Mathware & Soft Computing 7 (2000) 245-255

The Application of Generalised Constraints to Object-Oriented Database Models

G. de Tré & R. de Caluwe
Computer Science Laboratory
Dept. of Telecommunications and Information Processing
Ghent University
Sint-Pietersnieuwstraat 41,
B-9000 Ghent, Belgium

Abstract

A formal framework for a generalised object-oriented database model is presented, which is able to cope with fuzzy and uncertain information. This model is obtained as a generalisation of a crisp object-oriented database model, which is consistent with the ODMG de facto standard and is built upon an algebraic type system and a constraint system. Generalised constraints have been used to enforce integrity rules and to specify the formal semantics of the database model.

Keywords: Object-oriented database models, generalised constraints, fuzzy and uncertain information management.

1 Introduction

During the past decade several "fuzzy" object-oriented database models have been proposed [1, 2, 3, 4, 5, 6, 7, 8]. An overview can be found in [9]. The definitions of these models do not conform to a single underlying object data model, as a logical consequence of the present lack of (formal) object standards.

The ODMG de facto standard data model [10] offers new promising perspectives. However, it still suffers from several shortcomings such as some lack of formal semantics and its limited ability to deal with constraints, despite the fact that a thorough support of constraints is the most obvious way to guarantee the integrity of a database. In this paper a formal framework for the definition of an object-oriented database model is presented. This framework is consistent with the ODMG model and overcomes the mentioned ODMG shortcomings. It also supports both the modelling of fuzziness and uncertainty.

The fundamental concepts of the proposed model are the generalised type system and the generalised constraint system, which respectively allow to define generalised types and generalised constraints. The generalised type system is obtained as a generalisation of a crisp type system and establishes the definition of so-called generalized types. Three different kinds of generalised types are distinguished: generalised literal types, generalised object types and generalised reference types. Generic generalised constraints, as defined by L.A. Zadeh, are used to define the semantics of the components of a generalised object type. Beside generic generalised constraints, the model also deals with specific generalised constraints, which are used to define the semantics of a database and especially to guarantee the integrity of the data in a database. The generalised constraint system establishes the definition of specific generalised constraints and is obtained as a generalisation of a crisp constraint system. Both the generalised type system and the generalised constraint system are used to define generalised object schemes and generalised database schemes. A generalised object scheme describes the (semantics of the) common characteristics of a set of objects, whereas a generalised database scheme describes (the semantics of) a (fuzzy) database.

2 A formal framework for generalised object-oriented databases

Since a crisp database can be seen as a special case of a fuzzy and/or uncertain database, it has been the aim to define a generalised object-oriented database model, which encompasses the crisp one (rather than defining an extension of the crisp database model). This approach allows to define database schemes in a more "natural" way: every generalised type implicitly deals with fuzziness and no extra, specific "fuzzy types" are necessary to define the database scheme. As a consequence, the use of fuzzy sets will be as "natural" to the user as the use of crisp values.

2.1 The definition of types

Types have been defined as instances of a generalised type system. The definition of a generalised type system is the generalisation of the definition of a crisp type system which is in accordance with the directions given in [12] and is completely consistent with the ODMG specifications.

2.1.1 The crisp type system

In the type system, the rules which define the valid types of the data model are stated. In order to be consistent with the ODMG data model [10], a distinction has been made between the so-called void type (which is the most primitive type of the system and will be used in situations where no specific type is applicable), literal types, object types and reference types (which enable to refer to the instances of an object type and are used to formalize the binary relationships between the object types in the database scheme) [13]. A literal type is either a base type, a collection literal type or a structured literal type. The set of the type specifications of all the literal types will be denoted as $T_{literal}$, the set of the specifications of all the object types as T_{object}, and the set of the specifications of all the reference types as $T_{reference}$.

The definition rules for the type specifications are not limited to their syntax, but are also enriched with formal semantics. The proposed semantic definitions are related to domains and sets of domains. The semantics of a type specification ts are completely captured by a set of domains D_{ts}, a designated domain $dom_{ts} \in D_{ts}$, a set of operators O_{ts} and a set of axioms A_{ts}. Each domain contains a bottom value $\perp_{ts}$, which represents an "undefined" domain value [13].

