end, we define two operations for manipulating populations: inserting entities into a relation and removing entities from a relation. Note that an update can be implemented as a transaction of first deleting and then inserting certain entities.

Definition 2.3 (Transaction)**.**
Let $D = (T, R, \rho, \Psi)$ be a data model, let $r \in R$ be a relation, let $v \in \prod_{i=1}^{|\rho(r)|} \rho(r)(i)$ be a tuple, called a *fact*, and let $\pi \in \Pi(D)$ be a population.

- ◦ Population $\pi' \in \Pi(D)$ is the result of *inserting* fact v into r in π, denoted by $(D : \pi \xrightarrow{r \oplus v} \pi')$, if and only if $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \cup \{v\})\}$.
- ◦ Population $\pi' \in \Pi(D)$ is the result of *deleting* fact v from r in π, denoted by $(D : \pi \xrightarrow{r \ominus v} \pi')$, if and only if $\pi' = (\pi \setminus \{(r, \pi(r))\}) \cup \{(r, \pi(r) \setminus \{v\})\}$.

A *transaction* is a finite sequence composed of the above two operations such that every subsequent operation is performed in a population that results from the previous operation. It is called *valid* iff the starting and final resulting populations are valid. ⌟

When the context is clear, i.e., the scope of the data model and its current population are known, we write $\text{insert}(r,\ v)$ and $\text{delete}(r,\ v)$ instead of $(D : \pi \xrightarrow{r \oplus v} \pi')$ and $(D : \pi \xrightarrow{r \ominus v} \pi')$, respectively.

As Robin is working on her study plan, refer to the population proposed at the end of Section 2.2, updating the course Programming to Data modeling in her study plan can be implemented using this transaction:

$$\begin{array}{l} \text{delete}(\mathit{contains}, (\text{SP98}, \text{PR})); \\ \text{insert}(\mathit{contains}, (\text{SP98}, \text{DM})); \end{array}$$

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

3 Process Models

Process models mainly focus on the control flow aspects of processes. Classical Petri nets, in the form of workflow models represent processes with a clear start and finish. An important correctness property is soundness [1]: a workflow model is sound if the model is 1) *weakly terminating*, i.e., it is always possible to reach a marking in which only the final place is marked, and 2) *properly completing*, i.e., once the final place receives a token, there are no other tokens pending in the net.

In general, many different cases may be active concurrently. Generalized soundness [1] extends soundness by strengthening the weak termination property: if the net starts with some number of tokens in the initial place, the marking in which all these tokens have been transferred to the final place should be reachable.

The net in Fig. 3 is a generalized sound workflow, if one ignores transitions *start* and *archive*, the yellow-colored places *education track*, *administrator*, *course*, *manager* and *lecturer*, and the arc inscriptions (i.e., the net starting at place i, and finishing

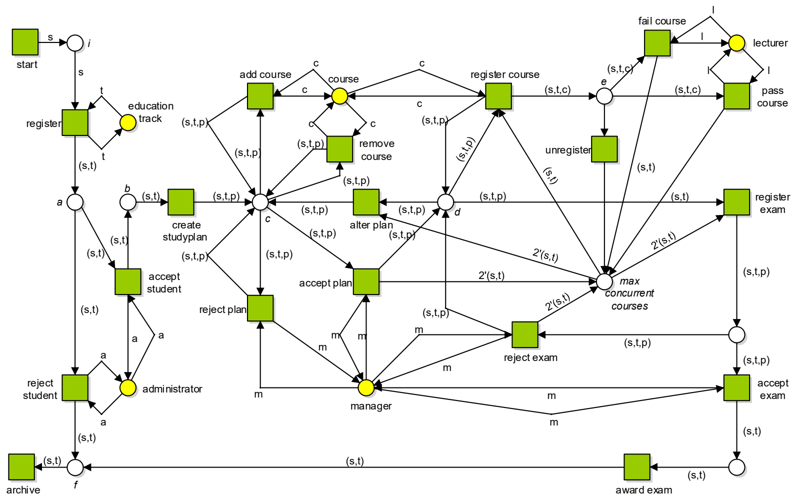

Figure 3: PNID of the process a student follows at PTI.

in place f). Starting with two tokens in place i, the model eventually marks place *max concurrent places* with four tokens. Now, each student starts following a course, i.e., firing *register* twice. As two tokens remain in place *max concurrent places*, transition *register exam* remains enabled. However, if considering the students in isolation, this transition would not have been enabled. Hence, classical Petri nets give an over-approximation of the possible transition firings.

