An Approach to Multi-Domain Data Model Development Based on the Model-Driven Architecture and Ontologies

Denis A. Nikiforov, Igor G. Lisikh, Ruslan L. Sivakov

Centre of Information Technology, Ekaterinburg, Russia

Denis.Nikiforov,Igor.Lisyih,Ruslan.Sivakov@centre-it.com

Abstract. To date, there are many diverse data representation technologies (EDIFACT, XML, JSON, CSV, relational model, NoSQL). Transition to new technologies or the integration of information systems based on different technological stacks is a complex and expensive process. Platform-independent models take an important role in this process. The structure of such a model is described in this article. However, given the data model has been created at the junction of different domains, it may be not enough. In such case, a one more step of abstraction and a movement to the computation-independent model is required. The authors propose to create it in an ontological form.

Keywords: ontology, model-driven architecture, platform-independent model, computation-independent model, data model

1 Introduction

When developers create complex, heterogeneous, distributed information systems, they face a number of questions: which technologies for data storage and transfer to choose, how to ensure data-model consistency across different participants of information exchange, and how to simplify future maintenance of the system under development. In order to assist the developers of such information systems, the OMG consortium has developed the model-driven architecture [1]. This architecture considers an information system as a set of models and the development process is transformation of some models into others.

The architecture is not a technical specification. It describes only basic principles. Specifically, it describes platform-dependent, platform-independent, computation-independent models without governing the structures thereof. In other words, developers choose existing or create new modeling languages (metamodels) for each specific information system. The NIEM specification [2] is an example of such a metamodel. It describes a platform-independent metamodel and a platform-dependent metamodel as well as the rules for transforming instances of the former into instances of the latter. We have also developed a similar platform-independent metamodel as well as the rules for transforming its instances into platform-dependent models (an XML schema and an ER model).

As a rule, the platform-independent model solves a considerable number of problems related to the development and maintenance of an information system without the need to provide the third level of abstraction (in the computation-independent model). However, if information exchange is sufficiently complex and covers multiple domains, then the computation-independent model is required.

The purpose hereof is to describe the problems occurring in the data-modeling process that could indicate the need for the computation-independent model. We also show that such a model is actually an ontology. We hope that our experience will help the developers of complex, heterogeneous, distributed information systems to take correct architectural decisions. We will try to answer the question whether and why ontologies are needed for data modeling.

2 The Platform-Dependent Data Model

Let us consider the following example. Some organization produces shrimps at its aquafarm and sells these to another organization. If the organizations reside in different countries, then this process additionally involves customs, sanitary and veterinary, transport and other control authorities. All the stages of this process are carried out together with the storage, transfer, and processing of data on the consignor, consignee, forwarder, transport vehicle, cargo, and other objects (Fig. 1).