The (crisp) type system TS, which allows to derive both the specification and the implementations of the valid types, is defined as the quintuple:

$$TS = [ID, T, P, f_{impl}^{type}, \prec]$$

in which

- ID is the set of the valid (type) identifiers,
- $T = \{\textbf{Void}\} \cup T_{reference} \cup T_{literal} \cup T_{object}$ is the set of the valid type specifications,
- P is the (infinite) set of all the (type) implementations,
- f_{impl}^{type} is called the (type) implementation function:

  $$\begin{aligned} f_{impl}^{type} : T &\rightarrow \wp(P) \\ ts &\mapsto \{p_1, \ldots, p_n\} \end{aligned}$$

  This function is a mapping which maps each type specification ts of the domain T onto the subset $\{p_1, \ldots, p_n\}$ of its co-domain $\wp(P)$ (the powerset of P) which contains all the implementations of ts and
- $\prec$ identifies the operator $\prec: T_{object}^2 \rightarrow dom_{Boolean}$, which defines a partial ordering on T_{object} and is used to define the inheritance-based type-subtype relationships between object types.

An instance t of the type system TS is called a type and is defined by a triple:

$$t = [ts, f_{impl}^{type}(ts), V_t^{instance}], \text{ with } ts \in T$$

If $ts = \textbf{Void}$, t is called a void type; if $ts \in T_{literal}$, t is a literal type; if $ts \in T_{object}$, t is an object type; otherwise t is called a reference type. The third component $V_t^{instance}$ is the set of all the instances of the type t. The instances of a literal type, of an object type and of a reference type are respectively called literals, objects and reference instances, whereas the void type cannot have instances at all.

Hereafter, the focus is on the object types, as they provide a formal basis for the definition of the object, which is the most important notion in the ODMG specifications. The syntax of the specification $ts \in T_{object}$ of an arbitrary object type t is defined by:

$$\textbf{Class } id : \widehat{id}_1, \ldots, \widehat{id}_m(id_1 : s_1; \ldots; id_n : s_n)$$

in which id is an identifier that represents the name of the object type, $\widehat{id}_i$, $1 \leq i \leq m$ are identifiers representing the supertypes of t (if any) and $id_i : s_i$, $1 \leq i \leq n$ are the characteristics (attributes, relationships and methods) which are explicitly specified within the syntax of the object type. For each $id_i : s_i$, $1 \leq i \leq n$, id_i is called the identifier of the characteristic, whereas s_i is called the specification of the characteristic. For attributes and relationships this specification is a type specification, for methods the specification is a signature.

Example 1: With **String**, **Integer**, **Set** and the enumeration type

$$\textbf{Enum } TLang(Dutch, French, English)$$

all being elements of $T_{literal}$, the specification ts of an object type

$$TPerson = [ts, f_{impl}^{type}(ts), V_t^{instance}]$$

can be given as:

$$\begin{aligned} \textbf{Class } TPerson(Name &: \textbf{String}; \\ Age &: \textbf{Integer}; \\ Languages &: \textbf{Set}(TLang)) \quad \diamond \end{aligned}$$

The properties, as well as the behaviour of both objects and literals, are formalised by the types of proposed type system. (As far as the literals are concerned, the definition of the behaviour is obviously limited to the behaviour which is implicitly defined.) An object o or instance of an object type is defined by the quadruple

$$o = [oid, N, t, v]$$

in which

- oid is a unique object identifier,
- N is a set of object names,
- $t = [ts, f_{impl}^{type}(ts), V_t^{instance}]$ is the object type and
- $v \in dom_{ts}$ is the state of the object.

N can be an empty set and for every attribute or relationship $id_i : s_i$ specified within ts, v contains a value $v_i \in dom_{s_i}$. The extent V_t^{extent} of an object type t is the set of all its persistent instances. If an object belongs to the extent of t, then it has to be an instance of type t, i.e. $V_t^{extent} \subseteq V_t^{instance}$. If type t is a subtype of type $\widehat{t}$, then the extent of t has to be a subset of the extent of $\widehat{t}$.