Different approaches exist to represent cases in Petri nets. For example, cases can be encoded in colored Petri nets (CPN), as done in e.g., [8]. However adding data in general breaks soundness: models that are sound without considering data can become unsound, and vice versa. Another approach is followed in the class of ν-PN, in which tokens carry an identifier. A marking then maps each place to a bag 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. In our extension, *Petri nets with identifiers* (PNID) [12], each token represents a vector of identifiers. Each arc is labeled with a vector of variables. Similar to ν-PN, a mode instantiates the variables to an identifier. Vectors of identifiers have the advantage that a single token can represent multiple objects or entities at the same time. Each place in a PNID has a *cardinality*, defining the size of the identifier vectors tokens carry in that place. Tokens carrying vectors of size 0 represent classical – black – tokens. Arcs are labeled with vectors of variables. The size of the vector on the arc is implied by the cardinality of the place it is connected to.

3.1 Formalization of PNIDs

Recall that $\mathcal{I}$ denotes the countably infinite set of identifiers. Places have a *cardinality*, defining the color $C(p)$ of each place p. Arcs are labelled with a variable vector over the non-empty set of variables Σ.

Definition 3.1 (Petri net with identifiers)**.**
A *Petri net with identifiers* (PNID) N is a 5-tuple (P, T, F, α, β), where:
1. (P, T, F) is a Petri net;
2. $\alpha \colon P \to \mathbb{N}$ defines the *cardinality* of a place, i.e., the length of the vector of identifiers carried on the tokens residing at that place;
3. 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 $p \in P, t \in T$.

The color of a place $p \in P$ is defined by $C(p) = \mathcal{I}^{\alpha(p)}$. Given a transition $t \in T$, $I(t) = \{v \mid v \in \text{Rng}(\beta((p,t))), (p,t) \in F\}$ denotes the set of *input variables*, and $O(t) = \{v \mid v \in \text{Rng}(\beta((t,p))), (t,p) \in F\}$ its set of *output variables*. The set of all variables of t is defined by $\text{var}(t) = I(t) \cup O(t)$. The set of *creator variables* of t is defined by $\text{new}(t) = O(t) \setminus I(t)$. ⌟

In a PNID, each token is represented by a vector of identifiers. Consequently, a marking of a PNID is a bag over vectors of identifiers.

Definition 3.2 (Marking)**.**
A *marking* of PNID $N = (P, T, F, \alpha, \beta)$ is an element $m \in \prod_{p \in P} \mathbb{N}^{C(p)}$. The pair (N, m) is called a *marked PNID*. The function Id returns all the identifiers present in marking m, i.e., $\text{Id}(m) = \{a \in \mathcal{I} \mid \exists_{p \in P, v \in C(p)} [a \in \text{Rng}(v) \wedge m(p)(v) > 0]\}$. ⌟

Consider again the net in Fig. 3. This net is a PNID. The first transition, *start*, creates a token with a single identifier, representing a person entering the institute. The place *education track* contains all the tracks PTI offers. Firing transition register, models that the student chooses a track, t, and registers for that track. The result is a token with two identifiers: one for the student, and one for the education track. As for the data model in Fig. 1, a student is the combination of a person and a track. Next, the student creates a study plan. As adding courses to the plan, requires information about the plan, tokens in place c are the combination of a student (i.e., a person and a track), and a plan. Tokens in place d resemble the student together with the approved plan. As for counting the number of courses a student follows concurrently, the plan is not required, place max concurrent courses only carries tokens representing the student. Similarly, place e represents students following a course, and hence carries three identifiers: the student (person and track), and the actual course.

The *mode* is a function that instantiates an identifier for each variable, such that no two variables are instantiated to the same identifier. The mode chooses which tokens are taken from the input places, and which tokens will be produced. A mode is called a (t, m)-mode for a transition t and a marking m, if each creator variable receives a fresh, new identifier, and all other variables of t are instantiated to identifiers present in m. Each $(t - m)$-mode implies a *binding*, indicating which tokens are consumed and produced. If all tokens to be consumed are present, the transition is enabled.

Definition 3.3 (Mode, binding, firing rule)**.**
Let (N, m) be a marked PNID with $N = (P, T, F, \alpha, \beta)$, and let $t \in T$ be a transition.

