Серия «Математика»
2019. Т. 29. С. 52—67

Онлайн-доступ к журналу:
http://mathizv.isu.ru

ИЗВЕСТИЯ
Иркутского
государственного
университета

УДК 510.62:004.82
MSC 68T27, 68N19
DOI https://doi.org/10.26516/1997-7670.2019.29.52

Towards Semantic Document Modelling of Business Processes *

A. V. Mantsivoda
Sobolev Institute of Mathematics, Novosibirsk, Russian Federation, Irkutsk State University, Irkutsk, Russian Federation

D. K. Ponomaryov
Sobolev Institute of Mathematics, Ershov Institute of Informatics Systems, Sobolev Institute of Mathematics, Novosibirsk, Russian Federation

Abstract. In this paper, we introduce a document–based approach to business process modelling. We argue that declarative semantic modelling should be preferred against the procedural one, which is typically used in software implementations of business processes. Semantic modelling allows for a transparent description of business processes, which is accessible both to manual and automated analysis, verification, and reuse. We present the idea of semantic document modelling and report on its implementation in a web platform, which has been successfully applied to automate business processes of real-world complexity.

The basic feature of our semantic models is executability. This means that having been developed, a semantic model can function as a practical information system. For instance, a model, which semantically depicts business processes for enterprise resource planning can be directly used as an ERP system. This advantage makes the programming stage mainly obsolete and allows for disruptive efficiency/productivity and cost management improvements. The level of the 'executability' of semantic models can range from proof-of-concept prototypes to real-life production-level systems. We have built a semantic modelling management system on top of the Libretto Web Framework. The combination of modelling and web technologies leads to new approaches of web development.

Keywords: semantic modelling, Libretto, document model, buisness process

* The research was supported by the Russian Science Foundation (Grant No. 17-11-01176)

1. Introduction

The choice of an appropriate set of concepts for describing a subject domain is a crucial step in conceptual modelling. It forms the basis of conceptualization, which should be clear to the domain experts (ideally, to a broader group of specialists) and easy to use for describing the ingredients of the domain. One more property, which is sometimes neglected in conceptual modelling is that the constructed domain model should be alive and executable. In other words, instead of only postulating the static part of the domain, it should also be appropriate for representing the dynamic component. This opens the way to replacing programming by modelling, which makes the dynamics of a subject domain more transparent and accessible to formal analysis.

The complexity of any subject domain can not be addressed without appealing to cognitive aspects. A subject domain can be arbitrarily complex by its nature, but humans tend to choose those primitives, which are convenient for cognition. The notion of *a document* is an example of such a primitive, which has been employed for centuries and is evidently important in the context of business processes. It would be fair to say that this concept has been formed by the human experience in information structuring and organization of social processes. Typically, a document has a static and dynamic nature. The static aspect of a document defines its structure and content, while the dynamic one corresponds to modifications and versioning. Numerous activities, e.g., those related to the Enterprise Resource Planning and similar areas, can be naturally described in terms of document processing, including the static and dynamic aspects of documents.

In this paper, we present a document approach to business process modelling, which is based on the notion of document and transaction. A semantic model management system has been built as a PaaS platform on the top of the Libretto Web Framework. The conception of the integration of modelling and web-development techniques was established in [8]. The approach has been successfully implemented in real-world scenarios for automating business processes of large enterprises including budgeting, sales online control, business intelligence, and others.

Within our approach, business processes are represented in terms of semantic document models, which combine the static part, i.e., document types and possible states thereof, and the dynamic part, i.e., rules of transitions between document states. Semantic document models are declarative and executable and retain the meaning of business processes in contrast to the procedural approaches implemented in today's Enterprise Resource Planning systems. Typically, the semantics of interacting business processes is cut into pieces and dissolved in program modules and databases of these systems. Thus, the logic behind the implemented business processes is not accessible to humans and AI tools. In contrast, semantic document

models are fully declarative and open to both, manual and automated analysis.

2. Semantic Document Models

In this section, we formulate the key notions used in our approach. The exposition we follow here is intentionally semi-formal for the sake of clarity. It should be clear from the definitions below that semantic document models [7] allow for a declarative logic-based formulation. An example of such formalization in the framework of the Semantic Programming [3; 4] is given in [9]. We note however that formal logic is elitist and difficult in understanding for non-specialists. So our idea is to employ the notion of document as a metaphor familiar to the people, which would allow them to work with models without having to learn formal logic. Then users can work correctly with such 'documents' without thinking that, in fact, they can be presented as logical formulas. In this way, working with a semantic document model can be viewed as the conventional work with documents.

2.1 Basic Types. Let $\mathcal{B} = \langle B_1, \ldots, B_k;\ \Omega\rangle$ be a multi-sorted algebraic system, which defines the basic data types, where B_i are the main data sets. In practice they can be strings, integers, reals, images, videos, etc. The signature $\Omega = \langle \Omega_P, \Omega_F, \gamma\rangle$ consists of predicate symbols Ω_P, functional symbols Ω_F, and a function γ, which determines the arity of predicate and functional symbols. All elements of all sorts are distinguished (represented by constants, that is, 0-ary functional symbols from Ω_F). Constants are denoted by c, c_i. Elements corresponding to constants are denoted by $\mathbf{c}, \mathbf{c}_i$.

By $\mathbb{B} = \{\mathbf{b}^1, \ldots, \mathbf{b}^k, \mathbf{any}\}$ we denote the set of *names* of the basic datatypes. The name **any** denotes the type of all elements.

All predicate and functional symbols are typed. *The type of a predicate symbol* $p, \gamma(p) = n$ is an expression $\langle \mathbf{b}_1, \ldots, \mathbf{b}_n\rangle$, and $\mathbf{b}_i \in \mathbb{B}$ means that the i-th argument of the predicate corresponding to p must belong to the basic set B_i. The *type of a functional symbol* $f, \gamma(f) = n$ is an expression $\langle \mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{b}_{n+1}\rangle$, where $\mathbf{b}_i \in \mathbb{B}$, and $\mathbf{b}_i$, $1 \le i \le n$, determine the types of arguments and $\mathbf{b}_{n+1}$ determines the type of result of the corresponding function.

The notions of a term, an atomic formula, and the term type are defined inductively as usual.

2.2 Sequences. A sequence is an expression $(e_1, \ldots, e_m)$, where e_i are some elements. Below, constants from Ω_F and references to documents will play the role of these elements. The following equalities hold on sequences

$$(e) = e$$
$$(\ldots, (e_1, \ldots, e_k), \ldots) = (\ldots, e_1, \ldots, e_k, \ldots)$$

The first equality indicates that a singleton sequence is not distinguishable from the element itself. The second equality says that sequences are flat (without nesting, unlike, for example, lists). The empty sequence with no elements is denoted by ( ).

To determine the number of elements in a sequence, we use cardinalities:

$$\mathbb{C} = \{\texttt{()}, \texttt{?}, \texttt{!}, \texttt{+}, \texttt{*}\}$$

where `()` is the empty sequence, `?` a sequence with zero or one element, `!` a sequence containing exactly one element, `+` a sequence with one or more elements, and `*` a sequence with any number of elements.

2.3 Documents. *A document* is the main concept of a semantic document model. The role of documents is similar to objects in the object-oriented approach (OO). We assume the following in our approach:

- a document consists of *fields*, for which type and cardinality are determined.
- *document forms* are templates that describe the structure of documents of a particular type (similar to classes in OO).
- documents can refer to each other through the *enumeration mechanism.*
- each document has a *state.* Transitions between states form the document life cycle. The states are defined in the document form.
- the rules that specify *admissible transitions* of a document from one state to another are defined in the corresponding document form.

Let $\mathbb{I} = \{id_1, id_2, \ldots\}$ be a countable set of new constants, which is called the *set of names (identifiers).* This set is divided into two disjoint countable subsets of form names $\mathbb{I}_F$ and document field names $\mathbb{I}_D$: $\mathbb{I}_F \cap \mathbb{I}_D = \emptyset$, $\mathbb{I}_F \cup \mathbb{I}_D = \mathbb{I}$. In what follows, the form names will determine the types of documents. So, we can define the set of all datatypes as the union of basic type names and form names: $\mathbb{B} \cup \mathbb{I}_F$

A document field description is a triple

$$\mathbf{d} = \langle d, \mathbf{g}, \mathbf{c} \rangle,$$

where $d \in \mathbb{I}_D$ is a field name, $\mathbf{g} \in \mathbb{B} \cup \mathbb{I}_F$ its type, and $\mathbf{c} \in \mathbb{C}$ its cardinality. Document field names will be denoted by d, possibly with indices. The document field description corresponding to d will be denoted by $\mathbf{d}$.

Let us introduce a countable set of new constants, which are called document states: $\mathbb{S} = \{\mathbf{s}^1, \mathbf{s}^2, \ldots\}$ *A transaction description* is a triple

$$\mathbf{p} = \langle \mathbf{s}_{in}, \mathbf{s}_{out}, P(o) \rangle$$

where $\mathbf{s}_{in}, \mathbf{s}_{out} \in \mathbb{S}$ $\mathbf{s}_{in}$ is called the initial state of the transaction, $\mathbf{s}_{out}$ is called the final state, and $P(o)$ is a transaction code. The transaction

changes a document state to s_{out} and performs the set of instructions generated by executing $P(o)$.

The strength of the whole system and computational feasibility of its properties depends on the language, in which $P(o)$ is formulated. Conceptually, it should be quite weak to ensure the elementary nature of transactions. The other important feature of the language is that it must have a clear declarative semantics, e.g., to apply AI tools for controlling and predictions. In practice, we use an automata-like language.

We now define the notion of a *document form*, which determines the structure of documents of the same type. A document form is a tuple

$$\mathbf{f} = \langle\, f, \{\mathbf{d}_1, \ldots, \mathbf{d}_n\}, \{\mathbf{s}_1, \ldots, \mathbf{s}_m\}, \{\mathbf{p}_1 \ldots \mathbf{p}_k\}\rangle$$

where $f \in \mathbb{I}_F$ is the name of the form, $\{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$ a finite set of field descriptions, $\{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$ a finite set of admissible states, $\{\mathbf{p}_1 \ldots \mathbf{p}_k\}$ a finite set of transaction descriptions.

We are now ready to introduce the main concept of this paper, the notion of a *document*. To identify and access documents, an *enumeration* is used. To enumerate documents we use a copy of the set of natural numbers $\mathbb{N}$. The numbers enumerating documents will be called *references*. To distinguish references from ordinary integers, we will write them with the prefix *id*, for example, $id{:}n_1, id{:}5$. The document corresponding to the reference $id{:}n$ is denoted by $\nu{:}n$. If $\mathbb{D}$ is the set of all documents then we have $\nu : \mathbb{N} \to \mathbb{D}$.

A *document field* is a pair

$$\mathfrak{d} = \langle d, w\rangle$$

where $d \in \mathbb{I}_D$ is a field name, and w is a sequence of admissible values. The admissible values of fields are the elements of the basic sets $B_1, \ldots, B_k$ and references from $\mathbb{N}$.

We say that an element e has a type $\mathbf{g}$ w.r.t. an enumeration ν if one of the following conditions holds:

1) $\mathbf{g} = \mathbf{any}$

2) $\mathbf{g} = \mathbf{b}^i$ and $e \in B_i$

3) $\mathbf{g} = f$, $e = id{:}n$, and f is the form name of the document $\nu{:}n$.

A *document* $\mathfrak{o}$ is a structure

$$\mathfrak{o} = \langle\, f, \{\mathfrak{d}_1, \ldots, \mathfrak{d}_n\}, \mathbf{s}\rangle$$

where $f \in \mathbb{I}_F$ is a form name, $\{\mathfrak{d}_1, \ldots, \mathfrak{d}_n\}$ is a set of fields, and $\mathbf{s} \in \mathbb{S}$. In this case we say that the document $\mathfrak{o}$ has the state $\mathbf{s}$ and denote it by $\mathfrak{o}[\mathbf{s}]$.

Let σ be a syntactic structure (e.g., a form or a document). We use operations $id_F(\sigma)$ and $id_D(\sigma)$, which give the set of all form names and field names occurring in σ, respectively. Let us also define

$$\begin{aligned} id_F(\{\sigma_1, \ldots, \sigma_m\}) &= id_F(\sigma_1) \cup \ldots \cup id_F(\sigma_m) \\ id_D(\{\sigma_1, \ldots, \sigma_m\}) &= id_D(\sigma_1) \cup \ldots \cup id_D(\sigma_m) \end{aligned}$$

The *signature of a document model* is a finite set of document forms $\mathbb{M} = \{\mathbf{f}_1, \ldots, \mathbf{f}_l\}$ closed w.r.t. the names: $id_F(\mathbb{M}) \subseteq \{f_1, \ldots, f_l\}$, where f_i is the name of the form $\mathbf{f}_i$.

A *document model* is a finite set of documents

$$\mathcal{M} = \langle \{\mathfrak{o}_1, \ldots \mathfrak{o}_m\}, \nu \rangle$$

with the function ν, which defines an enumeration of documents.

We say that $\mathcal{M}$ is *a model of a signature* $\mathbb{M}$, if for each document $\mathfrak{o} \in \mathcal{M}$ of a form named f the following conditions hold:

1) $\mathbf{f} \in \mathbb{M}$, that is, a form with the name f is defined in the signature $\mathbb{M}$;
2) for each field $\mathfrak{d} = \langle d, w \rangle$ of the document $\mathfrak{o}$, the form $\mathbf{f}$ contains the description $\mathbf{d} = \langle d, \mathbf{g}, \mathbf{c} \rangle$, the size of the sequence w corresponds to the cardinality $\mathbf{c}$, and each element from w has the type $\mathbf{g}$;
3) the state $\mathbf{s}$ of the document $\mathfrak{o}$ is a state admissible in $\mathbf{f}$.

Proposition 1. *The following conditions hold:*

$$id_F(\mathcal{M}) \subseteq id_F(\mathbb{M}) \text{ and } id_D(\mathcal{M}) \subseteq id_D(\mathbb{M})$$

2.4 Transactions. Via transactions, a model $\mathcal{M}$ evolves over time. Transactions are executed sequentially. Each new transaction determines the next time point of the model life cycle. Let us introduce an ordered countable set $\mathbb{T} = \{\mathbf{t_0}, \mathbf{t_1}, \mathbf{t_2}, \ldots\}$, which is called the set of time points. $\mathbf{t_0}$ is called the initial time point. The state of the model at the time point $\mathbf{t}_i$ is denoted by $\mathcal{M}^{\mathbf{t}_i}$. The application of a transaction description $\langle \mathbf{s}, \mathbf{s}', P(o) \rangle$ to a document $\mathfrak{o}$ is defined as follows.

Rule 1. *Document Transaction*

$$\frac{\mathcal{M}^{\mathbf{t}_i}[\mathfrak{o}[\mathbf{s}_{in}]] \qquad \langle \mathbf{s}_{in}, \mathbf{s}_{out}, P(\mathfrak{o}) \rangle}{\mathcal{M}^{\mathbf{t}_{i+1}}[\mathfrak{o}[\mathbf{s}_{out}]]}$$

The model state $\mathcal{M}^{\mathbf{t}_{i+1}}$ is obtained from the state $\mathcal{M}^{\mathbf{t}_i}$ by the execution of instructions generated by $P(\mathfrak{o})$.

The next rule formulates the possibility of an external influence on the model. The document model is not isolated. It is embedded in a context,

which generates various input via creating new documents and changing field values.

The external information sources, which can influence the model, are called *oracles*. For instance, in the scope of a business process management system an oracle can be a data analytics/ prediction tool. It can supply the system with the correspoding instructions depending on the previous and current observations. Each interaction with the outer world is a separate transaction that executes the code provided by the oracle.

Rule 2. *Oracle Interaction*

$$\frac{\mathcal{M}^{\mathbf{t}_i} \qquad P_{oracle}}{\mathcal{M}^{\mathbf{t}_{i+1}}}$$

Here P_{oracle} is a code given by the oracle for the execution. The model state $\mathcal{M}^{\mathbf{t}_{i+1}}$ is obtained from $\mathcal{M}^{\mathbf{t}_i}$ by applying the instructions generated by P_{oracle}.

Rules 1 and 2 are read as follows:

1) P (P_{oracle} or $P(o)$) is executed in the context of $\mathcal{M}^{\mathbf{t}_i}$.

2) The execution of P generates a finite set of instructions $ins_1, \ldots, ins_k$.

3) The instructions are applied sequentially to $\mathcal{M}^{\mathbf{t}_i}$ transferring it to $\mathcal{M}^{\mathbf{t}_{i+1}}$.

4) If all instructions are successfully applied, then the rule is applicable and the model goes into state $\mathcal{M}^{\mathbf{t}_{i+1}}$.

5) If the execution of some instruction fails then the model stays in state $\mathcal{M}^{\mathbf{t}_i}$.

Thus, the set of instructions is atomic: either all instructions are executed, or none of them (the whole computation 'rolls back'). Let us represent a transaction as a triple

$\mathbb{p}_i = \langle \mathbf{t}_i, \langle \mathbf{s}_{in}, \mathbf{s}_{out}, P(o) \rangle, [ins_1, \ldots, ins_k] \rangle$ (for Rule 1)
$\mathbb{p}_i = \langle \mathbf{t}_i, P_{oracle}, [ins_1, \ldots, ins_k] \rangle$ (for Rule 2)

Here $\mathbf{t}_i$ is a time point generated by the transaction, Φ is the applied document for the first rule, and $ins_1, \ldots, ins_k$ are executed instructions. Now the model state at time point $\mathbf{t}_n$ can be implicitly represented as a pair

$$\langle \mathcal{M}^{\mathbf{t}_0}, [\mathbb{p}_1, \ldots, \mathbb{p}_n] \rangle,$$

where $\mathcal{M}^{\mathbf{t}_0}$ is an initial model state (usually an empty model).

Proposition 2. *The explicit model state $\mathcal{M}^{\mathbf{t}_n}$ at time point $\mathbf{t}_n$ can be obtained by the consecutive application of all instructions from the transactions $\mathbb{p}_1, \ldots, \mathbb{p}_n$.*

We found that it is enough to have a quite simple set of instructions:

1) `newdoc(formname)` creates an empty document of a particular form.

2) `set(doc, field, value)` assigns a `value` to a `field` of a document `doc`.

3) `state(doc, s)` sets a new state `s` for a document `doc`.

Though in practice it is useful to have a wider range of instructions, theoretically these three instructions suffice.

2.5 Building Document Models. Let us briefly consider how to construct document models. Documents accompany our entire life and every activity of enterprises. These are passports, bills, receipts, instructions, sheets, etc. Each document reflects a certain facet of our activity and has an appropriate form and structure so as to most accurately reflect this facet. Therefore, to represent a subject domain by a document model we need to

- determine document forms that most precisely describe the domain;
- define document states reflecting the dynamic properties of documents of a given form;
- introduce admissible transitions between states; chains of such transitions reflect the typical processes in the subject domain.

3. Business Processes

Document models can be considered as multi-agent systems, in which documents play the role of agents. Triggering one document and changing its parameters triggers associated agents. A document can change only in one way — by transferring from one state to another. A side effect of this transition is an update of the document itself as well as triggering updates of other documents or creating new ones. In this situation, the model undergoes several iterations of changes until it reaches a new stable state (a fixed point, see [9]). This mechanism resembles the operational semantics of constraint satisfaction techniques.

This approach allows us to introduce the concept of a business process in a quite natural way. In document models, a business process is the life cycle of some document and the states of this document correspond to the stages of the unfolding business process. It is important that this definition is very close to human understanding: in the real world business processes are always accompanied by documents and the document flow is an important part of an implementation of a business process. We define a business process as follows.

A *business process model* is a triple

$$\langle f, s_{start}, s_{fin} \rangle$$

where f is a document form and $s_{start}, s_{fin} \in \mathbb{S}$ are admissible states in f. We call them the initial and final states of the business process, respectively. Let $\mathfrak{o}$ be a document of the form $\mathbf{f}$. A *business process* of $\mathfrak{o}$ implementing a model $\langle f, s_{start}, s_{fin} \rangle$ is a sequence of transactions of $\mathfrak{o}$:

$$\mathfrak{o}[s_{start}] \to \mathfrak{o}[s_1] \to \ldots \to \mathfrak{o}[s_n] \to \mathfrak{o}[s_{fin}]$$

which starts, when $\mathfrak{o}$ has the initial state s_{start}, and moves $\mathfrak{o}$ to the final state in such a way that $s_i \neq s_{fin}$ for each $i, 1 \leq i \leq n$.

For instance, in bSystem (our software platform for semantic document modelling, see below), we can implement a simple book library management system in roughly 20 minutes. It suffices to introduce three basic document forms: a bibliography card for books, a document form for reader's data, and a form for issuing a book to a reader. For these three forms we have three business processes. The first one corresponds to the life cycle of books, the second one to the readers management, and the third one represents the main business process of issuing a book to a reader and getting it back. bSystem implements a solver, which executes the constructed document models in order to compute the required state of a document model after executing a sequence of instructions. Thus, it suffices to build a declarative description of document forms (including transactions) to obtain an operating library management system.

4. Smart Contracts

A smart contract is a computer protocol intended to facilitate, verify, or enforce the negotiation or performance of a contract [11]. While a standard contract textually specifies the terms of a relationship (enforceable by law), a smart contract enforces it via automated support of contract operations and saving the history of executed operations in a trusted (e.g., blockchain based) ledger. Smart contracts are positioned to play the role of a glue, which connects scalable decentralized economic spaces comprising many companies, industries, and customers, regardless of the level of trust between them. However, we evidence that today's implementation of smart contracts suffers from the same problems as business automation.

The logic of business processes is dissolved in a program code of a contract implementation and thus, a declarative 'what' turns into an imperative 'how'. The code is no longer easily accessible for human or automated control. For a Turing-complete language (e.g., Solidity [12]) automated code analysis is impossible even theoretically due to the insolvability of

basic algorithmic problems. More importantly, contracts-as-programs cannot be elegantly combined with traditional business components based on document workflow and in order to use contracts in traditional businesses one needs a programmer as an interpreter. Finally, separating smart contracts from business processes of a company and placing them on an external processing platform means transferring confidential data beyond the security boundaries of the company, which is a risky and hard decision for management.

We argue that programming in smart contracts should be replaced by semantic modeling. This step allows us to

- integrate smart contracts with AI tools;
- significantly simplify the perception of contracts by non-programmers;
- integrate smart contracts into the environment of traditional businesses;
- make contracts more transparent;
- explicitly control the business semantics of contracts;
- ease the legal problems associated with smart contracts.

Using our terminology, the notion of smart contract is easily formulated as follows.

A *smart contract* is a business process model, the instructions of which are stored in a ledger ensuring decentralized trust.

This approach makes the interaction of smart contracts and internal business processes seamless and integrates smart contracts into the company workflow.

5. bSystem and Locally Simple Models

We have implemented the document model technology within bSystem, a cloud platform, which facilitates company digitalization via construction of document models for business processes. bSystem has been built on the top of the Libretto Web Platform, and this brings the full range of instruments necessary for developing real-life web services. As document models are executable, they can be directly used for business process management. bSystem provides special tools for semi-automated wrapping of document models into web services. These services can substitute various ERP, CRM, financial, and accounting systems, which are widely used by businesses today. Document modeling has strong advantages over standard approaches based on programming. The development and maintenance of services based on document models is much more efficient compared to conventional software systems. A document model can combine all the operational components of a company (sales, logistics, accounting, customer relationship, etc.) and provides a single digital pool of the company. This

allows for automated management of complex 'multi-component' business processes. Careful preservation of their semantics allows for incorporating AI tools, e.g, for knowledge discovery and prediction, as well as integration of operational management with business intelligence. In other words, document models allow the management to gain really full control over the company.

bSystem is primarily focused on building *locally simple models* (LSM) [6;7]. LSMs are arbitrarily complex models that are assembled from reasonably simple submodels as components. From our point of view, such models most accurately reflect businesses: the structure and business processes in, e.g., Boeing company are immensely complex, but each job in it is comprehensible for a single person or a group of people. A LSM consists of three types of submodels:

- *A locale*, which is a submodel describing a relatively closed and isolated area of the global domain. In particular, it can be the locale of product distribution, the locale of personnel management, the customer relationship locale, etc. Locales are the main building blocks for LSMs.
- *An interface*: an LSM is built through the integration of locales. However uncontrolled merging is not efficient: first, it increases the complexity of the resulting 'mixture' and, second, it does not correspond to practice (nobody mixes the accountant's competence with that of a merchandiser). Therefore, locales 'communicate' through *interfaces*, which are mutual submodels of locales.
- *An oracle* is a pseudo-model which is interpreted within the LSM as *an interface*, but actually provides interaction with the outer world, e.g., a person's workplace, a programming robot, an external IT system such as data analytics/prediction software or IoT (e.g. an automated weather station or a point of sale).

All industrial applications mentioned below have LSMs as their kernel.

6. Applications

Using bSystem, we have implemented a number of industrial-scale services. In particular, we have developed a service for sales management for a large regional retailer. The service processes more than 200,000 sale receipts per day and allows the company management to use online monitoring of sales in 255 supermarkets and discounters.

Our other application supports personnel management of a company with a staff of over 5000 people. The company management has developed a system of employee motivation, which consists of dozens of different types of incentive rewards, both individual and collective. The descriptions of incentives contain data, which is required in the calculation of salaries. The application automatically uses these declarative data for salary calculation.

It also takes into account complex logical connections between motivations, employee positions, various employee activities, and the general structure of the company. The application does not rely on explicit programming. The model is executable and all the operational work is done by the bSystem solver. This model has been developed in two months by two members of our team [6]. It included several locales and oracles like a motivation system locale, a personnel (jobs and company structure) locale, and a set of oracles for outer components, such as logistics, inventory schedules, etc.

The process of salary calculation is as follows. Payrolls are the main document of the personnel locale intended for salary representation. It has two states 'To-Be-Calculated' and 'Completed'. The first state is the initial one. Salary calculation is performed within the transaction transferring a payroll to the state 'Completed'. The personnel and motivation locales are 'communicating' through an interface, in which incentives are assigned to jobs.

Here are some statistics on the calculation of the annual salary for the retailer's line personnel. The number of jobs is 3,576 from 243 operational branches. The number of formalized incentives is 63. The oracle of inventory contains 87 schedules. Totally, the transaction has generated 1,159,113 documents, and 81,552 payrolls among them. Thus, with the help of modeling, we managed to formally describe a unique and quite complex motivation system working in a company. This motivation system would have been harder to implement within a standard salary calculating software. Importantly, the models (locales), which have been developed within this project, were efficiently applied in other companies.

7. Discussion

Semantic document modeling is similar to developing a specification of an information system and provides a semantic document model as an outcome. The executability of a model means that as soon as we have specified some business process, the programming stage is not needed, since its functionality is automatically derived from the semantic model itself. The global consequence of this idea is that we can replace programming by modeling. The results of this step are quite impressive.

Semantic modeling is much less expensive in development and maintenance, than programming. Unlike programming, modeling is accessible to a much wider range of specialists, e.g., experts in a modeled domain. AI tools and software robots can integrate with semantic models.

The first point means that semantic modeling, where applicable, has significant economic advantages over classical programming. The second point means that programmers are no longer required as intermediaries between domain experts and information systems, since the former can

directly build models. The third point is very important in the light of the 4th industrial revolution, which we witness today. It means that the roles (jobs) formalized in the model can be robotized. Thus, whenever it is possible to apply semantic modeling in some industry, it has a disruptive impact on it, and gives significant competitive advantages.

Our claims here might resemble the fundamental proposal of Tim Berners-Lee [2] to use the potential of knowledge modeling on the web. The idea was to apply the logical means to achieve more efficient data management on the web and to create automated agents. From this idea the concept of Semantic Web has grown. A variety of Description Logics were used as its logical basis [1; 5].

Unfortunately, the Semantic Web did not achieve the goals set by Berners-Lee. It has become a rather restricted mathematical discipline with a limited influence on the outside world and very weak dissemination in practical domains. The major problem with the Semantic Web, in our opinion, is that it considers knowledge processing as a purely mathematical problem, whereas in fact it can only be solved at an interdisciplinary level. The key problems of knowledge processing are perceptual in nature and lie in the field of cognitive psychology, so finding 'yet another' logical formalism does not improve the situation. We have a great spectrum of brilliant logical techniques, but little of them enjoy practical significance.

Here we fall into an 'intellectuality' trap, when trying to model our own thinking instead of looking around. We are fighting for automated reasoning and develop complex knowledge models, but we do not pay attention to the fact that the bulk of tasks that people around us are trying to solve are much simpler. For example, a retailer company as a business model is an extremely complex system. But this model is designed so that each of its components ('tasks') is quite transparent for a fairly wide range of people and can be modeled logically. We face the real complexity of the retailer's model when these locally-simple 'tasks' are used as puzzle pieces to build a holistic business mosaic. And here the complexity can be enormous.

Our hypothesis is that a huge number of practically significant models in our world are locally simple. But we must ensure that a logical formalism we use is comprehensible for the wide range of users. Otherwise, the fate of the Semantic Web will await us. So, ideally, people should not even realize that they work within a formal logical framework. The solution here is to find a metaphor, which is familiar to people, and which would allow them to correctly operate within locally simple models without the need to study logic. As such a metaphor, we have identified the concept of *document*.

8. Conclusion

Currently our main efforts are focused on the practical document modeling of real-world complexity. Our implementation of semantic document modelling allows for working efficiently with models containing hundreds of document forms and tens of millions of documents. In particular, we apply document models in business process management services. In cooperation with our business partners we are implementing a solution for budgeting and commodity circulation for a 255-store retailer, a CRM-model for a chain of furniture stores, a commercial reporting and staff management in food production, and other projects.

We are also trying to integrate document models with machine learning techniques. Unfortunately, it appeared that popular approaches to machine learning, e.g., neural networks and deep learning, cannot directly work with knowledge bases. However, there are great prospects here for methods, which combine logic with probability (see, e.g., [10]). For these methods, semantic document models can serve as ontologies.

References

1. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P., editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2010.
2. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. *Scientific American*, May 2001.
3. Ershov Yu. L., Goncharov S. S., and Sviridenko D. I. Semantic Programming *Information processing 86: Proc. IFIP 10th World Comput. Congress*, 1986, vol. 10, Elsevier Sci., Dublin, pp. 1093–1100.
4. Goncharov S. S. and Sviridenko D. I. Σ-programming. *Transl. II. Amer. Math. Soc.*, 1989, no. 142, pp. 101–121.
5. Horrocks I., Patel-Schneider P., Van Harmelen F. From SHIQ and RDF to OWL: The making of a Web Ontology Language. *Journal of Web Semantics*, v.1, no. 1, pp.7–26.
6. Kazakov I.A., Kustova I.A., Lazebnikova E.N., Mantsivoda A.V. Locally Simple Models Construction: Methodology and Practice. *The Bulletin of Irkutsk State University. Series Mathematics*, 2017, vol. 22, pp. 71-89. (In Russian). https://doi.org/10.26516/1997-7670.2017.22.71
7. Malykh A.A., Mantsivoda A.V. Document Modelling. *The Bulletin of Irkutsk State University. Series Mathematics*, 2017, vol. 21, pp. 89-107. (in Russian). https://doi.org/10.26516/1997-7670.2017.21.89
8. Malykh A.A., Mantsivoda A.V. The Libretto System: Web-resource Development in a Single Model of Data and Knowledge. In Proc. *6th All-Russian Conference of Control Problems (MCPU-2013)*, Gelendzhik, 2013, pp.73-75.
9. Mantsivoda A.V., Ponomaryov D.K. A Formalization of Document Models with Semantic Modelling. *The Bulletin of Irkutsk State University. Series Mathematics*, 2019, vol. 27, pp. 36-54. https://doi.org/10.26516/1997-7670.2019.27.36
10. Vityaev E. The Logic of Prediction. *In Proceedings of the 9th Asian Logic Conference, World Scientific*, 2006, pp. 263–276.