Example 2: An instance of $TPerson$ is, e.g.

$$[oid1, \emptyset, TPerson, \textbf{Struct}(\text{“Ann”}, 28, \textbf{Set}(Dutch, French, English))] \quad \diamond$$

2.1.2 The generalised type system

The proposed generalised (fuzzy) type system is obtained by generalising the definition of the crisp type system

$$TS = [ID, T, P, f_{impl}^{type}, \prec]$$

First of all, the set of type specifications T is generalised to the set $\tilde{T}$ by generalising the definitions of the domains and the operators of each type specification $ts \in T$. For each type specification $ts \in T$, the generalised counterpart $\tilde{ts}$ has a domain $dom_{\tilde{ts}}$ that is defined as the crisp set

$$\tilde{\wp}(dom_{ts})$$

of fuzzy sets on dom_{ts} and a set of operators $O_{\tilde{ts}}$ that contains the generalised counterparts of the operators of O_{ts} (generalised using Zadeh's extension principle).

Furthermore, the definition of a characteristic $id_i : s_i$ of an object type is generalised through the use of (generic) generalised constraints as defined by L.A. Zadeh [15]: each attribute or relationship is generalised as

$$\tilde{id}_i \; isr \; \tilde{ts}_i$$

where isr is a variable copula and r is a discrete variable which value defines the way in which the values of the (domain of the type of the) characteristic are constrained [16]. For now, the considered types of constraints are:

1. *Equality constraint*, $r = e$. In this case $\tilde{id}$ ise $\tilde{ts}$ means that $\tilde{id}$ will be assigned a "crisp" value of $dom_{\tilde{ts}}$. This case is semantically equivalent with the definition $id : ts$ in the crisp counterpart.
2. *Possibilistic constraint*, $r = blank$. In this case $\tilde{id}$ is $\tilde{ts}$ means that $\tilde{id}$ will be assigned a value of $dom_{\tilde{ts}}$, i.e. a fuzzy set, and $\tilde{id}$ is interpreted as a disjunctive (possibilistic) variable.
3. *Veristic constraint*, $r = v$. In this case, $\tilde{id}$ isv $\tilde{ts}$ means that $\tilde{id}$ will be assigned a value of $dom_{\tilde{ts}}$, i.e. a fuzzy set, and $\tilde{id}$ is interpreted as a conjunctive (veristic) variable.

With these generalisations, the generalised type system GTS is defined as the quintuple:

$$GTS = [ID, \tilde{T}, P, \tilde{f}_{impl}^{type}, \tilde{\prec}]$$

in which

- ID remains the set of the valid identifiers,
- $\tilde{T}$ is the set of the valid generalised type specifications,
- P remains the (infinite) set of all the (type) implementations,

- $\tilde{f}^{type}_{impl}$ is a mapping which maps each generalised type specification $\tilde{ts} \in \tilde{T}$ onto its set of implementations:

$$\begin{aligned} \tilde{f}^{type}_{impl} : \tilde{T} &\rightarrow \wp(P) \\ \tilde{ts} &\mapsto \{p_1, \ldots, p_n\} \end{aligned}$$

- $\tilde{\prec}$ identifies the operator $\tilde{\prec} : \tilde{T}^2_{object} \rightarrow dom_{\tilde{Boolean}}$, which defines a partial ordering on $\tilde{T}_{object}$ and is used to define the inheritance-based type-subtype relationships between generalised object types.

An instance $\tilde{t}$ of the generalised type system GTS is called a generalised type and is defined by the triple:

$$\tilde{t} = [\tilde{ts}, \tilde{f}^{type}_{impl}(\tilde{ts}), V^{instance}_{\tilde{t}}], \text{ with } \tilde{ts} \in \tilde{T}$$

As in the crisp case, the third component $V^{instance}_{\tilde{t}}$ is the set of all the instances of the generalised type $\tilde{t}$.

Example 3: With **$\tilde{\text{String}}$**, **$\tilde{\text{Integer}}$** and the generalised enumeration type

$$\tilde{\mathbf{Enum}}\ TLang(Dutch, French, English)$$

all being elements of $\tilde{T}$, the specification $\tilde{ts}$ of a (generalised) object type

$$GTPerson = [\tilde{ts}, \tilde{f}^{type}_{impl}(\tilde{ts}), V^{instance}_{\tilde{t}}]$$

can be given as:

$$\begin{aligned} \tilde{\mathbf{Class}}\ GTPerson(Name\ ise\ &\tilde{\mathbf{String}}; \\ Age\ is\ &\tilde{\mathbf{Integer}}; \\ Languages\ isv\ &TLang) \quad \diamond \end{aligned}$$