A *mode* is a function $\nu : \Sigma \to \mathcal{I}$. It is called a (t, m)*-mode* iff $\nu(x) \notin \mathrm{Id}(m)$ for all $x \in \mathrm{new}(t)$, and in addition for all $y \in \mathrm{var}(t)$, if $\nu(x) = \nu(y)$, then $x = y$. We lift the mode to vectors, and sequences, by applying the mode on each of the constituents. The set of all (t, m)-modes is denoted by $V(t, m)$.

A *binding* for transition t with mode ν is a pair $(q, r) \in \prod_{p \in {}^\bullet t} \mathbb{N}^{C(p)} \times \prod_{p \in t^\bullet} \mathbb{N}^{C(p)}$, such that for $p \in {}^\bullet t$, $q(p)(a) = F(p, t)$ iff $a = \nu(\beta((p, t)))$ and $q(p)(a) = 0$ otherwise, and for $p \in p^\bullet$, $r(p)(a) = F(t, p)$ iff $a = \nu(\beta((t, p)))$ and $r(p)(a) = 0$ otherwise.

If such a binding exists the transition is called *enabled*. An enabled transition may *fire*, resulting in marking m', denoted by $(N : m \xrightarrow{(t,\nu)} m')$, with $m + r = m' + q$. ⌟

3.2 Expressiveness

Although PNIDs use vectors of identifiers, their expressive power is similar to the class ν-PN. As a result, reachability is undecidable, whereas coverability, boundedness and termination are decidable [18].

Theorem 3.4 (Expressiveness PNIDs)**.**
PNIDs are bisimilar to ν-PN. ⌟

Proof. (*sketch*) $(\Rightarrow)$ Let (N, m) be a marked PNID, and $k \in \mathbb{N}$ be its largest cardinality. A bijection exists that maps each vector $\mathcal{I}^l$, for $0 \leq l \leq k$ to an identity in a new universe $\mathcal{I}'$ (i.e., $\mathcal{I} \cap \mathcal{I}' = \emptyset$). Similarly, a bijection g exists that maps variable vectors in β to new variables, such that if $\beta(f) = \beta(f')$, then $g(f) = g(f')$. Then, the PNID is simulated by the resulting ν-PN. $(\Leftarrow)$ By definition, as any ν-PN is also a PNID. □

One of the main reasons for the undecidability result of ν-PNs lies in the absence of any structure in the identifiers. Consequently, each transition creating new variables has in any marking m an infinite set of $(t - m)$-modes. In fact, providing a structure, such as the natural numbers, reachability remains decidable, as there is always an upper bound on the number of identifiers possible, as shown in [19]. A similar result holds for PNIDs, cf. [22]. Although this translation always results in a classical Petri net, it results in an exponential explosion of the resulting net, as each place is duplicated for each possible token vector.

Theorem 3.5 (Reachability when utilizing natural numbers)**.**
If the natural numbers are used as identities, i.e., $\mathcal{I} = \mathbb{N}$, together with a creation strategy based on the total order implied by the natural numbers, then PNIDs are bisimilar to classical Petri nets. ⌟

Proof. (*sketch*) Create a counter place c that always contains the last identity being created, and connect it to all transitions t with $\mathrm{new}(t) \neq \emptyset$. The mode instantiates variables such that always the next free variable is used, i.e., the next identifier from the counter place. This way, the counter place always gives an upper bound to the identifiers ever created, and thus the resulting net can be transformed into a finite classical Petri net [22]. □

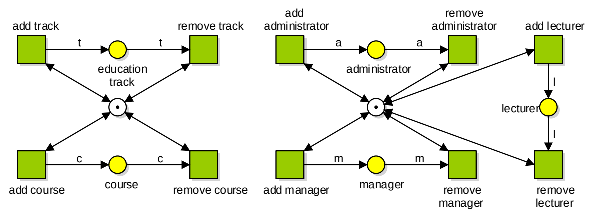

Figure 4: PNID of the secondary processes at PTI.

$\mathrm{insert}(Person, (s))$ $\mathrm{insert}(registers, (s, t))$	$\mathrm{insert}(Person, (520639))$ $\mathrm{insert}(registers, (\mathrm{IS}, t))$

Figure 5: Abstract transaction for *register*, and its instantiation with mode $\{s \mapsto 520639, t \mapsto \mathrm{IS}\}$.

