Elementary Patterns for Converting Textual and Visual Formalisms based on Set Theory and ORM

Sietse Overbeek[1], Patrick van Bommel[2]
[1]Faculty of Technology, Policy and Management
Delft University of Technology, Jaffalaan 5
2628 BX Delft, The Netherlands
S.J.Overbeek@tudelft.nl

[2]Institute for Computing and Information Sciences
Radboud University Nijmegen
Heijendaalseweg 135, 6525 AJ Nijmegen
The Netherlands
P.vanBommel@cs.ru.nl

Journal of Digital Information Management

ABSTRACT: *Textual formalisms, visual formalisms, or both types of formalisms can be used to develop a formal model. The exact syntax and semantics of textual formalisms reduce ambiguity and increase formulation precision. This has as side effect that textual formalisms can become complex and suitable for limited audiences only. Visual formalisms consist of graphical notations with formal semantics. The intuitive and well-organized character of such models improves their comprehensibility and eases communication. Unfortunately, visual formalisms can become unreadable when the complexity or number of formalisms increases. Conversion patterns are introduced to enable conversion of textual to visual formalisms and vice versa. These patterns can be applied to benefit from the advantages of both ways to represent formalisms. The Object-Role Modeling (ORM) language has been used to develop the visual patterns and the textual formalisms are founded in set theory. While the primary focus of this paper is on the theoretic coupling between textual and visual formalisms, the presented patterns are also materialized and evaluated in two applications. First, an ORM model of the ISO/IEC 19763-5 standard to register information of process models is converted to a textual formal model. Second, a textual formal model of an information market is converted to an ORM variant.*

Categories and Subject Descriptors
I.2.4 [Knowledge Representation Formalisms and Methods]; Representation languages: **I.1.3 [Languages and Systems]**

General Terms: Knowledge representation, Formal languages, Modeling languages

Keywords: Object role modeling, Textual formalisms, Visual formalisms, Set theory

Received: 8 September 2010, Revised 19 November 2010, Accepted 26 November 2010

1. Introduction

In software engineering and computer science, *formal methods* are a particular kind of mathematically-based techniques for specifying and verifying complex software and hardware systems [4]. The application of formal methods results in one or more *formal models* that are modeled in a *formal language*. When developing a formal model, one may choose to use textual or graphical techniques (or both) to display formalisms. Formal specifications make use of mathematical notations that offer precise syntax and semantics, which reduces ambiguity when interpreting such specifications [9]. Textual formalisms, however, may be complex in nature and not acceptable to many audiences [5]. Visual formalisms offer graphical notations with formal semantics and also offer the possibility to model intuitive and well-organized formalisms, which improves comprehensibility and eases communication [18]. Unfortunately, diagrams drawn by hand or with a computer-based modeling tool can easily become unreadable when the complexity of the formalisms increases [7].

Conceptual modeling languages that incorporate a formal modeling language are suitable to visualize formalisms, but can also be used to generate textual formalisms from the graphical models [15]. In such a modeling language the syntax and (in case of fully formal languages) the semantics can be coherently formulated in a mathematical language. The Object-Role Modeling (ORM) is such a language which is useful to visualize formalisms because of its formal foundations [11, 6], its demonstrable applications in visualizing formalisms [9], its long running affiliation with the field of conceptual modeling involving varied, often non-technical audiences [5], and its stable attribute-free graphical notation [15]. Objects are treated as concepts in ORM, which makes ORM immune to changes in the model that cause attributes to be remodeled as objects or relationships. A specific application of ORM to visualize formalisms of a theory about the use of modeling knowledge to achieve more effective information modeling support can be found in [27].

In earlier work, we have dealt with the technique of visualizing formalisms by using ORM models [23]. We have shown how textual formalisms as part of a theoretical framework could be converted to visual counterparts. The primarily theoretical research presented here extends this previous work by introducing patterns for converting textual formalisms to graphical representations and vice versa. In section 2, abstract elementary patterns are introduced to understand which elementary textual and visual formalisms can be used to build up a formal model. Because some readers may not be familiar with the main constructs of ORM, a brief summary of those constructs is provided in section 2. Section 3 shows and evaluates how these abstract patterns can be concretely applied to understand their usage. Section 4 shows an overview of research contexts in which the presented technique of converting formalisms has already been applied and a comparison of our work to that of others is made. Before discussing future research directions in section 6, the paper is concluded in section 5.

2. Abstract elementary patterns

Table 1 shows abstract elementary patterns that can be used to build up textual or graphical formal models by using the ORM-based graphical patterns or their textual set-theoretical counterparts. In an ORM model, ovals represent object types (which are counterparts of classes). There are two different sorts of atomic object types: *entity types* and *label types* [11]. Label types can, in contrast with entity types, be represented or reproduced on a communication medium. Text, graphics, sound and video are distinguished depending on the medium. The term multimedia is used as a collective noun. Typical examples of label types are 'Name', 'Number', and 'Code'. A typical example of an entity type would be 'Person'. Boxes represent relations between object types in an ORM model. These relations are dubbed as fact types. In the next section the patterns for conversion of formalisms that incorporate such relations between sets (textually) and object types (visually) are discussed.

2.1 Patterns for conversion of functional relations

The first ORM pattern shown at the top-left of the table visualizes two object types that are related by means of a fact type. The instances of object types $\mathcal{X}$ and $\mathcal{Y}$ play a role in fact type R. For example, object type $\mathcal{X}$ can be 'Person' and object type $\mathcal{Y}$ can be 'Driver's licence'. If fact type $\mathcal{R}$ would be 'Has', then the pattern can be used to show that a certain person would have a driver's licence. More concrete examples of the abstract patterns that are discussed in this section can be found in section 3, when the patterns are applied and evaluated. Two types of constraints have been added to the first pattern, which are the uniqueness and total role constraints. A uniqueness constraint has been added to the role of object type $\mathcal{X}$. This ensures that every instance of object type $\mathcal{X}$ that plays a role in the corresponding fact type is unique. A total role constraint is also added to object type $\mathcal{X}$, ensuring that every instance of object type $\mathcal{X}$ plays a role in the fact type.