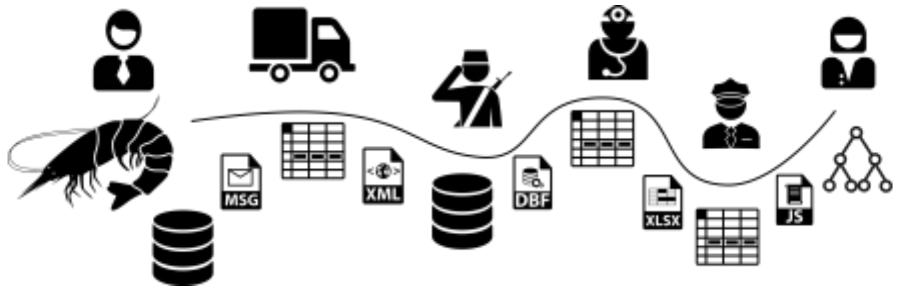
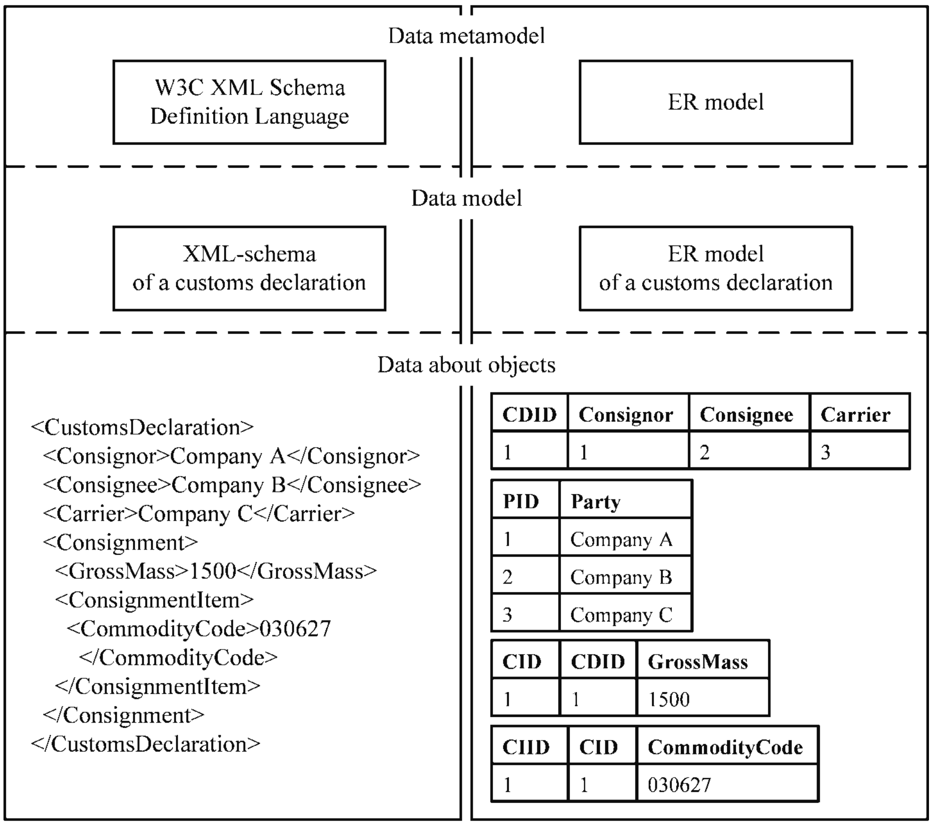


Fig. 1. The process of delivering the commodity and its accompanying data

The participants process data on the same objects. However, they can use different technologies for data storage (RDBMS, NoSQL, Excel) and transfer (XML, JSON, CSV). It means that, in the general case, every participant can have its own platform-dependent data models.

For example, an applicant can file a customs declaration in electronic form in the XML format, whereas a customs authority can store the same data in a relational DB (Fig. 2). In this case, they need two platform-dependent data models: an XML schema and an ER model, respectively.

In order to enable the customs authority to extract the data from the XML message and save the data in its DB, another model mapping XML elements to DB fields is required, and such a model always exists. Even if operators manually input details into the DB, this model exists in their minds or as text instructions. In the general case, the number of mapping models is proportional to the square of the number of exchanging information systems. Obviously, the integration of such heterogeneous information systems with different data storage, transfer and mapping models is rather labor-consuming and error-prone. All these models will have to be changed in case a data structure changes.


Data metamodel
W3C XML Schema Definition Language
ER model
Data model
XML-schema of a customs declaration
ER model of a customs declaration
Data about objects

```
<CustomsDeclaration>
 <Consignor>Company A</Consignor>
 <Consignee>Company B</Consignee>
 <Carrier>Company C</Carrier>
 <Consignment>
  <GrossMass>1500</GrossMass>
  <ConsignmentItem>
   <CommodityCode>030627
    </CommodityCode>
  </ConsignmentItem>
 </Consignment>
</CustomsDeclaration>
```

CDID	Consignor	Consignee	Carrier
1	1	2	3

PID	Party
1	Company A
2	Company B
3	Company C

CID	CDID	GrossMass
1	1	1500

CIID	CID	CommodityCode
1	1	030627

Fig. 2. The example of platform-dependent models and metamodels

3 The Platform-Independent Data Model

Platform-dependent models described above have rather different forms but, on the whole, consider more or less the same data. The development and maintenance of all these models can be significantly simplified if we abstract away from data-representation differences between different platforms and create a single platform-independent model. If one needs to modify a data structure, corrections will have to be made only in this model and then platform-dependent models and mapping models will be automatically generated on its basis. Such an approach is described in the model-driven architecture [1]. An example of the platform-independent model is shown in Fig. 3.