4 Synergy of Data and Processes

In BPMSs such as Bizagi, Bonita BPM and ProcessMaker, creating an information system consists of several intertwined steps, resulting in separate deliverables that together form the system. These deliverables include a data model describing the domain and data storage, a (set of) process models describing the information streams within the organization, and a set of forms and services for each of the activities in the processes.

Based on the data model, each transition is specified with an *abstract transaction* that describes how the transition manipulates the population. An abstract transaction uses the same variable set as the Petri net. Similar to creating a binding for changing the marking, the mode is used to calculate the transaction by instantiating the abstract transaction with the same mode.

Definition 4.1 (Abstract transaction)**.**
Let $D = (T, R, \rho, \Psi)$ be a data model, and let Σ denote a nonempty set of variables. An *abstract transaction* is a sequence of operations $o \in \left(R \times \{\oplus, \ominus\} \times \bigcup_{n \in \mathbb{N}} (\Sigma \cup \mathcal{I})^n\right)^*$, using both variables from Σ, as identifiers $\mathcal{I}$. An abstract transaction o can be *instantiated* using a mode $\nu : \Sigma \to \mathcal{I}$, denoted by $\nu(o)$, which results in a transaction. The set of all abstract transactions for data model D is denoted by $\Sigma(D)$. ⌟

Consider the PNID depicted in Fig. 3. As shown in Fig. 5, transition *register* adds two facts to denote a student registering for a track.

Starting with a valid population, a transaction specified by the transition should not invalidate the population. Hence, we only allow transitions to fire in our model, if, given some mode, both the transition is enabled, and its corresponding transaction is valid in the current population. This forms the basis of our model of an *Information System*. An information system consists of three elements: a data model, a PNID, and a specification, which maps an abstract transaction to each of the transitions in the PNID.

Definition 4.2 (Information system, semantics)**.**
An *information system* (IS) is a 3-tuple (D, N, S) with $D = (T, R, \rho, \Psi)$ a data model, $N = (P, T, F, \alpha, \beta)$ a PNID, and $S : T \to \Sigma(D)$ a specification. A *state* of an information system is the pair (π, m), with $\pi \in \Pi(D)$ a population, and $m \in \prod_{p \in P} \mathbb{N}^{C(p)}$ a marking. Given markings $m, m' \in \prod_{p \in P} \mathbb{N}^{C(p)}$ and valid populations $\pi, \pi' \in \Lambda(D)$, transition $t \in T$ with (t, m)-mode ν is enabled in state (m, π), iff $(D : \pi \overset{\nu(S(t))}{\longrightarrow} \pi')$ and $(N : m \overset{(t,\nu)}{\longrightarrow} m')$. Its firing results in the new state (π', m'), and is denoted by $((D, N) : (\pi, m) \overset{(t,\nu)}{\longrightarrow} (\pi', m'))$. ⌟

Typically, an organization has multiple processes. For example, for PTI we can have different processes, e.g., the processes shown in Fig. 4. As the union of a set of Petri nets is again a Petri net, the different processes for an information system can be divided into separate models, which are then united into a single PNID. The yellow places are the places that are in both PNIDs, and will be merged in the united PNID of the information system.

4.1 A Spectrum of Information System Models

The semantics of an information system is based on two conditions: first, the transaction specified by the transition should yield a valid population, and second, the transition should be enabled. It is not necessary that the former condition implies the latter, or vice versa. For example, in Fig. 3, if transition register is enabled, its execution will always yield a valid population: adding a new fact for `Person`, and one for `registers` is always allowed. Hence, the transition is *process-driven*: its enabledness in the PNID implies a valid transaction. The other way around, a student is only allowed to create a single study plan. The transaction of *create studyplan*, is only valid if the study plan is accepted, which is only the case if place b is marked. Therefore, transition *create studyplan* is *data-driven*: always if its transaction is valid, the transition is enabled in the PNID.