The generalisation of the definition of an object becomes:

$$\tilde{o} = [oid, N, \tilde{t}, \tilde{v}, \mu_{True}, \mu_{False}, \mu_{\perp_{\tilde{Boolean}}}]$$

in which

- *oid* remains a unique object identifier,
- N remains a set of object names,
- $\tilde{t} = [\tilde{ts}, \tilde{f}^{type}_{impl}(\tilde{ts}), V^{instance}_{\tilde{t}}]$ is the generalised object type,
- the state $\tilde{v}$ is an element of $dom_{\tilde{ts}}$ and

- μ_{True}, μ_{False} and $\mu_{\perp_{\tilde{Boolean}}}$ are three elements of the unit interval $[0, 1]$, which together express the fuzzy truth value

$$\{\mu_{True}/True, \mu_{False}/False, \mu_{\perp_{\tilde{Boolean}}}/\perp_{\tilde{Boolean}}\}$$

of the proposition "the object $\tilde{o}$ is an instance of the generalised object type $\tilde{t}$", i.e. μ_{True} is the degree to which this proposition is true, μ_{False} is the degree to which this proposition is false and $\mu_{\perp_{\tilde{Boolean}}}$ is the degree to which it is not defined that the object is an instance of the object type.

Example 4: An instance of the generalised object type $GTPerson$ is, e.g.

$$[oid1, \emptyset, GTPerson, \textbf{Struct}(\{(\text{“Ann”}, 1)\}, \{(26, .6), (27, .8), (28, 1), (29, 1), (30, .8), (31, .6)\}, \{(Dutch, .4), (French, .7), (English, 1)\}), 1, 0, 0] \quad \diamond$$

2.2 The definition of specific constraints

Specific constraints are used to enforce integrity rules on databases (e.g. domain rules, referential integrity rules, etc.) and to specify the formal semantics of a database model (e.g. null values, definition of keys, etc.) [11, 13]. In this approach a specific constraint is formally defined as an instance of a constraint system.

2.2.1 The crisp constraint system

Each specific crisp constraint is defined as a function $\rho : V_t^{instance} \rightarrow dom_{Boolean}$, which indicates whether:

- a given object can exist within the context of a given database ($True$),
- a given object can not exist within the context of a given database ($False$),
- or the constraint is not applicable to the given object ($\perp_{Boolean}$).

The crisp constraint system CS, which allows to derive both the specification and the implementations of the specific crisp constraints, is defined by the quintuple:

$$CS = [ID, T, C, P, f_{impl}^{constr}]$$

in which

- ID is the set of the valid identifiers,
- T is the set of the valid type specifications,
- C is the set of all the valid specific constraint specifications,
- P is the (infinite) set of all the (constraint) implementations and

- f_{impl}^{constr} is the (constraint) implementation function, which maps each constraint specification cs of the domain C onto the subset $\{p_1, \ldots, p_n\}$ of the co-domain $\wp(P)$ that contains all the implementations of cs:

$$\begin{aligned} f_{impl}^{constr} : C &\to \wp(P) \\ cs &\mapsto \{p_1, \ldots, p_n\} \end{aligned}$$

An instance c of the constraint system CS is called a specific constraint and is defined by the couple:

$$c = [cs, f_{impl}^{constr}(cs)], \text{ with } cs \in C$$

2.2.2 The generalised constraint system

The generalisation of the constraint system is straightforward: each constraint function ρ has been generalised to a function $\tilde{\rho} : V_{\tilde{t}}^{instance} \to dom_{\tilde{Boolean}}$, which associates with each object $\tilde{o} \in V_{\tilde{t}}^{instance}$ a fuzzy set

$$\{\mu_{True}/True, \mu_{False}/False, \mu_{\perp_{\tilde{Boolean}}}/\perp_{\tilde{Boolean}}\}$$

that represents the fuzzy truth value of the proposition "the object $\tilde{o}$ satisfies the constraint $\tilde{\rho}$", i.e. μ_{True} is the degree to which this proposition is true, μ_{False} is the degree to which this proposition is false and $\mu_{\perp_{\tilde{Boolean}}}$ is the degree to which the constraint is not applicable to $\tilde{o}$.
This leads to the definition of the generalised constraint system GCS