Textually, the formalism is verbalized as $\mathsf{R} : \mathcal{X} \to \mathcal{Y}$. Notice that the fact type R can be formalized as a total function. The domain of the function is object type $\mathcal{X}$ and the range of the function is object type $\mathcal{Y}$. Because function R is a total function, every member of the set $\mathcal{X}$ should be related with a member of the set $\mathcal{Y}$. Every member of the set $\mathcal{X}$ may not be related more than once with a member of the set $\mathcal{Y}$. In other words, a total function exactly expresses the constraints shown in the related ORM pattern. For example, driver 'John' has (exactly one) driver's licence.

The following pattern displayed at the top-left of the table shows a label type. The instances of the shown label type may be real numbers within the range [0, 1]. This is the graphical interpretation of a function that takes an element of the set $\mathcal{X}$ as parameter and returns a real number from 0 up to and including 1. Obviously, label types can be introduced that also contain other types of numerical values. For example, label types can be created that contain natural numbers, integers, or rational numbers. Because function R is a total function, every member of the set $\mathcal{X}$ should be related with a real number in the given range and every member of the set $\mathcal{X}$ may not be related more than once with a number. Suppose that object type $\mathcal{X}$ is a 'FuelTank'. If fact type R would be 'HasAmount', then the pattern can be used to show how full or empty a fuel tank of, for example, a car would be by indicating the current amount of fuel in the tank with a real value between 0 and 1, where 0 would be empty and 1 would be full.

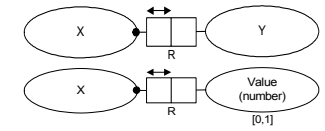
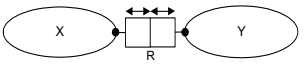
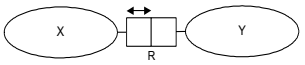
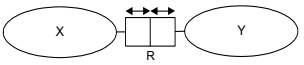
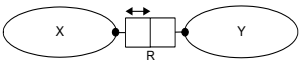
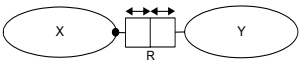
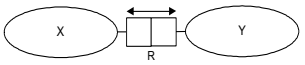
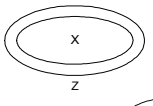
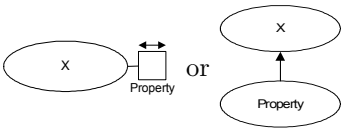
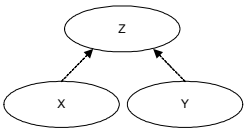
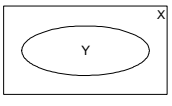
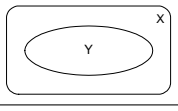
ORM model	Set-theoretic formalism
	$\mathsf{R} : \mathcal{X} \to \mathcal{Y}$
	$\mathsf{R} : \mathcal{X} \to [0,1]$
	$\mathsf{R} : \mathcal{X} \leftrightarrow \mathcal{Y}$
	$\mathsf{R} : \mathcal{X} \rightarrowtail \mathcal{Y}$
	$\mathsf{R} : \mathcal{X} \mapsto \mathcal{Y}$
	$\mathsf{R} : \mathcal{X} \rightsquigarrow \mathcal{Y}$
	$\mathsf{R} : \mathcal{X} \hookrightarrow \mathcal{Y}$
	$\mathsf{R} \subseteq \mathcal{X} \times \mathcal{Y}$
	$\mathcal{Z} \subseteq \wp(\mathcal{X})$
	$\mathsf{Property}(x)$
	$\mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$
	$\forall_{x \in \mathcal{X}} \exists_{\mathcal{Y} \subseteq \mathcal{O}} [\bigcup_{x>0} \mathcal{Y}^x]$
	$\forall_{\mathcal{Y} \subseteq \mathcal{O}} [\mathcal{X} = \wp(\mathcal{Y})]$

Table 1. Abstract elementary patterns

The next pattern shown at the top-left of the table is the same as the first one, except for the additional uniqueness constraint that is added to the role of object type $\mathcal{Y}$. This pattern enforces that all members of the sets $\mathcal{X}$ and $\mathcal{Y}$ should play a role in the fact type and that every relation between two members of these sets may not occur more than once. The textual formalism is verbalized as $\mathsf{R} : \mathcal{X} \leftrightarrow \mathcal{Y}$. This formalism can be illustrated by means of the fingerprint example. Object type $\mathcal{X}$ can be the set of humans and object type $\mathcal{Y}$ can be the set of fingerprints. Every human has a fingerprint, which implies that all members of both sets should occur in the fact type. Suppose for this example that every human has only one unique fingerprint. In practice it is possible to create fingerprints of more than one finger of course, but in this case only the fingerprint of the left thumb is recorded. This means that the relation between a human and a fingerprint of the left thumb should be unique, which can be enforced by the two uniqueness constraints.

The following pattern shown at the top-left of the table is the same as the first one, except for the total role constraint. The textual formalism shows a partial function, which implies that if a member of the set $\mathcal{X}$ is related with a member of the set $\mathcal{Y}$, then it may not be related more than once with a member of the latter set. For example, suppose that object type $\mathcal{X}$ contains instances of cars and object type $\mathcal{Y}$ contains instances of 'air

conditioner types'. If a car is equipped with an air conditioner *type* (this is not a mandatory requirement), then it should only have exactly one air conditioner type. An air conditioner type can, for example, be an automatic air conditioner (climate control) or a manual air conditioner. The next pattern from the top-left of the table is in fact a map. A map is a way of associating unique objects to every element in a given set. So a map $R : \mathcal{X} \mapsto \mathcal{Y}$ from $\mathcal{X}$ to $\mathcal{Y}$ is a function R such that for every $x \in \mathcal{X}$, there is a unique object R(x) in $\mathcal{Y}$. In ORM, a map can be visualized by adding two uniqueness constraints on the relation between both object types. A map is applicable if a car is related with an *instance* of an air conditioner type. If a car is equipped with an air conditioner of a specific type, the unit that is installed in the car can't be installed in another car at the same time. Therefore, two uniqueness constraints are needed on the fact type that relates cars with air conditioners.

Subsequently, a formalism is shown that involves one uniqueness constraint on the role connected to object type $\mathcal{X}$ and two total role constraints. Textually, this pattern can be modeled as $R : \mathcal{X} \rightsquigarrow \mathcal{Y}$. The usage of this pattern can be illustrated when, for example, the members of object type $\mathcal{X}$ are cars and the members of object type $\mathcal{Y}$ are gearbox types that are sold in garages throughout the world. For instance, a gearbox type can be a manual transmission or an automatic transmission. This pattern can be used to express that every car has a gearbox of the types that are sold, but that a gearbox type is not uniquely related to a car.

The textual formalism $R : \mathcal{X} \rightarrow \mathcal{Y}$ can be used to indicate that the relations between members of object types $\mathcal{X}$ and $\mathcal{Y}$ should be unique and also that all members of object type $\mathcal{X}$ should be part of the fact type. This pattern can be useful to show what type of bodywork a car has. A car must have a type bodywork, but can only have one type at the same time. For example, a car can be a hatchback but not a sedan at the same time. Every car also should have some type of bodywork.

The subsequent pattern from the top-left of the table shows a uniqueness constraint spanning the whole fact type. This implies that a combination of an instance of object type $\mathcal{X}$ and an instance of object type $\mathcal{Y}$ may not occur more than once. In textual form, R is in this case a proper subset of the Cartesian product of $\mathcal{X}$ and $\mathcal{Y}$. An illustration of this pattern can be the relation between cars and types of damages. Of course, it is not mandatory that a car has a damage of a certain type. Furthermore, a car can have multiple damages of one type and a damage type may be related to multiple cars.

The last elementary ORM pattern that visualizes a functional relation shows that $\mathcal{Z}$ is a *power type* of object type $\mathcal{X}$, as introduced in [11]. An instance of a power type is identified by its elements, just as a set is identified by its elements (axiom of extensionality) [12]. Set-theoretically, this can be expressed by $\mathcal{Z}$ being a proper subset of the powerset of $\mathcal{X}$. The collection of cars that are owned by a car dealer can be a powerset of the set of cars. So far, we have discussed conversion patterns for conversion of functional relations. However, these aren't the only type of patterns that can be modeled. Non-functional relations between sets or object types can also be modeled and, therefore, the next section discusses patterns for these kind of relations.

2.2 Patterns for conversion of non-functional relations

The first ORM pattern that is introduced for conversion of non-functional relations can be used to show that an instance of an object type expresses a certain *property*. For example, a car may have a property that is has the color red. There are two alternatives to visualize this by using ORM. First, a unary fact type 'Property' can be used that can be populated with instances of an object type that express a certain property. Second, an object type 'Property' can be introduced that is a specialization of another object type [10]. Such a relation implies that the instances of the subtype are also instances of the super type. If object type $\mathcal{X}$ expresses a certain property, then this property can be modeled as a *specialization* of the object type. Textually, this is depicted as Property(x). Furthermore, the union of two sets are modeled by using the *generalization* relation as can be seen in table 1. In that case, object type $\mathcal{Z}$ contains all instances contained in object types $\mathcal{X}$ and $\mathcal{Y}$. Generalization is a mechanism that allows for the creation of new object types by uniting existing object types [12]. For instance, object type $\mathcal{Z}$ can be 'omnivores' while object type $\mathcal{X}$ is the object type of herbivores and $\mathcal{Y}$ is the object type of carnivores. I.e., an omnivore is a combination of a herbivore and a carnivore.

Sequence types can be compared to power types. The differences are that, in the case of sequence types, the ordering of elements is important and elements may occur more than once. An example usage of a sequence type can be when it is modeled how many train compartments a train has and in which order these compartments are connected. An instance of a sequence type is a sequence (tuple) of instances of its element type [10]. The penultimate ORM pattern shown in table 1 shows how such a sequence type is modeled. Here, $\mathcal{X}$ is a sequence type over object type $\mathcal{Y}$. An instance of object type $\mathcal{X}$ is then a row of instances contained in object type $\mathcal{Y}$. For short: $x \in \mathcal{X}$ is a row of instances of $\mathcal{Y}$, where rows with length 1 originate from $\mathcal{Y}$, rows with length 2 originate from $\mathcal{Y}^2$, and so on. Set-theoretically, this implies $\forall_{x \in \mathcal{X}} \exists_{\mathcal{Y} \subseteq \mathcal{O}} [\cup_{x>0} \mathcal{Y}^x]$, where $\mathcal{O}$ is the set of all object types.

Finally, schema objectification allows to define part of a schema as an object type. This object type is referred to as a *schema type* [10]. An instance of a schema type is an instantiation of the associated schema part. The idea is to construct a power type for each object type that is to take part in the schema type. Hence, the schema shown at the bottom-left of table 1 can be equated with the model of figure 1. Schema types are often used for meta-modeling [10].

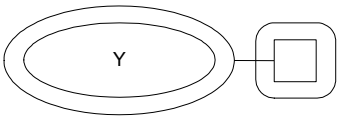


Figure 1. Elementary ORM pattern as an alternative to using a schema type

Suppose that the schema type $\mathcal{X}$ shown in table 1 is a meta-model of 'Car'. This meta-model would include object types such as 'Wheel', 'Tyre', 'Roof', and so on. The schema type can then be instantiated by parts of existing car models that fit the meta-model. Next, by studying textual and graphical formalisms in literature we are able to identify concrete elementary patterns and evaluate the patterns that we have identified so far.

3. Concretization and evaluation of conversion patterns

An ORM model that displays formalisms and a collection of concrete textual formalisms have been identified in literature and will be introduced in this section. These formalisms are utilized to concretize and evaluate the abstract elementary patterns that have been discussed.

3.1 Conversion of graphical to textual formalisms

First, the ORM model will be shown and explained before we will convert it to a textual formal model using the introduced abstract patterns. To concretize the visual patterns displayed in table 1, the ORM model of the ISO/IEC 19763-5 standard as modeled in [26] is used. This standard provides a meta model to register administrative information and meaningful semantics of process models. Such *process models* include workflows, business processes, Web services, software processes, etc. As a meta model, it focuses on the common structural and semantic content of process models rather than their representations. Figure 2 shows the ORM model of the ISO/IEC 19763-5 standard. This model can be converted to a textual formal model by using the abstract elementary ORM patterns of table 1. The results of this conversion can be categorized in groups of functions. The types of functions that can be identified in the ORM model of figure 2 are: total functions, bi-directional total functions, specialization relations, and Cartesian products. Table 2 shows the uni-directional and bi-directional total functions as a result of converting the related visual formalisms found in figure 2. Table 3 shows the specialization relations and Cartesian products as a result of converting the related visual formalisms found in figure 2. Table 4 shows the total functions that are also defined for all possible outputs. This concerns the elementary ORM pattern that has two total role constraints and one uniqueness constraint. This pattern, referred to as the *extended* total function, occurs several times in the ORM model of figure 2 as is shown in table 4. A total number of five elementary patterns have been identified in the ORM model of the ISO/IEC 19763-5 standard that are repeatedly used. Besides that a graphical formal model can be converted to a textual model by using the elementary patterns, it is possible to repeat this exercise the other way around.

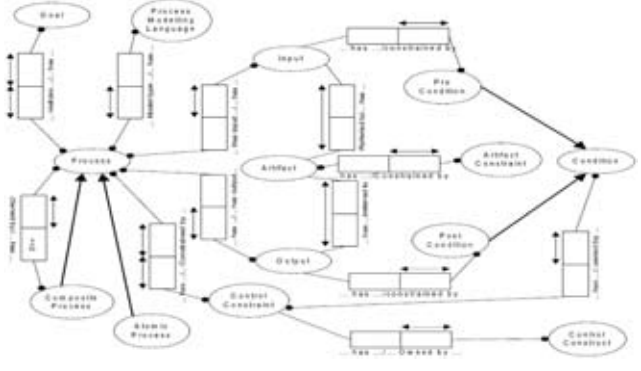


Figure 2. ORM model of the ISO/IEC 19763-5 standard [26]

Total function	Bi-directional total function
OwnedBy : ControlConstruct → ControlConstraint	Realizes : Process ↔ Goal
OutConstrained : PostCondition → Output	ConstrainedBy : Process ↔ ControlConstraint
InConstrained : PreCondition → Input	

Table 2. Graphical to textual conversion of uni-directional and bi-directional total functions

Specialization	Cartesian product
CompositeProcess(*Process*)	OutReferred ⊆ Output × Artifact
AtomicProcess(*Process*)	InReferred ⊆ Input × Artifact
PostCondition(*Condition*)	
PreCondition(*Condition*)	

Table 3. Graphical to textual conversion of specializations and Cartesian products

Extended total function
ProcOwnedBy : Process ⇝ CompositeProcess
ModelType : Process ⇝ ProcessModellingLanguage
HasInput : Input ⇝ Process
HasOutput : Output ⇝ Process
ArtifactConstrained : ArtifactConstraint ⇝ Artifact
CondOwnedBy : Condition ⇝ ControlConstraint

Table 4. Graphical to textual conversion of extended total functions

3.2 Conversion of textual to graphical formalisms

An example of a textual formal model that is suitable for conversion to a graphical variant can be found in [2]. The mentioned formalisms concern a formal model of an *information market*. An information market is defined as: "the market where resources are exchanged between searchers and publishers, possibly by means of brokers" [2]. The formal model includes five different sets. The set $\mathcal{AS}$ is the set of assets, which is defined as: "Any thing that can be exchanged in a transaction" [2]. The set $\mathcal{TA}$ is the set of *transactions*, which is defined as: "A specific, identifiable exchange of assets between two or more players where each participant in the transaction pays something and receives something in return" [2]. The set of players, $\mathcal{PL}$, refers to persons or organizations that may participate in a transaction. Subsequently, the *value domain* referred to as $\mathcal{VD}$ includes the values of assets to players. *Value* is defined as "the increment / decrement in the satisfaction levels of players when assets are exchanged in transactions" [2]. Finally, the set of *player states*, $\mathcal{ST}$, is defined as "the present satisfaction of a player with respect to his goals" [2].

A total of six equations have been introduced as part of the information market model. First, the *transactor* equation can be used to denote which assets are exchanged by a player. This equation is modeled as follows:

$$\text{-[-]-} \subseteq \mathcal{AS} \times \mathcal{PL} \times \mathcal{AS} \quad (1)$$

If an asset $a_1 \in \mathcal{AS}$ is exchanged for asset $a_2 \in \mathcal{AS}$ by player $p \in \mathcal{PL}$ this can be expressed as: $a_1[p]a_2$. Additionally, the *transactand* function can be used to denote which players exchange an asset. This is formalized as:

$$\text{-[-]-} \subseteq \mathcal{PL} \times \mathcal{AS} \times \mathcal{PL} \quad (2)$$

The expression $p_1[a]p_2$ shows that player p_1 exchanges asset $a \in \mathcal{AS}$ with player p_2. Subsequently, *buyers* and *sellers* of assets can be identified by means of the corresponding functions:

$$\text{Buyer, Seller} : \mathcal{TA} \rightarrow \mathcal{PL} \quad (3)$$

The expression Buyer(t) = p reflects that player p is a buyer of a transaction $t \in \mathcal{TA}$. Analogously, the expression Seller(t) = p shows that player p is the seller of that transaction. The value of an asset to a player can be expressed by applying the value function:

$$\text{Val} : \mathcal{PL} \times \mathcal{AS} \rightarrow \mathcal{VD} \quad (4)$$

The expression Val(p, a) = v shows that asset a is of a value $v \in \mathcal{VD}$ to player p. Finally, the actual identification of players by states can be done by using the identification equation:

$$\text{Id} : \mathcal{ST} \rightarrow \mathcal{PL} \quad (5)$$

Thus, a player p can be identified by state $s \in \mathcal{ST}$.

At this point the textual formalisms as part of the information market model have been identified. As a next step, it is possible to convert these formalisms to a graphical ORM model. However, when studying the textual formalisms it can be concluded that only three out of the six equations can be converted by using our abstract elementary patterns presented in table 1. These are the buyer, seller, and identification functions. The remaining transactor, transactand, and asset value equations require more complex patterns. To succeed in converting the textual model to a graphical one, we need to introduce two patterns of a more complex nature for this occasion. The pattern required to convert the transactor and transactand functions is actually an extension of the 'Cartesian product' pattern shown in table 1, which has been formalized as: $R \subseteq \mathcal{X} \times \mathcal{Y}$. The difference is that the complex pattern is a Cartesian product of three sets instead of two, which can be formalized as: $R \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$. Figure 3 shows the graphical variant of this pattern. The pattern required to convert the asset value equation is an extension of the arbitrary 'total function' pattern, which has been modeled as: $R : \mathcal{X} \rightarrow \mathcal{Y}$. The difference is that the domain of the more complex pattern consists of a Cartesian product instead of only one set. The more complex total function pattern can be formalized as: $R : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Its visual counterpart is shown in figure 4. This exercise shows that the elementary patterns can be easily extended for more complex conversion scenarios. At this point, the discussed textual formalisms as part of the information market model can be converted to a single graphical ORM model, which is shown in figure 5. Subsequently, the application of the abstract elementary patterns in the two scenarios of the ISO/IEC standard and the information market leads to several evaluative remarks.

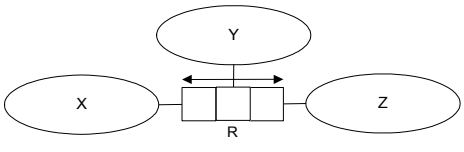


Figure 3. 'Complex Cartesian product' pattern

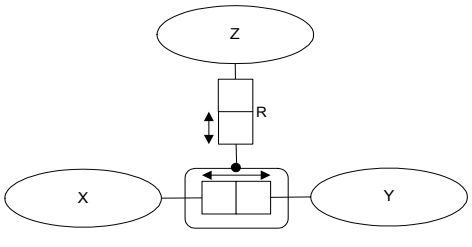


Figure 4. 'Complex total function' pattern

3.3 Evaluation

The textual and graphical variants of the ISO/IEC 19763-5 standard description and the information market model can now be compared to identify advantages and disadvantages of both approaches. The graphical models have as a striking advantage that they provide a complete overview of, in this case, the ISO/IEC 19763-5 standard and an information market. Next

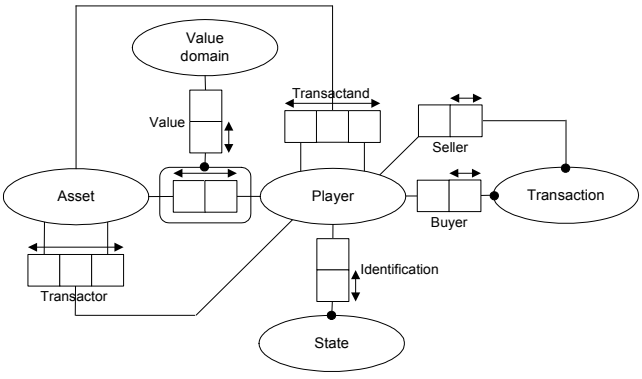


Figure 5. ORM variant of a textual formal model of an information market as shown in [2]

to that, the ORM models are readable for audiences that have a mathematical background and for audiences that have other (non-mathematical) backgrounds. Another advantage is that the names of the relations that members of an object type have with each other can be easily displayed below the fact type. This is impossible in a textual counterpart. In ORM, it is also easy to create a variety of elementary patterns by simply adding or removing a constraint. Removing or adding a constraint graphically may express something completely different that sometimes requires the introduction of a new mathematical symbol in a textual variant of the formalism, possibly making the textual variant more difficult to comprehend. This is because each operator that is used in a mathematical function has its own interpretation, while an ORM model consists of a limited number of different concepts that can be applied in multiple ways. For example, adding a uniqueness constraint on a fact type can be easily applied but immediately changes the semantics of the formal model.

A disadvantage of the graphical models is that they become cluttered if too many formalisms are visualized in one model, which may force the modeler to break a model up in parts. The textual models can be easily categorized in groups of functions that are used, which has already been shown earlier. Compared to the ORM model, the textual model may also be easier to understand by an audience consisting of mathematicians. Furthermore, a textual formalism can be verbalized in a compact way, while the graphical (ORM) variant consumes more space. In general, the readability of both the textual and visual formalisms decreases when modeling very complex formalisms. In that case, trying to decrease the level of complexity can increase overall readability. In summary, when comparing the graphical variant and the textual variant it can be noted that the graphical variant:

- shows a complete overview,
- becomes cluttered in case of too many formalisms,
- is accessible for different audiences,
- is less easier to understand by pure logicians,
- enables naming of relations between object types,
- consumes more space when compared to textual formalisms *and*
- enables to easily change or add semantics by means of constraints.

Furthermore, the conversion from the textual information market model to a graphical model required the introduction of two more patterns. These patterns were necessary to convert a triplet of more complex textual formalisms. This shows that the elementary patterns can serve as basic building blocks that can be easily extended to more complex patterns.

4. Discussion

Before comparing the approach to convert formalisms discussed in this paper with other approaches, an overview is provided in table 5. This overview shows those research contexts in which we have already informally applied this technique, i.e. without making explicit which conversion patterns are used to generate the models. Each application includes textual (set-theoretical) formalisms and visual formalisms by means of at least one ORM model. In retrospect, the patterns have also surfaced during the development of the models mentioned in table 5. Each formal model mentioned in the table has been visualized as one ORM model, while the textual formalisms are less cohesive and often require more explanation in natural language. The latter can be assigned to the limited number of constructs that constitute an ORM model. The model can be interpreted once these constructs are understood. Textual formalisms require more explanation when new constructs such as new operators are introduced. We experienced that by creating the ORM visuals of the formal models mentioned in table 5 deficiencies in the textual formalisms came to light. This can be assigned to modeling activities such as the appliance of constraints, the creation of fact types and by studying the overall ORM model. When a deficiency in a formalism is discovered, the formalism is corrected leading to a modification of the ORM model and the textual variant.

Model	Description
Ontology for managing cross-organizational cooperation	A formal ontology to share information for managing dependencies among cooperating organizations has been presented in [24]. In this ontology, organizations can register, modify, and remove information that is relevant for cooperating with other organizations.
Knowledge exchange model	A formal model showing and relating foundations of knowledge exchange is presented in [20].
Cognitive actor settings	An actor is an entity that is able to perform a task [19]. The cognition of actors can differ, because not every actor has the same cognitive characteristics. A cognitive actor setting is a specific set of cognitive characteristics belonging to an actor. A formal model of possible cognitive actor settings has been provided in [19].
Cognitive matchmaking framework	A formal framework to match supply and demand of cognitive characteristics [19].
Fuzzy cognitive matching	A formal model to attach linguistic values to cognitive matching [19].
Properties of knowledge intensive tasks	Knowledge intensive tasks have been defined as tasks for which the acquisition, application, or testing of knowledge is needed to fulfill it [21]. The properties of those tasks have been formalized in [21].
Knowledge market model	Formalization of elements that constitute a market in which knowledge is supplied and acquired [22].

Table 5. Applications of formalism conversion technique

Other works related to conversion of textual and visual formalisms include that of Harel [7]. Harel shows how formalisms can be visualized by using *hi-graphs*. Higraphs are diagrams that provide a powerful and concise way of visualizing set-theoretical formalisms, extended with the ability to visualize the Cartesian product of sets and the relationships between sets. A consequence of introducing an additional diagramming technique is that its interpretation must be learned by the reader before it can be used in practice. In contrast, an audience that is already capable of understanding ORM is immediately able to understand visualizations of set-theoretical formalisms. Like ORM, the hi-graph models are also founded in a formal syntax and semantics. An advantage of higraph models is that they are mainly aimed at visualizing set-theoretical formalisms, which make the resulting diagrams very clear and concise. This makes it a less suitable diagramming technique for other types of formalisms in contrast to ORM. Another difference with higraphs is that ORM models can be extended with a variety of constraints.

The advantages of both visual formalisms and textual formalisms have been combined in [3]. Visual formalisms are used to create specifications of reactive systems combined with formal verification and program transformation tools developed for textual formalisms. A tool is presented that automatically produces statechart layouts based on information extracted from an informal specification. Statecharts are extended finite state machines used to describe control aspects of reactive systems. The tool presented in [3] is also capable of translating the statecharts to specifications in the Z language. Z is a formal notation based on set theory and predicate logic. Due to these differentiations the tool cannot be utilized for an audience that is more interested in the relations between textual and visual formalisms in a broader way. As a consequence, the tool can be used if textual formalisms in Z and visual statechart formalisms are present. The tool is less suitable when textual formalisms need to be developed in set theory without specifically focusing on reactive systems. In that case, the more generally applicable conversion patterns described in section 2 are more suitable to apply. On the other hand, our approach may be less suitable for the visualization of formal models based on a more *specialized* formal language.

The creation of visual UML views from formal B specifications has been pre sented in [13]. B is a formal notation related to the aforementioned Z notation and supports development of programming language code from specifications. B leads to a better precision than UML while UML produced more intuitive and readable documents [13]. The difference in this approach compared to ours is that an informal visual language (UML) is supported by formal reasoning in B. Because the UML constructs are not based on formal foundations they are not suitable for direct conversion to textual formal counterparts. Thus, patterns to convert B specifications to UML view diagrams are presented in [13], but conversion patterns from UML views to B specifications are not offered. Finally, graphical representations for knowledge structures in the design method for component-based design of knowledge-based and multi-agent systems are presented in [17]. Moreover, a translator has been described which translates these graphical representations to textual representations in DESIRE. It is mentioned that, originally, a textual knowledge representation language was used in DESIRE that is based on order sorted predicate logic and some extensions. Moreover, a translator is described which translates these graphical representations to textual representations in DESIRE. However, a translator from textual formalisms to graphical models is not covered. A

mapping from DESIRE to conceptual graphs is also mentioned. Conceptual graphs visually represent logic. This mapping has led to a more unifying perspective on the DESIRE knowledge representation formalism, and has opened the possibility to use DESIRE in connection to the knowledge representation tools of the conceptual graph community. This can be compared to the usage of ORM in our conversion patterns, because the ORM community can understand and benefit from the described conversion patterns.

In this paper, we have focussed on ORM as a formalism for the graphical representation of sets and relations between sets. Obviously, this is one of the possible ways to graphically represent set-theoretical formalisms. Support for graphical representations of sets has been discussed earlier in [25]. Pakkan and Akman use a graphical representation for an alternative set theory and thereby also allow the representation of non-well-founded sets. Cyclic sets, i.e., sets which can be members of themselves, are examples of non-well-founded sets that can be represented by this alternative set theory. Non-well-founded sets have infinite descending membership sequences and have generally been neglected by the practicing mathematician since the classical well-founded universe was a satisfying domain for his practical concerns. However, non-well-founded sets are useful in modeling various phenomena in computer science, viz. concurrency, databases and artificial intelligence [1]. Thus, Pakkan and Akman argue that these kind of sets are also useful to take into account when specifying a formal model. A prototypical software application is shown in [25] which can solve systems of equations defined in terms of sets in the universe of that alternative set theory which is useful for commonsense reasoning.

5. Conclusions

In this paper, elementary conversion patterns have been presented to convert textual formalisms to visual formalisms and vice versa. The Object-Role Modeling (ORM) language has been used to develop the visual patterns and set-theoretical notations have been used for the textual patterns. Moreover, these patterns have been materialized and evaluated in two applications. First, an ORM model of the ISO/IEC 19763-5 standard is converted to a textual formal model [26]. This standard provides a meta model to register administrative information and meaningful semantics of process models. Second, the patterns are used to convert a textual formal model of an information market to an ORM variant. An information market is a market where resources are exchanged between searchers and publishers, possibly by means of brokers [2]. The materialization of the conversion patterns showed that these elementary patterns can serve as basic building blocks that can be easily extended to more complex patterns.

The visual patterns are useful to provide a complete overview of related formalisms. Next to that, ORM models are readable for audiences that have a mathematical background and for audiences that have other backgrounds. Another advantage of a graphical model is that names can be assigned to the relations that formalisms have with each other. This is impossible in a textual counterpart. The visual ORM patterns also enable the possibility to create a variety of elementary patterns by simply adding or removing a constraint. Removing or adding a constraint graphically may express something completely different that sometimes requires the introduction of a new mathematical symbol in a textual variant of the formalism. This can make the textual variant more difficult to comprehend, because each operator that is used in a mathematical function has its own interpretation. On the contrary, an ORM model consists of a limited number of different concepts that can be applied in multiple ways. For example, adding a total role constraint to an object type can be easily applied but immediately changes the semantics of the formal model.

A disadvantage of the graphical models is that they become cluttered if too many formalisms are visualized in one model. When this happens, a modeler is forced to break up a model in parts. If there are a lot of textual formalisms in a formal model, they can be easily categorized in groups of functions. Compared to the visuals, the textual variants may be easier to understand by an audience purely consisting of mathematicians. Furthermore, a textual formalism can be verbalized in a compact way, while the graphical (ORM) variant consumes more drawing space. In general, the readability of both the textual and visual formalisms decreases when very complex formalisms are modeled. In that case, decreasing the level of complexity can increase overall readability.

6. Future research

For the future, the purpose is to pursue a triad of research directions. One of them is the development of more complex conversion patterns. Two complex conversion patterns have already been introduced in section 3.2. Each of the elementary patterns discussed in this paper are at the basis of a possible complex pattern. For example, an elementary visual pattern can already be extended to a complex one by adding one more object type and a fact type that relates that object type with an object type that exists in the elementary pattern. Eventually, the elementary and complex patterns can lead to modularization of ORM schemes, i.e. patterns can be combined to form on-demand ORM schemes. The main goals of modularity are to enable and increase reusability, maintainability, and distributed development of ORM schemes [14].

The second research direction consists of discovering textual and visual representations of constraints. Two types of constraints can be identified in the elementary patterns: the total role constraint and the uniqueness constraint. At least two main reasons underlie the need to introduce such constraints. A first reason is that during domain analysis (in this case the analysis of certain formalisms) certain constraints may be necessary. A next step is then to determine how these constraints, which arise from the formalisms under analysis, can be modeled in ORM. In this situation the analysis of the formal theory delivers relevant feed back for an ORM schema. In a second case, it might happen that every constraint has already been textually formalized (in an underlying assumably completely formalized theory). Such formal constraints can then easily be visualized in an ORM model according to their corresponding semantics. However, the acquisition and specification of constraints is far from being trivial [8]. This task not only demands high abilities to abstract from the ORM schema, but also tends to be rather complex. Therefore, pre-defined textual and visual representations of constraints supports modelers to identify and specify them.

The third and final research direction is related to the development of a computer-based tool to realize automated conversion of textual and visual patterns. Eventually, an advanced tool can also be utilized to enable automated discovery of complex patterns by using elementary patterns as their foundation. An ORM-based tool that is aimed to guide ontology builders towards building ontologies that are easier to build and to maintain is introduced in [16]. An ontology, which is an agreed understanding of a certain domain [16], is formally represented as logical theory in the form of a computer-based resource. An ORM

model of such an ontology provides a visualization of a formal ontology. Just like the ORM-based tool for ontology modeling, the realization of a tool related to our research can also ease development and maintenance of conversion patterns.

References

[1] Barwise, J., Etchemendy. J. (1987). *The Liar: An Essay on Truth and Circularity*. Oxford University Press, New York, NY, USA.

[2] van Bommel, P., van Gils, B., Proper, H.A.,van Vliet, M., van der Weide, Th.P. (2007). Value and the information market. *Data & Knowledge Engineering*, 61(1)153–175.

[3] Castello, R., Mili, R. (2003). Visualizing graphical and textual formalisms. *Information Systems*, 28 (7) 753–768.

[4] Clarke, E.M.,Wing, J.M. (1996). Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4) 626–643.

[5] Falkenberg, E.D. (1976). Concepts for modelling information. In G.M. Nijssen, editor, *Proceedings of the IFIP Working Conference on Modelling in Data Base Management Systems*, pages 95–109. Freudenstadt, Germany, EU, North-Holland Publishing, Amsterdam, The Netherlands, 1976.

[6] Halpin, T. (2001). *Information Modeling and Relational Databases, from Conceptual Analysis to Logical Design*. Morgan Kaufmann, San Mateo, CA, USA.

[7] Harel, D (1998). On visual formalisms. *Communications of the ACM*, 31(5) 514– 530.

[8] Hartmann, S., Link, S., Trinh, T. (2009). Constraint acquisition for Entity– Relationship models. *Data & Knowledge Engineering*, 68(10) 1128–1155.

[9] ter Hofstede, A.H.M., Proper, H.A. (1998). How to formalize it? *Information and Software Technology*, 40 (10) 519–540.

[10] ter Hofstede, A.H.M., Proper, H.A., van der Weide, Th.P. (1993). Formal defi nition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7) 489–523.

[11] ter Hofstede, A.H.M., van der Weide, Th.P.(1993). Expressiveness in conceptual data modeling, *Data & Knowledge Engineering*, 10(1) 65–100.

[12] Hull, R., R. King, R. (1987). Semantic database modeling: Survey, applications, and research issues, *ACM Computing Surveys*, 19 (3) 201–260.

[13] Idani, A., Ledru, Y (2006). Dynamic graphical UML views from formal B specifications. *Information and Software Technology*, 48 (3) 154–169.

[14] Jarrar, M. (2005). Modularization and automatic composition of Object-Role Modeling (ORM) schemes. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, Agia Napa, Cyprus, October 31 -November 4, 2005, Proceedings, volume 3752 of Lecture Notes in Computer Science*, pages 613–625. Agia Napa, Cyprus, Springer, Berlin, Germany.

[15] Jarrar, M. (2007). Mapping ORM into the SHOIN/OWL description logic – towards a methodological and expressive graphical notation for ontology. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Vilamoura, Portugal, November 25 -30, 2007, Proceedings, Part I*, volume 4805 of Lecture Notes in Computer Science, pages 729–741. Vilamoura, Portugal, EU, Springer, Berlin, Germany, 2007.

[16] Jarrar, M., Meersman, R. (2008). Ontology engineering – the Dogma approach. In T. Dillon, E. Chang, R. Meersman, and K. Sycara, editors, *Advances in Web Semantics I*, p. 7–34. Springer, Berlin, 2008.

[17] Jonker, C.M., Kremer, R., van Leeuwen, P., Pan, D.,Treur, J. (2005). Mapping visual to textual knowledge representation. *Knowledge-Based Systems*, 18 (7) 367–378.

[18] Nosek, J.T., Roth, I. (1990). A comparison of formal knowledge representation schemes as communication tools: predicate logic vs. semantic network. *International Journal of Man-Machine Studies*, 33(2):227–239, 1990.

[19] Overbeek, S.J., van Bommel, P.,Proper, H.A. (2008). Matching cognitive characteristics of actors and tasks in information systems engineering. *Knowledge-Based Systems*, 21(8) 764–785.

[20] Overbeek, S.J., van Bommel, P., Proper, H.A. (2009). Embedding knowledge exchange and cognitive matchmaking in a dichotomy of markets. *Expert Systems with Applications*, 36 (10) 12236–12255.

[21] Overbeek, S.J., van Bommel, P., Proper, H.A., Rijsenbrij, D.B.B.(2007). Characterizing knowledge intensive tasks indicating cognitive requirements – Scenarios in methods for specific tasks. In: J. Ralyt´e, S. Brinkkemper, and B. Henderson-Sellers, editors, *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences*, volume 244, p. 100–114. Geneva, Switzerland, Springer, Boston, USA.

[22] Overbeek, S.J., van Bommel, P., Proper, H.A., Rijsenbrij, D.B.B.(2007). Knowledge discovery and exchange – Towards a web-based application for discovery and exchange of revealed knowledge. In: J. Filipe, J. Cordeiro, B. Encarna¸c˜ao, and V. Pedrosa, editors, *Proceedings of the Third International Conference on Web Information Systems and Technologies (WEBIST)*, pages 26–34. Barcelona, Spain, EU, INSTICC Press, Set´ubal, Portugal, EU, 2007.

[23] Overbeek, S.J., van Bommel, P., Proper, H.A., Rijsenbrij, D.B.B (2007). Visualizing formalisms with ORM models. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Vilamoura, Portugal, November 25 -30, 2007, Proceedings, Part I*, volume 4805 of Lecture Notes in Computer Science, p. 709–718. Vilamoura, Portugal, Springer, Berlin, Germany.

[24] Overbeek, S.J., Klievink, A.J., Janssen, M.F.W.H.A (2009). A flexible, event-driven, service-oriented architecture for orchestrating service delivery. *IEEE Intelligent Systems*, 24 (5) 31–41.

[25] Pakkan, M., Akman, V. (1995). Hypersolver: A graphical tool for commonsense set theory. *Information Sciences*, 85 (1–3) 43–61.

[26] Piprani, B., Wang, C., He, K. (2008). A metamodel for enabling a service oriented architecture. In: R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2008: OTM 2008 Workshops, Monterrey, Mexico, November 9 -14, 2008, Proceedings, volume 5333 of Lecture Notes in Computer Science*, p. 668–677. Monterrey, Mexico, Springer, Berlin, Germany.

[27] Verhoef, T. (1993). *Effective Information Modelling Support*. PhD thesis, Delft University of Technology, The Netherlands.