Definition 4.3 (Data-driven, process-driven information system)**.**
Let (D, N, S) be an information system. Transition $t \in T$ is:

- *data-driven*, if $(D : \pi \overset{\nu(S(t))}{\longrightarrow} \pi')$ implies $((D, N) : (\pi, m) \overset{\nu(S(t))}{\longrightarrow} (\pi', m'))$
- *process-driven*, if $(N : m \overset{(t,\nu)}{\longrightarrow} m')$ implies $((D, N) : (\pi, m) \overset{\nu(S(t))}{\longrightarrow} (\pi', m'))$

for all markings $m, m' \in \prod_{p \in P} \mathbb{N}^{C(p)}$, (t, m)-mode ν and population $\pi, \pi' \in \Lambda(D)$. If all transitions in the PNID are data-driven (process-driven), the information system is *fully data-driven* (*fully process-driven*). ⌟

Any of the transitions in Fig. 4 is *data-driven*: if executing its specification results in a valid population, it is also enabled in the PNID. Hence, the net is *fully data-driven*. Similarly, each of the 'add' transitions in the same PNID are *process-driven*: if it is enabled in the PNID, executing its specification yields a valid population. However, removing facts is not always possible, hence the PNID is not *fully process-driven*.

Transitions are not necessarily either data or process driven. For example, in Fig. 3, transition *register exam* is directly enabled once the study plan has been accepted.

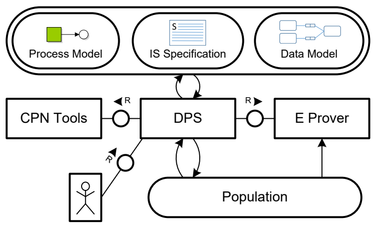

Figure 6: FMC Compositional Structure Diagram of the DPS prototype.

However, the constraints in the data model require a student to have finished all courses specified in the study plan, before the student is entitled to register. Hence, although the transition is enabled in the model (the latter condition), it does not yield a valid transaction, and hence, the transition is not allowed to fire in the information system.

These examples show that any information system balances in a spectrum. One extreme is a fully data-driven model, the other extreme is a fully process-driven model. Notice that one model can be on both extremes at the same time, e.g., in a data model with no constraints, an information system is both fully data-driven as well as fully process-driven. Designers need to balance their models in this spectrum: some modeling requirements are better suited to be addressed in the process model, others in the data model. Making the spectrum explicit, helps in selecting the appropriate technology. For example, if most transitions are data-driven, an active database system [16] would be the preferred choice, whereas if most transitions would be process-driven, a BPMS would be the preferred option.

4.2 Expressiveness of Information Systems

Analysis remains an important aspect of the design of information systems. Notions like soundness assist the designer in detecting errors in the specification. In balancing data and processes, many potential errors are lurking. In our running example, a study plan should contain courses in order to be valid. Adding this constraint to the data model would break the process model: transition *create studyplan* would be dead, as its transaction will always violate this constraint. Such mistakes are easy to make, but difficult to trace. Hence, analyzability of the approach would assist in discovering such errors, and in creating a better understanding of the model. Basing the PNIDs on the natural numbers, and maintaining a mapping of each natural number to its entity type, allows to create a decidable – yet expensive – model for information systems.

Theorem 4.4 (Reachability decidable)**.**
Let (D, N, S) be an information system, and let $\mathcal{I} = \mathbb{N}$, using a creation strategy based on the total order of $\mathbb{N}$, Then reachability of (D, N, S) is decidable. ⌟

Proof. (sketch) By Thm. 3.5, PNID N is bisimilar to some Petri net. Deciding a transition t to be enabled depends on decidability of checking validity of the transaction $S(t)$. As any population is finite, this is decidable. □

4.3 Prototype Implementation

To show the applicability of our presented approach, we have implemented our approach in a prototype: DPS [5]. Its architecture is depicted in Fig. 6 as an FMC Compositional Structure Diagram [13]. The PNID is translated into a CPN-model, and the data model in a set of constraints for the automated theorem prover E [20]. DPS is written in Java, and uses Access/CPN [23] to retrieve the enabled transitions and their binding from the CPN-model. Next, DPS checks for each enabled transition whether it is valid according to the E prover. For this, the constraints need to be specified in the TPTP syntax [21]. The DPS applies the E Theorem Prover by checking validity for every transition and binding individually. Internally, DPS maintains a current population, with an axiom for each of the facts in that population. Validating a transition involves updating the axioms of the population, and running the E-prover for the new population. If a transition is found to be valid, it is added to the list of valid and enabled transitions within the current state. After checking all enabled transitions for validity, a valid transition is chosen to be executed, either manually, or randomly chosen by DPS.