$$GCS = [ID, \tilde{T}, \tilde{C}, P, \tilde{f}_{impl}^{constr}]$$

in which

- ID is the set of the valid identifiers,
- $\tilde{T}$ is the set of the valid generalised type specifications,
- $\tilde{C}$ is the set of the valid specific generalised constraint specifications,
- P is the (infinite) set of all the (constraint) implementations and
- $\tilde{f}_{impl}^{constr}$ is the generalised (constraint) implementation function

$$\begin{aligned} \tilde{f}_{impl}^{constr} : \tilde{C} &\to \wp(P) \\ \tilde{cs} &\mapsto \{p_1, \ldots, p_n\} \end{aligned}$$

2.3 The definition of generalised object schemes and database schemes

Both the definitions of a generalised object scheme and of a generalised database scheme rely on the definition of the generalised types and the specific generalised constraints.

2.3.1 The generalised object schemes

The full semantics of an object are described by a generalised object scheme $\tilde{os}$. This scheme "in fine" completely defines the object, now including the specific constraints that apply to the object. The object scheme is defined by an identifier id, an object type $\tilde{t} \in \tilde{T}_{object}$ (i.e. an instance of the generalised type system GTS), a meaning $\tilde{M}$ and a conjunctive fuzzy set of constraints $\tilde{C}_{\tilde{t}}$ (i.e. a finite fuzzy set defined over the instances of the generalised constraint system GCS)

$$\tilde{os} = [id, \tilde{t}, \tilde{M}, \tilde{C}_{\tilde{t}}]$$

The meaning $\tilde{M}$ is an informal component of the definition and is usually described in a natural language [14]. The membership degree of an element of $\tilde{C}_{\tilde{t}}$ indicates to which degree the constraint applies to the object type $\tilde{t}$.

An instance $\tilde{o}$ of the object type $\tilde{t}$ is defined to be an instance of the generalised object scheme $\tilde{os} = [id, \tilde{t}, \tilde{M}, \tilde{C}_{\tilde{t}}]$ if it satisfies (with a truth value which differs from $\{1/False\}$) all the constraints of $\tilde{C}_{\tilde{t}}$ and all the constraints of the sets $\tilde{C}_{\hat{\tilde{t}}}$ of the object schemes $[\widehat{id}, \hat{\tilde{t}}, \widehat{\tilde{M}}, \tilde{C}_{\hat{\tilde{t}}}]$ which have been defined for the supertypes $\hat{\tilde{t}}$ of $\tilde{t}$.

The membership degrees μ_{True}, μ_{False} and $\mu_{\perp_{\tilde{Boolean}}}$ of $\tilde{o}$ result from the aggregation of the truth values of the satisfaction of the constraints.

2.3.2 The generalised database schemes

A generalised database scheme $\tilde{ds}$ describes the structure and the behaviour of the information which is stored in generalised database and is defined as the quadruple

$$\tilde{ds} = [id, \tilde{D}, \tilde{M}, \tilde{C}_{\tilde{D}}]$$

in which id is the identifier of the database scheme,

$$\tilde{D} = \{\tilde{os}_i = [id_i, \tilde{t}_i, \tilde{M}_i, \tilde{C}_{\tilde{t}_i}] | 1 \leq i \leq n, i, n \in N_0\}$$

is a finite set of generalised object schemes, $\tilde{M}$ represents the meaning of $\tilde{ds}$, and $\tilde{C}_{\tilde{D}}$ is a conjunctive fuzzy set of constraints which imposes extra conditions on the instances of the set of generalised object schemes $\tilde{D}$ (e.g. referential constraints between two object schemes). Again, the membership degrees are an indication for the relevance of the constraints. Every generalised object scheme in $\tilde{D}$ has a different object type. If a generalised object scheme $\tilde{os}_i \in \tilde{D}$ is defined for an object type $\tilde{t}$ and $\tilde{t}'$ is a supertype of $\tilde{t}$, or $\tilde{t}'$ is binary related with $\tilde{t}$, then there exists a generalised object scheme $\tilde{os}'_i \in \tilde{D}$ which is defined for $\tilde{t}'$.

An instance of a generalised object scheme $\tilde{os}_i \in \tilde{D}$ of a generalised database scheme $\tilde{ds} = [id, \tilde{D}, \tilde{M}, \tilde{C}_{\tilde{D}}]$ is defined to be an element of the extent of the generalised object scheme if it satisfies (with a truth value which differs from $\{1/False\}$) all the constraints of $\tilde{C}_{\tilde{D}}$.

The membership degrees μ_{True}, μ_{False} and $\mu_{\perp_{\tilde{Boolean}}}$ of an element $\tilde{o}$ of the extent of a generalised object scheme are calculated taking into account the aggregation of the truth values of the satisfaction of the constraints of $\tilde{C}_{\tilde{D}}$.

2.4 The definition of a (fuzzy) database

An instance of a generalised database scheme

$$\tilde{ds} = [id, \tilde{D}, \tilde{M}, \tilde{C}_{\tilde{D}}]$$

is called a (fuzzy) database and is by definition the set of all the extents of the generalised object schemes of $\tilde{D}$.

3 Conclusion

A formal framework for the definition of a fuzzy and/or uncertain object-oriented database model has been presented. This framework is based on a type system and a related constraint system, which is meant to guarantee database integrity. Zadeh's extension principle and generalised constraints have been used to support the database model in which each object has an associated truth value. Finally, databases have been defined as sets of sets of objects.

In the presented framework, both the data and the semantics of the data can be modelled. Generalised constraints are provided to support the modelling of the data semantics:

- Generic generalised constraints, as defined by Zadeh, are applied to specify the semantics of the domain values of the object types in a database scheme.
- Specific generalised constraints are provided to model the integrity rules, which are necessary to guarantee the database integrity.

The approach of defining a generalised database model, encompassing the "traditional" one, allows to define database schemes in a more "natural" way: every generalised type is suited to handle fuzzy, as well as crisp information. In this way no extra, specific "fuzzy types" are necessary to define the database scheme.

References

[1] Rossazza, J.-P. (1990), *Utilisation de hiérarchies de classes floues pour la représentation de connaissances imprécises et sujettes à exception: le système "SORCIER"*. PhD. Thesis, Université Paul Sabatier, Toulouse, France.

[2] Tanaka, K. et al. (1991), *Uncertainty Management in Object-Oriented Database Systems.* Proceedings of the International Conference on Database and Expert System Applications, DEXA 1991, Karagiannis, D. (ed.), Springer-Verlag, Berlin, Germany, pp. 251–256.

[3] George, R. (1992), *Uncertainty Management Issues in the Object-Oriented Database Model.* PhD. Thesis, Tulane University, New Orleans, LA, USA.

[4] Bordogna, G., Pasi, G. and Lucarella, D. (1999) *A Fuzzy Object-Oriented Data Model for Managing Vague and Uncertain Information.* International Journal of Intelligent Systems, Vol. 14, No. 7, pp. 623–651.

[5] Van Gyseghem, N. (1998) *Imprecision and Uncertainty in the UFO Database Model.* Journal of the American Society for Information Science, Vol. 49, No. 3, pp. 236–252.

[6] Rocacher, D. and Connan, F. (1996) *Fuzzy Algebra for Object Oriented Databases.* Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing, EUFIT'96, Vol. 2, Elite Foundation, Aachen, Germany, pp. 871–876.

[7] Na, S. and Park, S. (1997) *Fuzzy Object-Oriented Data Model and Fuzzy Association Algebra.* In: Fuzzy and Uncertain Object-Oriented Databases: Concepts and Models, De Caluwe, R. (ed.), Advances in Fuzzy Systems — Applications and Theory,Vol. 13, World Scientific, Singapore.

[8] Mouaddib, N. and Subtil, P. (1997) *Management of Uncertainty and Vagueness in Databases: The FIRMS Point of View.* International Journal of Uncertainty, Fuzziness and Knowledge Based Systems, Vol. 5, No. 4, pp. 437–457.

[9] De Caluwe, R. (ed.) (1997). *Fuzzy and Uncertain Object-Oriented Databases: Concepts and Models.* Advances in Fuzzy Systems - Applications and Theory, Vol. 13, World Scientific, Singapore.

[10] Cattell, R. G. G. et al. (2000). *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann Publishers Inc., San Francisco, CA USA.

[11] De Tré, G. and De Caluwe, R. (1999). *A generalised object-oriented database model with generalised constraints.* