Visualizing Formalisms with ORM Models

S.J. Overbeek[1], P. van Bommel[2], H.A. (Erik) Proper[2], and D.B.B. Rijsenbrij[2]

[1] e-office B.V., Duwboot 20, 3991 CD Houten, The Netherlands, EU
`Sietse.Overbeek@e-office.com`
[2] Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
`{P.vanBommel,E.Proper,D.Rijsenbrij}@cs.ru.nl`

Abstract. During the development of theoretical frameworks researchers often graphically represent formal textual notations as part of a developed theory. This may lead to enrichments and new insights regarding a theory. A possibility for graphical representation of formalisms is the utilization of modeling languages such as ORM. This paper deals with the technique of visualizing formalisms by using ORM models and shows the advantages of graphically representing a formal theoretical framework. An application of the approach that has already been successfully practised is elaborated. This application concerns a theoretical framework consisting of knowledge intensive task properties and shows how the approach to visualize formalisms with ORM can be materialized.

1 Introduction

When developing a formal theory, 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. Textual formalisms, however, may be complex in nature and not acceptable to many stakeholders. Visual formalisms offer graphical notations with semantics and also offer the possibility to model intuitive and well-organized formalisms. Unfortunately, hand made diagrams become easily unreadable when the complexity of the formalisms increases. Conceptual modeling languages incorporating a (semi-)formal modeling language are, therefore, more suitable to visualize formal theory. 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) language is useful to visualize formalisms because of its formal foundations [1], its demonstrable applications in visualizing formalisms [2] and its long running affiliation with the field of conceptual modeling involving varied, often non-technical stakeholders [3]. In the Ph.D. thesis of Verhoef [4], for example, ORM is applied to visualize formalisms of a theory about the use of modeling knowledge to achieve more effective information modeling support.

This paper deals with the technique of visualizing formalisms by using ORM models and shows advantages for graphically representing a formal theoretical framework. Situations in which it is less obvious to visualize formalisms are also discussed. It is assumed that the reader is familiar with ORM. Section 2 first introduces the basics of visualizing formalisms with ORM. An application of the approach that has already been successfully practised is explained in section 3. Furthermore, other known approaches to visualize formalisms are discussed in section 4 and are compared with our approach. Eventually, the paper is concluded in section 5.

2 Visualization of Formalisms with ORM Models

In this section is elaborated how formalisms and constraints can be visualized by means of ORM. Several advantages and disadvantages are considered to indicate for which situations it is more respectively less obvious to visualize a formal model with ORM models.

2.1 Visualization of Basic and Complex Formalisms

Throughout this section our ideas are explained how to graphically represent formal notations as part of a theoretical framework. Assume that a theoretical framework consists of the functions 1 up to and including 5 as explained below. First, suppose that function 1 can be depicted as follows:

$$\mathsf{F} : \mathcal{X} \rightarrow \mathcal{Y} \tag{1}$$

The expression $\mathsf{F}(x) = y$ states that for an element x as part of the set $\mathcal{X}$ function 1 returns an element y from the set $\mathcal{Y}$. The sets as part of a function are visualized as object types in ORM. If dictated by the nature of the mathematical function, constraints such as total role constraints or uniqueness constraints should be added to a possible ORM model. As such, a uniqueness constraint should be 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 should also be added to object type $\mathcal{X}$, because function 1 prescribes that every instance of object type $\mathcal{X}$ should play a role in the fact type. A bit more complex function such as the one depicted below may also be part of a formal model:

$$\mathsf{G} : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z} \tag{2}$$

The expression $\mathsf{G}(w, x, y) = z$ shows that for $w \in \mathcal{W}$, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ function 2 returns an element $z \in \mathcal{Z}$. Assume that the set $\mathcal{W}$ contains constants, so that $\mathcal{W} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. The set $\mathcal{W}$ can now be visualized as a value type in the ORM model (indicated by parentheses), together with its corresponding values. Suppose that functions 1 and 2 are part of one and the same formal model. These two separate functions can now visually integrate in one ORM model as is shown in figure 1. Normally, when such functions as part of formal models are depicted one by one in the text and then textually explained, such an overview cannot be given. In that case, there is a chance that the reader of a formal model misses the interconnection between the functions of a formal model. This might

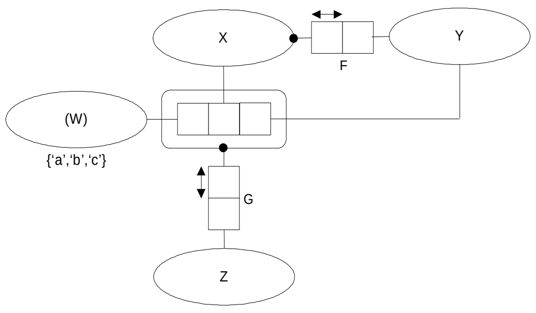

Fig. 1. ORM model of two functions.

make a formal model difficult to understand. To be certain that function 2 is correctly visualized, the introduction of an objectified fact type is necessary as can be seen in figure 1.

It is interesting to expand the ORM model with two more functions that incorporate additional mathematical symbols. The following function takes an element of the set $\mathcal{Z}$ as parameter and returns a (real) value from 0 up to and including 1:

$$\mathsf{H} : \mathcal{Z} \to [0, 1] \tag{3}$$

Thus, the expression $\mathsf{H}(z) = 0.5$ shows that for $z \in \mathcal{Z}$ the value 0.5 is returned. The following function returns an element from the powerset of the set $\mathcal{Z}$:

$$\mathsf{I} : \mathcal{W} \to \wp(\mathcal{Z}) \tag{4}$$

The expression $\mathsf{I}(w) = Z$ shows that for $w \in \mathcal{W}$ the set Z is returned, where $Z \subseteq \mathcal{Z}$. The powerset of the set $\mathcal{Z}$ should be visualized as a *power type* in the ORM model, as introduced by [5]. An instance of a power type is identified by its elements, just as a set is identified by its elements (axiom of extensionality) [6]. Normally, a set denoted in a function is displayed as an object type which equals the name of the set. Regarding function 3, however, the number range $[0, 1]$ is unnamed. There is a possibility to visualize this by introducing a value type named 'number', which includes values ranging from 0 up to and including 1.

Before visualizing the functions discussed so far, the following more complex function is tackled:

$$\mathsf{J} : \mathcal{V} \to (\mathcal{X} \to \mathcal{Y}) \tag{5}$$

An example expression of this function can be denoted as $\mathsf{J}_v(x) = y$, where $v \in \mathcal{V}$, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Note that function 5 is not the same as $\mathsf{J} : (\mathcal{V} \times \mathcal{X}) \to \mathcal{Y}$ because of the placed parentheses. Instead, it can be equated to $\mathsf{J} : \mathcal{V} \to \wp(\mathcal{X} \times \mathcal{Y})$. Figure 2 shows a complete overview of the functions defined up till now.

As can be seen in figure 2, an exclusion constraint has been added. Up till now, the visualization of complex constraints in an ORM model has not been discussed. The following section deals with this matter.

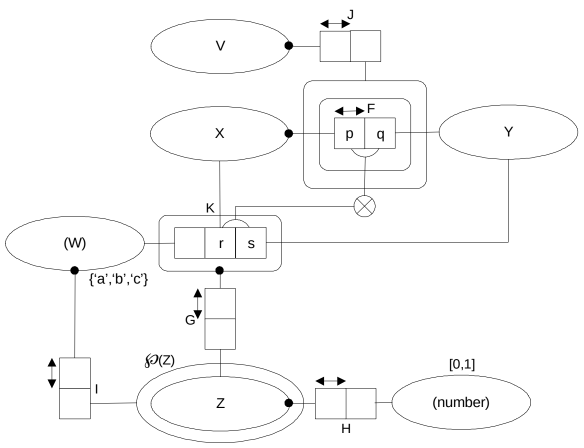

Fig. 2. ORM model of an example formal model.

2.2 Visualization of Complex Constraints

At least two main reasons underlie the need to visualize more complex 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 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. For instance, the semantics of the exclusion constraint of figure 2 must then be part of the formal theory: $\pi_{r,s}(\mathsf{K}) \otimes \pi_{p,q}(\mathsf{F})$. The exclusion constraint expresses that there is no overlap between X / Y combinations in the fact types F and K. The visualization of already formalized constraints results in a specific quality check, because when modeling such a constraint in ORM it is possible to determine the correctness of the constraint in the underlying theory. In this situation the analysis of an ORM schema delivers relevant feed back for the formal theory.

As can be seen, ORM is a powerful tool to stipulate relations between functions and to find out if the formalisms are indeed correctly defined. We assume that flaws in a formal model can more easily be found by visualizing and integrating formalisms this way. After all, the meaning of a function in a formal model as a whole is more understandable.

In the following section, we further explore possible advantages when using ORM models to visualize formalisms as well as situations in which it is less advantageous to use this approach.

2.3 Advantages and Disadvantages

The following list sums up possible advantages of the approach mentioned so far:

- Formalisms such as those discussed in section 2 each require a textual explanation for the reader to be able to interpret them. When these formalisms are visualized, however, additional text is not required provided that the reader understands ORM.
- Interrelationships between functions are visible in an ORM model by means of fact types and roles. The textual formal model of section 2 does not provide insight in these interrelationships.
- In an ORM model it is immediately obvious which instances of an object type play the most roles in fact types compared to other object types. This may be useful to identify the most important concept of the formal model as a whole.
- When creating an ORM model deficiencies in the formalisms may come to light. This is caused by activities such as the appliance of constraints, the modeling 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.
- Hofstede and Proper [2] have formalized ORM in set theoretic, logical based and category theoretic variants. This makes ORM a modeling language with a well-defined syntax and semantics. It is natural to use ORM to visualize textual formalisms of the aforementioned variants.
- ORM has a long running affiliation with domain modeling involving varied, often non-technical domain experts. This implies that stakeholders able to interpret ORM have different (and not only technical) backgrounds. This makes the eventual visualizations of formalisms more comprehensible for a broader audience than only technical or mathematical domain experts [7].
- Finally, ORM is a richer modeling language, meaning that it is suited to the visualization of more complex formalisms. This is in contrast with e.g. the Entity Relationship (ER) family of data modeling techniques [2].

There are, of course, situations in which our approach is a less obvious choice. Such possible situations can be explained as follows:

- When the number of sets and functions in a formal model are low, the added value of ORM as a visualization tool is considered negligible.
- The functions of section 2 show which sets are involved in the specific functions. It is probably not obvious to use ORM when the actual semantics of a function leads to difficult processes to actually populate a typical fact type with instances. In other words, ORM can be perfectly used to visualize sets together with their instances which play a role in certain fact types. The process to create set instances is not always straightforward. Such processes leading up to the creation of instances can not be shown easily with ORM, though.

In the following section we explain how we have materialized the approach discussed so far in the development of a formal model concerning *properties* of *knowledge intensive tasks* as elaborated in [8].

3 Application: ORM Model of Knowledge Intensive Task Properties

In [8], a formal model has been developed that consists of a characterization of knowledge intensive tasks based on task properties. We understand a knowledge intensive task to be a task for which acquisition, application or testing of already applied knowledge is necessary in order to successfully fulfill the task. Therefore, a trichotomy of three different task *types* has been elaborated: an *acquisition* task type, a *synthesis* task type and a *testing* task type. Each knowledge intensive task type is characterized (and can be distinguished) by their specific properties.

The knowledge intensive task properties are described in a formal way by using set theory. Table 1 shows the functions included in the present formal model, together with a short explanation of each function. The formal model of table 1 is visualized in figure 3. Strikingly enough, the ORM model of figure 3 shows

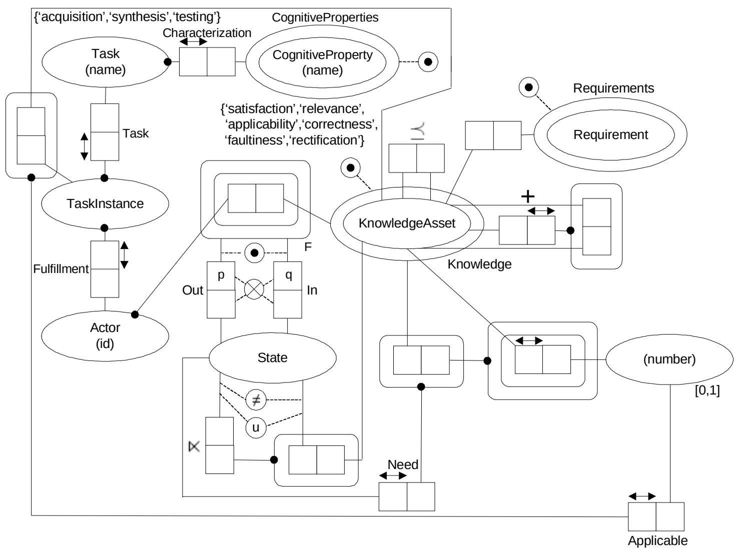

Fig. 3. ORM model of knowledge intensive task properties.

that elements of the *knowledge assets* object type play the most roles in fact types compared to the other object types. It can be concluded that the knowledge asset is a very important concept of the formal model as a whole. These *assets* are tradeable forms of knowledge, i.e. knowledge that is exchangeable between actors. This may include knowledge obtained by viewing a Web site or a

Table 1. Formal model of knowledge intensive task properties [8]

Function	Explanation
$\mathsf{Task} : \mathcal{TI} \to \mathcal{TA}$	$\mathsf{Task}(i) = j$ expresses that task instance i is of the type j.
$\mathsf{Fulfillment} : \mathcal{TI} \to \mathcal{AC}$	$\mathsf{Fulfillment}(i) = a$ expresses that task instance i is fulfilled by actor a.
$\mathsf{Characterization} : \mathcal{TA} \to \wp(\mathcal{CP})$	$\mathsf{Characterization}(j) = C$ expresses that task type j is characterized by the cognitive properties C.
$\mathsf{Need} : \mathcal{AS} \to (\wp(\mathcal{KA}) \times \mathcal{KA} \to [0,1])$	$\mathsf{Need}_t(\mathcal{S}, k)$ is interpreted as the residual need for knowledge asset k (sometimes also called a knowledge item) of an actor in state t after the set $\mathcal{S}$ has been presented to an actor. Here, $t \in \mathcal{AS}$, $k \in \mathcal{KA}$ and $\mathcal{S} \subseteq \mathcal{KA}$. The set $\mathcal{S}$ can be interpreted as the personal knowledge of an actor (also called a knowledge profile). $\mathsf{Need}_t(\mathcal{S}, k) = 0$ means that an actor has no need for knowledge asset k while in state t. The expression $\mathsf{Need}_t(\mathcal{S}, k) = 1$ shows that an actor has maximum need for knowledge asset k while in state t.
$\ltimes : \mathcal{AS} \times \mathcal{KA} \to \mathcal{AS}$	When actor a in state t experiences knowledge asset k, then this actor will end up in a new state $t \ltimes k$.
$\mathsf{In}, \mathsf{Out} : \mathcal{AS} \to (\mathcal{AC} \to \wp(\mathcal{KA}))$	If the input of actor a in state t comprises knowledge assets K, then this is shown by $\mathsf{In}_t(a) = K$. Ditto when output of knowledge is concerned.
$\mathsf{Applicable} : \mathcal{TI} \times \mathcal{KA} \to [0,1]$	The applicability of knowledge asset k for fulfilling task instance i is expressed by $\mathsf{Applicable}(i, k)$. Here, $\mathsf{Applicable}(i, k) = 0$ indicates that knowledge asset k is not applicable for task instance i. $\mathsf{Applicable}(i, k) = 1$ indicates that knowledge asset k is fully applicable for task instance i.
$\mathsf{Requirement} \subseteq \mathcal{KA} \times \wp(\mathcal{RQ})$	If knowledge asset k meets requirements R while applied in task fulfillment, this is indicated as $(k, R) \in \mathsf{Requirement}$.
$\preceq \subseteq \mathcal{KA} \times \mathcal{KA}$	The notation $k_1 \preceq k_2$ is verbalized as *the knowledge in* k_1 *is contained within* k_2.
$+ : \mathcal{KA} \times \mathcal{KA} \to \mathcal{KA}$	The concatenation of e.g. knowledge assets k_2 and k_3 can be shown as $k_2 + k_3$.

document or by conversing with a colleague. When an instructor explains to a learner how to drive a car for instance, the explanation may contain valuable knowledge assets for the learner.

Besides relatively basic set-theoretical functions, table 1 includes two more comprehensive formalisms. To be able to interpret the visualization of the function $\mathsf{In}, \mathsf{Out} : \mathcal{AS} \to (\mathcal{AC} \to \wp(\mathcal{KA}))$ in figure 3, it should be noted that this function can be equated to $\mathsf{In}, \mathsf{Out} : \mathcal{AS} \to \wp(\mathcal{AC} \times \wp(\mathcal{KA}))$. The most comprehensive function to visualize, however, is the need function:

$$\mathsf{Need} : \mathcal{AS} \to (\wp(\mathcal{KA}) \times \mathcal{KA} \to [0,1])$$

To correctly visualize this function, it should be boiled down so that insight is provided in the possible Cartesian products as part of the function. The need function can eventually be represented as:

$$\mathsf{Need} : \mathcal{AS} \to (\wp(\mathcal{KA}) \times \wp(\mathcal{KA} \times [0,1]))$$

A few complex constraints that have been added to the ORM model have not been explained so far. First, a uniqueness constraint and an inequality constraint over two roles of the ‘state change’ function denoted by the $\ltimes$ symbol can be identified. The following population is excluded to demonstrate the effect of the uniqueness constraint:

$$\mathsf{Pop}(\ltimes) = \left\{ \begin{array}{l} \{\{t_1, k_1\}, t_2\} \ , \\ \{\{t_1, k_2\}, t_2\} \end{array} \right\}$$

Note that $t_1, t_2 \in \mathcal{AS}$ and $k_1, k_2 \in \mathcal{KA}$. A population Pop is the formal assignment of a finite set of instances to an object type. In this case, the uniqueness constraint forces that the combination t_1 and t_2 does not occur more than once. The following population is excluded to demonstrate the effect of the inequality constraint:

$$\mathsf{Pop}(\ltimes) = \left\{ \begin{array}{l} \{\{t_3, k_1\}, t_3\} \ , \\ \{\{t_3, k_2\}, t_3\} \end{array} \right\}$$

Here, $t_3 \in \mathcal{AS}$. The combination of these constraints on the same roles enable that a state change from one state to another state cannot occur twice and a state cannot remain the same state (after a state change).

The exclusion constraint on the input and output fact types together with the total role constraint are also peculiar constraints. For instance, the following combination of populations is excluded due to the exclusion constraint:

$$\mathsf{Pop}(\mathsf{In}) = \left\{ \{\{\{a, K\}\}, t\} \right\} \quad \text{and} \quad \mathsf{Pop}(\mathsf{Out}) = \left\{ \{\{\{a, K\}\}, t\} \right\}$$

Here, $t \in \mathcal{AS}$, $a \in \mathcal{AC}$ and $K \subseteq \mathcal{KA}$. This constraint forces that, while in a certain state t, actor a cannot receive as well as broadcast some set of knowledge items K at the same time. The total role constraint on the input and output fact types forces that all the instances of object type F combined should be involved in role p, role q or in both roles. This implies that there must be instances that play role p and / or q if object type F also contains instances. Otherwise, the population of the fact types related to the input and output functions is incomplete.

Now that a materialization of the ideas to visualize formalisms by means of ORM has been elaborated, we would like to discuss how our approach relates to others. Therefore, an analysis and comparison of other relevant work is made in the following section.

4 Other Approaches

The work of Harel [9] shows how formalisms can be visualized by using *higraphs*. 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. An immediate consequence of introducing such 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 our visualizations of set-theoretical formalisms instead. Like ORM, the higraph models also have a formal (non-graphical) 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. However, this makes it a less suitable diagramming technique for other types of formalisms. Besides utilizing it for visualizing formalisms, ORM is a conceptual modeling language suitable for a broad range of applications. Amongst these are the ability to represent conceptual database schemas, the visualization of business rules and even the visualization of enterprise architectures [7]. As is explained in section 2.2, ORM is equipped with the ability to extend the resulting model with an extensive variety of constraints (also based on formal foundations). This is something which higraphs lack.

Sometimes the advantages of both visual formalisms and textual formalisms are clearly combined in one effort [10]. Here, 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 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, one can utilize the tool if textual formalisms in Z and visual statechart formalisms are present. When one would like to visualize textual formalisms as part of theoretical frameworks based on set theory in general (and when one is not specifically focussed on the development of reactive systems) the tool is less suitable. When the latter is the case, the described method of section 2 is more suitable as a way to relate textual with visual formalisms. On the other hand, our approach may be less suitable for the visualization of formal frameworks based on a more specialized formal language.

5 Conclusions & Future Work

This paper describes how set-theoretical formalisms can be visualized by means of ORM. Several examples of possible textual formalisms are discussed and then visualized forming an overall visual model. An application of the approach that has already been successfully practised in earlier work is elaborated. This application concerns a theoretical framework consisting of knowledge intensive task properties and shows how the approach to visualize formalisms with ORM can be materialized. Finally, other approaches to visualize formalisms are discussed and compared to our approach.

An obvious possibility for future research is the application of ORM for visualization of e.g. formal architecture principles as part of enterprise architectures [7]. The idea here is that formalization by ORM, when properly and systematically performed, may also lead to better analysis of certain patterns of meaning underlying enterprise architecture principles. This should lead to improvement of the (formulation of) architecture principles as such (even of their informal formulations).

Another future research path is that of using ORM to reason about domains. ORM can then be used to model the ontology of domains in general [11]. This leads to ORM models capturing the concepts of a domain, as well as an associated language to express rules (such as business rules) governing the behavior of the domain. When combined with a formal reasoning mechanism, this rule language becomes a domain calculus. In the case of ORM, such a domain calculus has been presented in the form of Object-Role Calculus (ORC) [7]. Remarkably, it is becoming more and more obvious that ORM, as a modeling language traditionally developed with the aim of providing conceptual models of database structures, can be utilized in many more ways related to many different purposes.

References

1. Halpin, T.: Information Modeling and Relational Databases, from Conceptual Analysis to Logical Design. Morgan Kaufmann, San Mateo, CA, USA (2001)
2. Hofstede, A., Proper, H.: How to formalize it? Information and Software Technology **40**(10) (1998) 519–540
3. Falkenberg, E.: Concepts for modelling information. In Nijssen, G., ed.: Proceedings of the IFIP Working Conference on Modelling in Data Base Management Systems, Freudenstadt, Germany, EU, North-Holland Publishing, Amsterdam, The Netherlands, EU (1976) 95–109
4. Verhoef, T.: Effective Information Modelling Support. PhD thesis, Delft University of Technology, The Netherlands, EU (1993)
5. Hofstede, A., van der Weide, T.: Expressiveness in conceptual data modelling. Data & Knowledge Engineering **10**(1) (1993) 65–100
6. Hull, R., King, R.: Semantic database modeling: Survey, applications, and research issues. ACM Computing Surveys **19**(3) (1987) 201–260
7. van Bommel, P., Hoppenbrouwers, S., Proper, H., van der Weide, T.: Giving meaning to enterprise architectures - Architecture principles with ORM and ORC. In Meersman, R., Tari, Z., Herrero, P., eds.: OTM Workshops (2). Volume 4278 of Lecture Notes in Computer Science., Montpellier, France, EU, Springer, Berlin, Germany, EU (2006) 1138–1147
8. Overbeek, S., van Bommel, P., Proper, H., Rijsenbrij, D.: Characterizing knowledge intensive tasks indicating cognitive requirements – Scenarios in methods for specific tasks. In Ralyté, J., Brinkkempers, S., Henderson-Sellers, B., eds.: Proceedings of the IFIP TC8 / WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, University of Geneva, Switzerland, Springer, Boston, USA (2007)
9. Harel, D.: On visual formalisms. Communications of the ACM **31**(5) (1998) 514–530
10. Castello, R., Mili, R.: Visualizing graphical and textual formalisms. Information Systems **28**(7) (2003) 753–768
11. Trog, D., Vereecken, J., Christiaens, S., de Leenheer, P., Meersman, R.: T-Lex: A role-based ontology engineering tool. In Meersman, R., Tari, Z., Herrero, P., eds.: OTM Workshops (2). Volume 4278 of Lecture Notes in Computer Science., Montpellier, France, EU, Springer, Berlin, Germany, EU (2006) 1191–1200