5 Conclusions

This paper proposed an approach to modeling an information system as a synergy of data and processes, i.e., as an integration of a data and process model via a specification on how transitions in the process manipulate the data population. A model of an information system specifies that a transition can execute if it is enabled in the process captured as a Petri net with identifiers, and the corresponding transaction yields a valid population. This approach implies a spectrum of system specifications, from fully data-driven, i.e., a transition is always enabled in the process when its corresponding transaction is valid, to fully process-driven, i.e., if the transition is enabled in the PNID it is allowed to execute. By making this spectrum explicit, designers have a great degree of flexibility when capturing models of information systems. Once captured, the model can aid in deciding on the best technology for implementing the system.

5.1 Related Work

Synergy between data and processes is a well-studied subject, both from the data and process modelling communities. However, the balance between the two is typically left implicit. For example, active databases have mechanisms in place that respond with transactions based on events [16]. Although a process is implied, it is not modeled expicitly. Verification of active databases is possible, e.g., with model checking on termination [17].

In data modeling, different approaches consider process-related aspects. For example, Data Flow Diagrams (DFDs) [24] depict which data elements flow through an information system, defining the input and output relations of activities, but not the order in which these activities can be performed. Consequently, formal analysis, such

[5]The material can be found at `http://architecturemining.org/tools/DPS`

as reachability and termination, of DFDs is not possible. Another approach is the Conceptual Task Model [2], which is based on Petri nets and a similar notion to ORM. Places are either task places, or information places. In this model, data and processes are mingled, leaving the balance between the two implicit.

Other approaches focus more on the process aspects of information systems, leaving constraints on the data model implicit. For example, DB-nets [15] extend ν-PN with three layers: a persistence layer storing the data model, a data logic layer providing a bidirectional "interface", and a control layer for the control flow. Places in the Petri net may have a query that retrieves data from the database instance through the data logic layer. Transitions in the control flow may have a guard on the data, and actions to manipulate the population. Although DB-nets allow to specify a schema in terms of entity types and relations, generic data constraints are not supported. Analysis on the combination of processes and data together is limited to simulation, as the model is richer than ν-PNs.

A different approach on process modelling for information systems is taken in [14], which proposes coloured automata to represent global information about the state, and local information about the states in individual process ordering constraints, expressed in LTL. However, this approach only works in a compliance setting, when the colors are known at runtime.

Our notion of an information system is closely related to Data-Centric Dynamic Systems [3], in which a DCDS specification is maintained by a database manager and the Flow engine, implemented in Java. The approach is similar to [11], in which the Petri net editor Yasper is used with a database engine and a forms engine to create a case handling system. However, both approaches mainly focus on implementing such systems rather than on the design, including modeling and analysis.

5.2 Future Work

Making the balance between processes and data explicit empowers the designer. A methodology is required to support the designer during the process of modeling an information system. Constraints in one model typically affect the other model. Understanding this interplay is crucial to the design of a system. All the transactions of a system need to be validated by an automated theorem prover before one is allowed to execute. If one can prove that validity for an abstract transaction is invariant, this check would not be necessary. This is closely related to proving semantic integrity [10].

Process-related properties such as depth-boundedness and width-boundedness [18] in ν-PN may increase model understandability. To this end, better tool support is needed to assist the designer in analyzing and modeling systems.

In addition, many constraints may be redundant, as they are implied by the other constraints. Minimizing the number of constraints in a system will improve the performance of analysis. Similar to the class of ν-PN, many interesting analysis techniques, such as soundness, are undecidable for PNID. Similar to classical Petri nets, certain subclasses pf PNID may guarantee soundness. Such subclasses remain an open research question. Another idea is to derive construction rules that guarantee correctness to support designers in specifying information systems that are correct by construction.

Despite several existing approaches for modeling information systems, more empirical research is required to better understand the synergy between data and processes, and the effectiveness of corresponding modeling and analysis techniques.

A Preliminaries

Let S be a, possibly infinite set. $|S|$ denotes the number of elements in S. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. Sets S and T are disjoint if $S \cap T = \emptyset$. A *bag* m over S is a function $m : S \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ denotes the set of natural numbers. The set of all bags over S is denoted by $\mathbb{N}^S$. We use $+$ for the element-wise addition of two bags, and $=$, $<$, $>$, $\leq$, and $\geq$, for the comparison of two bags, which are defined in the standard way.

The Cartesian product of A and B is denoted by $A \times B = \{(a, b) \mid a \in A, b \in B\}$. The generalized Cartesian product for a set I and sets A_i, $i \in I$, is defined as: $\prod_{i \in I} A_i = \{f : I \to \bigcup_{i \in I} A_i \mid \forall i \in I : f(i) \in A_i\}$. An element $x \in \prod_{i \in I} A_i$ is called a *vector*. Given some $n \in \mathbb{N}$, and set A, we write A^n for $\prod_{0 \leq i \leq n} A$.

A sequence σ of length $n \in \mathbb{N}$ over some set S is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \ldots, n\}$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. The length of a sequence σ is denoted by $|\sigma|$. The sequence of length 0 is called the empty sequence, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. We write $a \in \sigma$ if $\sigma(i) = a$ for some $1 \leq i \leq n$.

A Petri net N is a tuple $N = (P, T, F)$ with P and T the finite sets of places and transitions respectively, such that $P \cap T = \emptyset$, and $F : ((P \times T) \cup (T \times P)) \to \mathbb{N}$ is the flow relation. Places are drawn as circles, transitions as rectangles. If $F(n, m) > 0$, an arc is drawn from n to m with weight $F(n, m)$. An element $a \in (P \cup T)$ is called a node. The preset of node $a \in (P \cup T)$ is a bag defined by ${}^{\bullet}_{N}a = [b^{F(b,a)} \mid (b, a) \in \mathrm{Dom}(F)]$, and its postset is a bag defined by $a^{\bullet}_{N} = [b^{F(a,b)} \mid (a, b) \in \mathrm{Dom}(F)]$. Given two Petri nets N_1 and N_2, their *union* is the Petri net $N_1 \cup N_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2)$.

A marked Petri net is a tuple (N, m) where $m \in \mathbb{N}^P$ is a marking. Transition $t \in T$ is enabled in (N, m), denoted by $(N : m \xrightarrow{t})$ iff ${}^{\bullet}t \leq m$. An enabled transition may fire, resulting in a new marking m', denoted by $(N : m \xrightarrow{t} m')$ with $m' + {}^{\bullet}t = m + t^{\bullet}$.

References

[1] W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[2] S. Brinkkemper and A.H.M. ter Hofstede. The conceptual task model: a specification technique between requirements engineering and program development. In *CAiSE*, volume 436 of *LNCS*, pages 228–250. Springer, 1990.

[3] D. Calvanese, M. Montali, F. Patrizi, and A. Rivkin. Implementing data-centric dynamic systems over a relational DBMS. In *AMW 2015*, volume 1378 of *CEUR-WS*, 2015.

[4] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, January 1976.

[5] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[6] M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer, 2018.

[7] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure v2.3*. Object Management Group, 2010.

[8] C.W. Günther and W.M.P. van der Aalst. Modeling the case handling principles with Colored Petri nets. In *Practical Use of Coloured Petri Nets*, 2005.

[9] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2010.

[10] M.M. Hammer and D.J. McLeod. Semantic integrity in a relational data base system. In *Proceedings of the 1st International Conference on Very Large Data Bases*. ACM, 1975.

[11] K.M. van Hee, J. Keiren, R.D.J. Post, N. Sidorova, and J.M.E.M. van der Werf. Designing Case Handling Systems. *ToPNoC*, 5100, 2008.

[12] 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.

[13] Andreas Knopfel, Bernhard Grone, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. John Wiley & Sons, 2006.

[14] F.M. Maggi, M. Montali, M. Westergaard, and W.M.P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *BPM*, volume 6896 of *LNCS*, pages 132–147. Springer, 2011.

[15] 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.

[16] N.W. Paton and O. Dìaz. Active database systems. *ACM Computing Surveys*, 31(1), 1999.

[17] I. Ray and I. Ray. Detecting termination of active database rules using symbolic model checking. In *ADBIS 2001*, volume 2151 of *LNCS*, pages 266–279. Springer, 2001.

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

[19] F. Rosa-Velardo, O. Marroquín-Alonso, and D. de Frutos-Escrig. Mobile synchronizing Petri nets: A choreographic approach for coordination in ubiquitous systems. In *MTCoord 2005*, volume 150 of *ENTCS*, pages 103–126. Elsevier, 2006.

[20] S. Schulz. E – A brainiac theorem prover. *AI Communications*, 15(2,3):111–126, 2002.

[21] 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.

[22] J.M.E.M. van der Werf. *Compositional Design and Verification of Component Based Information Systems*. PhD thesis, Technische Universiteit Eindhoven, 2011.

[23] 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.

[24] E. Yourdon. Data flow diagrams. In *Just Enough Structured Analysis*. 2006.