11. Smart Contract. Available at: https://en.wikipedia.org/wiki/Smart_contract (date of access: 20.08.2019)
12. Solidity. Available at: https://ru.wikipedia.org/wiki/Solidity (date of access: 20.08.2019) (in Russian)

Andrei Mantsivoda, Doctor of Sciences (Physics and Mathematics), professor, Irkutsk State University, 1, K. Marks Str., Irkutsk, 664003, Russian Federation, tel.: (3952)521241 Sobolev Institute of Mathematics, Ershov Institute of Informatics Systems, Novosibirsk State University 1, Lavrentyev pr., Novosibirsk, 630090, Russian Federation, tel.: +7 (383) 3306660, Libretto Labs startup (e-mail: andrei@baikal.ru)

Denis Ponomaryov, PhD, Sobolev Institute of Mathematics, Ershov Institute of Informatics Systems, Novosibirsk State University 1, Lavrentyev pr., Novosibirsk, 630090, Russian Federation, tel.: +7 (383) 3306660 (e-mail: ponom@iis.nsk.su)

Received 05.06.19

К семантическому документному моделированию бизнес-процессов

А.В. Манцивода, Д.К. Пономарев

Институт математики им. С.Л.Соболева СО РАН, Иркутский государственный университет

Аннотация. В работе представлен подход к моделированию бизнес-процессов, базирующийся на семантическом представлении понятия документа. Мы уверены, что декларативное семантическое моделирование предпочтительнее процедурного, которое обычно используется в программных реализациях бизнес-процессов. Семантическое моделирование обеспечивает прозрачное описание бизнес-процессов, которое доступно как для ручного, так и для автоматического анализа, верификации и повторного использования. Мы представляем идею семантического документного моделирования и рассматриваем его реализацию на веб-платформе, которая сегодня активно применяется для автоматизации бизнес-процессов реальной сложности.

Основной особенностью наших семантических моделей является исполняемость. Это означает, что разработанная семантическая модель может функционировать как практическая информационная система. Например, модель, которая семантически описывает бизнес-процессы планирования ресурсов предприятия, может функционировать как практическая ERP–система. Это преимущество делает стадию программирования, в основном, ненужной и обеспечивает резкое повышение эффективности / производительности и улучшение управления затратами. Уровень "исполняемости"семантических моделей может варьироваться от опытных образцов до реальных систем индустриального уровня. Система управления семантическими моделями была нами построена на базе Libretto Web Framework. Комбинация технологий семантического моделирования и веб-технологий ведет к новым подходам к веб-разработке.

Ключевые слова: семантическое моделирование, Libretto, документная модель

Список литературы

1. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P., editors. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edition, 2010.
2. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. Scientific American, May 2001.
3. Ershov Yu. L., Goncharov S. S., and Sviridenko D. I. Semantic Programming Information processing 86: Proc. IFIP 10th World Comput. Congress. vol. 10, Elsevier Sci., Dublin, 1986. pp. 1093–1100.
4. Goncharov S. S. and Sviridenko D. I. Σ-programming // Transl. II. Amer. Math. Soc. 1989. no. 142. pp. 101–121.
5. Horrocks I., Patel-Schneider P., Van Harmelen F. From SHIQ and RDF to OWL: The making of a Web Ontology Language // Journal of Web Semantics, v.1, no 1, pp.7–26.
6. Казаков И.А., Кустова И.А., Лазебникова Е.Н., Манцивода А.В. Построение локально-простых моделей: методология и практика // Известия Иркутского государственного университета. Серия Математика. 2017. Т. 22. С.71-89. https://doi.org/10.26516/1997-7670.2017.22.71
7. Малых А.А., Манцивода А.В. Документное моделирование // Известия Иркутского государственного университета. Серия Математика. 2017. Т. 21. с.89-107. https://doi.org/10.26516/1997-7670.2017.21.89
8. Малых А.А., Манцивода А.В. Система Libretto: разработка веб-ресурсов в единой модели данных и знаний // 6-я Всероссийская конференция по проблемам управления (МКПУ-2013). Геленджик. с.73-75.
9. Mantsivoda A.V. Ponomaryov D.K. A Formalization of Document Models with Semantic Modelling // The Bulletin of Irkutsk State University. Series Mathematics, 2019. vol. 27. pp. 36-54. https://doi.org/10.26516/1997-7670.2019.27.36
10. Vityaev E. The Logic of Prediction // In Proceedings of the 9th Asian Logic Conference, World Scientific. 2006. pp. 263–276.
11. Smart contract [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Smart_contract (дата обращения: 20.08.2019).
12. Solidity [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Solidity (дата обращения: 20.08.2019).

Андрей Валерьевич Манцивода, доктор физико-математических наук, профессор, Институт математики,экономики и информатики, Иркутский государственный университет, Российская федерация, 664003, Иркутск, ул. К. Маркса, 1, Институт математики им. С.Л.Соболева СО РАН, компания Libretto Labs тел.: (3952)521241
(e-mail: andrei@baikal.ru)

Денис Константинович Пономарев, кандидат физико-математических наук, Институт математики им. С.Л. Соболева, Инстиут Систем Информатики им. А.П. Ершова, Новосибирский государственный университет, Российская федерация, 630090, Новосибирск, пр. Лаврентьева, д. 6 тел.: (383)3306660
(e-mail: ponom@iis.nsk.su)

Поступила в редакцию 05.06.19