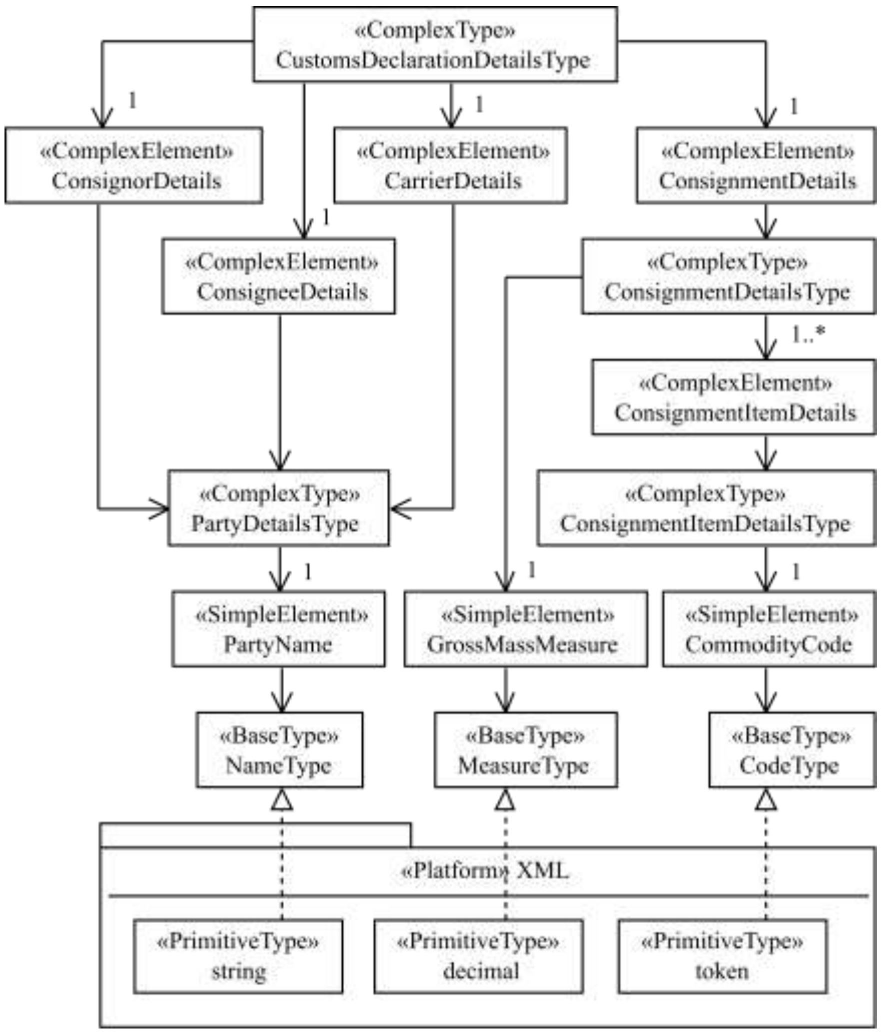

«ComplexType» CustomsDeclarationDetailsType
1
«ComplexElement» ConsignorDetails
«ComplexElement» CarrierDetails
«ComplexElement» ConsignmentDetails
«ComplexElement» ConsigneeDetails
«ComplexType» ConsignmentDetailsType
1..*
«ComplexElement» ConsignmentItemDetails
«ComplexType» PartyDetailsType
«ComplexType» ConsignmentItemDetailsType
«SimpleElement» PartyName
«SimpleElement» GrossMassMeasure
«SimpleElement» CommodityCode
«BaseType» NameType
«BaseType» MeasureType
«BaseType» CodeType
«Platform» XML
«PrimitiveType» string
«PrimitiveType» decimal
«PrimitiveType» token

Fig. 3. The example of the platform-independent data model

In our approach, a UML [3] profile is used to develop platform-independent models (Fig. 4). The profile is based on ISO/IEC 11179 [4] and contains the following stereotypes.

Classifier is any object of a data model.

Namespace is the container for logically-related objects with unique names.

Namespace Subset is the subset of related objects of the Namespace.

Platform is the set of the Primitive Types of a certain platform.

Primitive Type is the primitive data type of a certain platform.

Base Type is the data type used to abstract away from the platform. It is to be implemented by exactly one primitive data type for every platform.

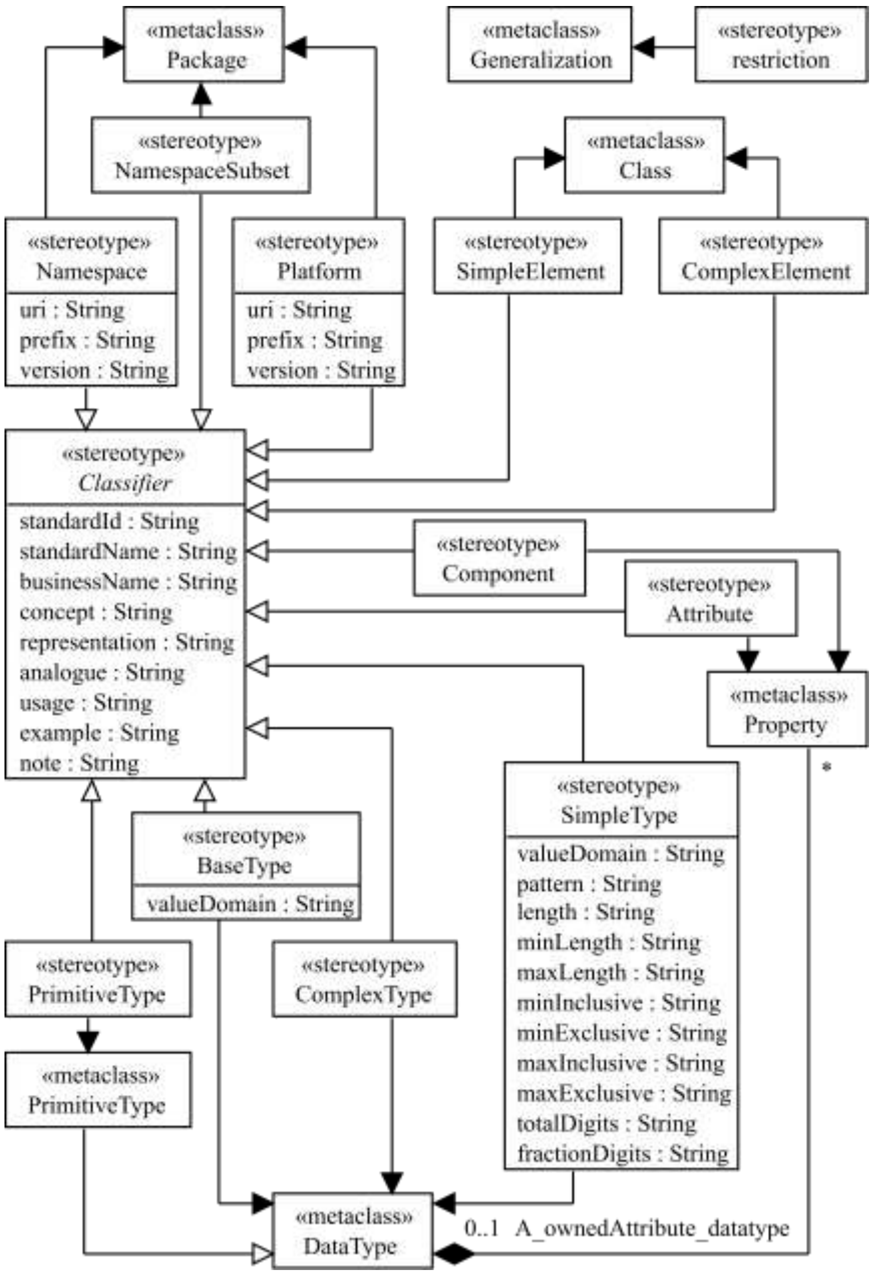


Fig. 4. The profile of the platform-independent data metamodel

Element is the data unit with the designated definition, identifier, representation, and permitted values.

Simple Element is the data element without properties.

Complex Element is the data element with properties.

Simple Type is the set of permitted values of the Simple Element.

Complex Type is the set of the Element's properties that are also Elements, in their turn.

Component is the Element's property.

Attribute is the context characteristic of the data type.

There are two types of inheritance: extension and restriction.

Every data element and type is described as a separate independent entity. This allows generating maximally-normalized platform-dependent models: XML schemas of the "Garden of Eden" pattern and relational models with the relations in the 6th normal form. If necessary, one can generate less normalized models. In other words, such a platform-independent model contains enough information to generate platform-dependent models on its basis with any required characteristics. Further, the metamodel allows describing, using the OCL language [5], business rules that can be converted into SQL or XPath expressions. Model transformation is implemented in [6] using the QVTo language [7], which detailed description falls beyond the scope hereof.

On the one hand, data elements in the described platform-independent model are abstracted away from specific data-modeling languages. They are not XML elements, not entities, not relations but some syntax-neutral data units. On the other hand, the data elements can be re-used in different data structures. This means that the same properties of real-world objects are described using the same data element in all particular data sets (documents, messages). One can conclude that, in fact, delinking of data elements from specific data-representation languages and from their contexts makes the described model conceptual (Fig. 5). The next section shows why this is not the case, and what difficulties it can cause.

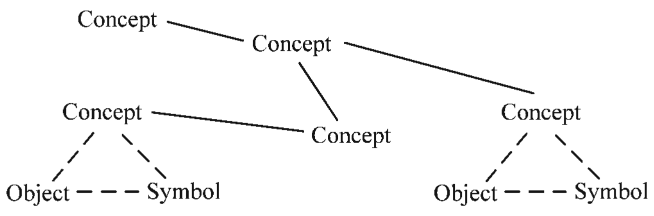


Fig. 5. A conceptual model

4 The Computation-Independent Data Model

The platform-independent data metamodel described above can considerably simplify the development and maintenance of information systems. However, if we attempt to create a single model for different participants (Fig. 1), then the following problems will be encountered.

Firstly, the participants can use different dictionaries and code lists for semantically identical data elements. For instance, a customs authority can code commodities based on the Harmonized Commodity Description and Coding System, whereas a sanitary and veterinary authority can use different classification codes of the products under control. Kinds of submitted documents for which every participant has its own code list is another example. Such semantically identical data elements can be combined in two ways: either to unify code lists or to accompany codes with references to the used code lists.

Secondly, the participants can structure details of the same objects in different ways. For example, customs authority needs to know a commodity code, price, intended use of goods, packaging kind, and mass. Transport control authority is not interested in price and intended use of goods. Sanitary and veterinary control authority checks date of production, best before date, and age and taxon information of shrimps (Fig. 6). All these appropriate authorities keep under control transportation of one and the same object, i.e. shrimps, but we had to create separate data types for each kind of authority. It increases the size of the model and complicates its maintenance. We can try to harmonize these types by one of the three methods: extension (general characteristics are put in a base data type), restriction (all possible characteristics are put to a base data type, and usage of these characteristics in derived data types is restricted), or composition (each characteristic is described as independent object wherein composite data types refer to these global characteristics only).

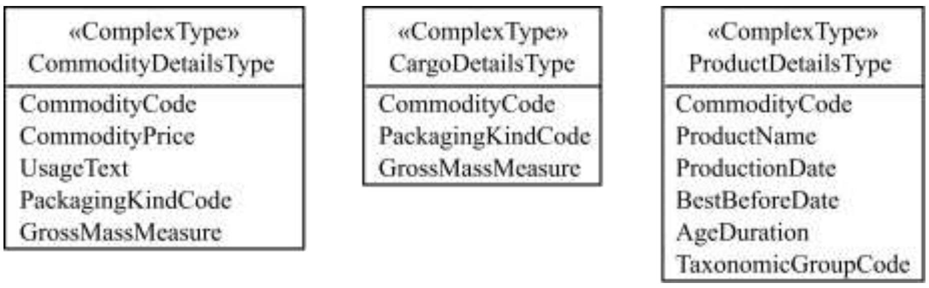


Fig. 6. Different details of one and the same object (shrimps)

Thirdly, the participants use different terminologies. For example, a customs authority mainly uses the term “commodity”, a transport authority uses the term “cargo”, and a sanitary and veterinary authority uses the term “product” to designate the shrimps (Fig. 1). In spite of this, the same object is controlled by all the authorities, which means that only one term should correspond to this object in a single data model. The question is: which of the above terms? Some sources use these terms as synonyms. Other sources state that “a commodity is a product of labor made for sale.” On the other hand, for example, a land lot can be a commodity without being a product of labor. In its turn, a cargo can or cannot be a commodity or a product, and so on.

Although the first and the second problems can be solved within the mentioned platform-independent model, they imply that this model is not as universal as was described above. It can contain several different types and elements in order to represent details of the same object. This fact complicates the integration of information systems and the maintenance of the model.

The third problem explicitly indicates a fundamental shortcoming of the described model. Despite abstracting away from particular data-representation forms (relational model, XML schema, et al.), our model still depends on the usage context, on the domain.

This is connected with the fact that the model describes particular sets of details of real-world objects (particular documents, messages), which can be different for different participants, rather than the objects themselves. A single model can be developed only if the context, documents, messages are disregarded and real-world objects are modeled. The goal of the process described in Fig. 1 is not to transfer the documents but to change the statuses, properties, and relations of the real-world objects. Documents are only a tool to reach the process's goal. If we abstract away from the documents, a single computation-independent model for all the participants can be obtained.

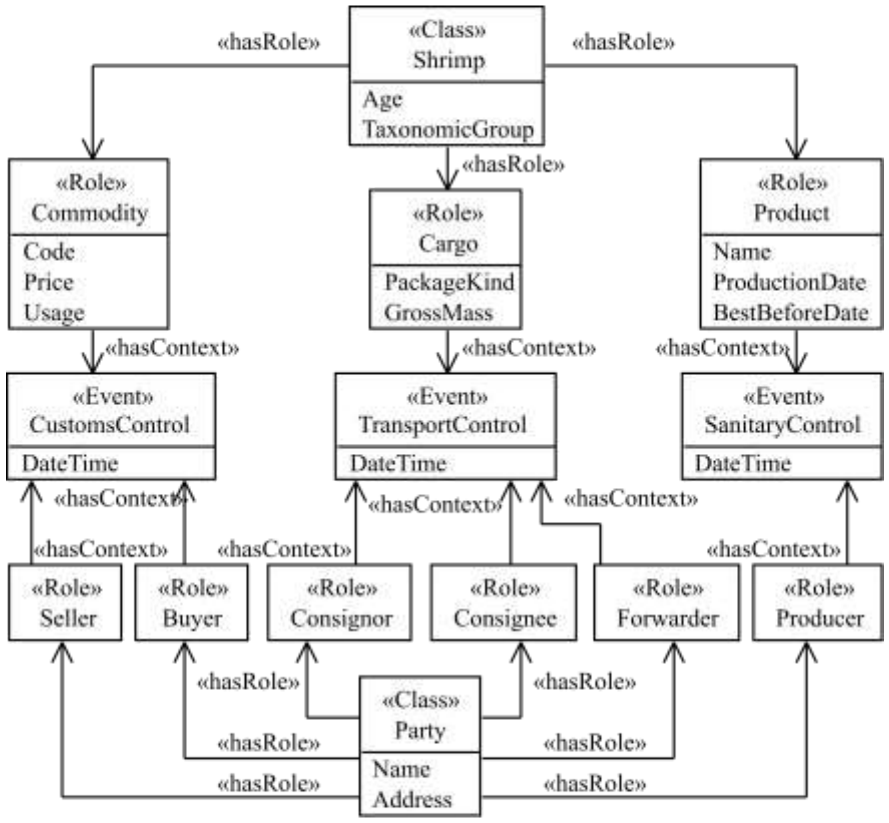


Fig. 7. The example of harmonization of different object' details by means of ontology

Upon reaching the ontological level of modeling, we can conclude that a commodity, a cargo, and a product are not the entity of the object (which data we transfer in the process) but the temporal roles that this object can play. A commodity is the role of the object that has a seller and a buyer. A cargo is the role of the object that has a consignor, a consignee, and a forwarder. A product is the role of the object that has a producer and a consumer. If the difficulties described in this section occur in the data-modeling process, this means that the capacity of the used modeling language is insufficient and an ontology is required.

An example of such ontology is presented in Fig. 7.With our approach; the ontology can include concepts of three types: classes, roles and events. Class describes an intrinsic essence of the object. Role describes temporary characteristics or relations of the object which appear only in a particular context [8, 9]. Context can be presented by a class (for example, an employee is a role of a man appearing in the context of organization) or by an event. Some authors extract one more kind of a context, i.e. *process*. [10]. However, we view processes as compound events [11]. The computation-independent model, firstly, does not have duplicate elements and types, and, secondly, does not change significantly with appearance of new messages or by change of existing documents or messages.

5 Similar Studies

As early as in the 1970-ies, the need for conceptual data modeling was understood, which resulted in the development of ER models and IDEF1X. It is interesting that the conceptual level of the ER model contains minimum information on which basis a more particular and detailed logical model is further created. In other words, the conceptual model is considered as the first approximation to a logical and then physical model. However, in IDEF1X, a conceptual model is needed to integrate several external and internal data models. Moreover, the conceptual model contains a single, integral, sufficiently detailed representation of data rather than the minimum information or the first approximation to other models.

Our approach based on the model-driven architecture is very similar to ER and IDEF1X: it also has 3 levels but slightly different ones. The first level is a computation-independent model describing real-world objects, the relations and properties thereof. In fact, this is a conceptual model but more detailed than the one of the ER model (in addition to entities and relations, it also contains properties) and less detailed than the one of IDEF1X (it does not indicate data types). The second level is a platform-independent model describing particular structures of documents, messages that contain data of real-world objects. This model is something between external and conceptual models as approached by IDEF1X. The third level is a platform-dependent model that is similar to the logical ER model but can also be non-relational. Another difference between our approach and the ER model and IDEF1X is that transformations of platform-independent models into platform-dependent ones are completely formalized and automated. Finally, our platform-independent model describes not only a data schema but also business rules in the OCL language.

NIEM is another analog of our approach [2]. Similarly, it is based on the model-driven architecture, its specification considers platform-independent and platform-dependent models, transformation of one model into the other is described using the QVTo language, and the metamodel is implemented as a UML profile. There are no other similarities, however. Firstly, our approach does not have a profile for platform-dependent models. There is no need for it as such a model is fully automatically generated from the platform-independent model. If a platform-dependent model with other characteristics needs to be generated, then the transformation itself should be changed. Secondly, the platform-independent model of NIEM contains several kinds of complex types (objects, roles, associations, et al.). In our approach, such a division is done on the level of the computation-independent model that NIEM lacks. In contrast to our approach, NIEM does not allow describing business rules using the OCL language and transforming these into XPath expressions.

CCTS is another analog of our approach [12]. Similarly, this one is based on ISO/IEC 11179 [4]. Data is modeled on two levels: core components and business-information entities that, for the purposes of this discussion, correspond to the computation-independent and the platform-independent models used in our approach. The difference is that, in CCTS, basic core components, association core components (and the corresponding business-information entities) cannot be reused. For this reason, only partially-normalized platform-dependent models can be generated from such a platform-independent model (the “Garden of Eden” and the 6th normal form are impossible). The second distinction is that new aggregate core components (and aggregate business information entities) cannot be defined in CCTS by extension or restriction; only qualification can be used that is not supported by any platform that the authors hereof are aware of. The third distinction is that, despite the fact that CCTS allows describing business rules in a data model, business rules are described using the XPath language and supported only in XML schemas but, for example, cannot be transformed into SQL expressions. In our platform-independent model, business rules are described using the OCL language and can potentially be transformed into expressions in the language of any platform.

Finally, Semantic Web gains rather significant popularity recently [13]. RDF and OWL allow to describe real-world objects conceptually and computation-independently. Moreover, several working groups developed data integration standards (ISO 15926, ISO 21127), intended on active usage of these technologies. Some authors propose to use RDF as a universal language for data exchange. [14], [15]. Other authors allow usage of XML or relational models for data exchange, but they propose to transform XML schemas and ER models to ontology [16], [17]. The latter, in turn, can be used for data integration.

Our approach goes in the opposite direction. Firstly, we consider that it is not always possible to be limited to usage of RDF and triplestore in data transfer and storage, sometimes XML schemas or relational models are required. Secondly, computation-independent model describing real-world objects must be primal. A data modeler has to develop platform-independent model describing documents and messages on basis of it. This platform-independent model can be transformed into platform-dependent models (XML schemas, ER models). We found no one article with description of transformation of OWL model to ER model or XML schema. Only backward transformations are available. However, for us, this direct transformation is of the most interest.

6 Conclusion and Follow-up Studies

Data modelers (including authors of this article) often face an issue of motivation for the use of one or another data modeling approach: ontologies, model-driven architecture, IDEF1X, ER. In this article, we tried to correlate these approaches. We also tried to specify problems which can testify a necessity of use of model-driven architecture and ontologies.

Multi-level data models including the conceptual model were developed as early as in the 1970-ies. At large, neither ontologies nor the model-driven architecture introduces anything fundamentally new into this field. These are only steps on the way to generalize, unify existing ideas. Ontologies play exactly the same role as the classical conceptual ER models or IDEF1X in the data-modeling process. In the first case, the ontology is the first approximation to a more particular, detailed, contextual data model that describes the sets of details of real-world objects represented by documents, messages rather than the objects themselves. In the second case, the ontology is a sufficiently detailed data model common for several participants.

If only data-exchange schemas (documents, messages) or particular data-storage schemas need to be designed, then XML schemas, relational models, or other logical models are sufficient, there is no need to use ontologies. If there is a need to unify all these particular data schemas, then we should abstract away from the sets of objects' details and start modeling the real-world objects themselves, and for this purpose, ontologies can be used. The possibilities of the languages intended for data modeling on a logical (rather than conceptual) level are rather limited when there is a need to unify data structures using different code lists, slightly different sets of details, and different terminologies. Such languages allow modeling only particular sets of details of objects rather than the real-world objects themselves.

It should also be noted that the developers of data models based on the model-driven architecture, as a rule, restrict themselves to platform-independent and platform-dependent models thus omitting computation-independent models. The present paper attempts to show that platform-dependent models can be well dispensed with as these can be automatically generated from platform-independent models. However, if the developed model is rather complex, covers multiple domains, the computation-independent model (describing real-world objects rather than particular sets of details thereof) can considerably simplify the development and maintenance of information systems. Furthermore, the computation-independent data model is an ontology.

The metamodel presented in this article has no essential novelty. As already stated, similar metamodels were discussed in [2], [12]. As a rule, these metamodels have only one target architecture. Our approach has a benefit: from one platform-independent model we form not only XML schemas but ER models as well. Secondly, we describe not only data schema but also business rules, and we do it in platform-independent OCL, not in a language of target architecture. Thirdly, we predicate that in some situations one platform-independent model is not enough, and a computation-independent model is required.

Currently, a metamodel is used for data modeling in several domains: customs control, transport control, technical regulation, sanitary and veterinary control, and others.

Our approach to data modeling can be used with other types of B2G or G2G interaction. Developed platform-independent model has around 2,000 elements and types. Around 100 exchange structures were developed on the basis of this model. XML schemas and ER models are formed automatically from this model. Unfortunately, transformation rules [6] are too voluminous to present them in this article. It will be described in details in next articles.

At this date, development of the computation-independent data model is in the initial stage. In next articles, we shall describe in details the model formation rules and rules for creation of platform-independent model on its basis. We shall also try to evaluate at what extent the computation-independent model will help to improve the quality of the platform-independent model.

7 References

1. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1 (2003). http://www.omg.org/cgi-bin/doc?omg/03-06-01
2. Object Management Group: UML Profile for NIEM, version 1.0 (2014)
3. Object Management Group: OMG Unified Modeling Language, version 2.4.1 (2011)
4. ISO: ISO/IEC 11179-1:2004. Information technology – Metadata registries – Part 1: Framework (2004)
5. Object Management Group: Object Constraint Language, version 2.4 (2014)
6. Nikiforov, D.A.: UML Model to XML Schema 1.1 Transformation. doi:10.5281/zenodo.16151 (2013)
7. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, version 1.1 (2011)
8. Pradel, M., Henriksson, J., Aßmann, U.: A good role model for ontologies: Collaborations. In: International Workshop on Semantic-Based Software Development (2007)
9. Henriksson, J., Pradel, M., Zschaler, S., and Pan, J. Z.: Ontology Design and Reuse with Conceptual Roles. In Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems, RR '08, pp. 104–118, Berlin, Heidelberg. Springer-Verlag (2008)
10. Mizoguchi, R., Kozaki, K., and Kitamura, Y.: Ontological Analyses of Roles. FedCSIS, pp. 489-496 (2012)
11. Partridge, C.: Business Objects: Re-Engineering for Re-Use, section 8.3.1.4. Butterworth Heinemann (1996)
12. UN/CEFACT: Core Components Technical Specification, version 3.0 (2009)
13. W3C: Semantic Web. http://www.w3.org/standards/semanticweb/
14. Gorshkov, S. "Business Semantics" practice of application integration using Semantic Web technologies. In international data science conference on Analysis of Images, Social Networks, and Texts (2013)
15. Booth, D., Dowling, C., Fry, C. E., Huff, S., Mandel, J.: RDF as a Universal Healthcare Exchange Language. In Semantic Technology and Business Conference San Francisco, CA (2013)
16. Bedini, I., Matheus, C., Patel-Schneider, P., Boran, A., Nguyen B.: Transforming XML schema to OWL using patterns. ICSC 2011 – 5th IEEE International Conference on Semantic Computing, Palo Alto, United States. pp.1-8 (2011)
17. Myroshnichenko, I., Murphy, M. C. Mapping ER Schemas to OWL Ontologies. In ICSC '09. IEEE International Conference on Semantic Computing (2009)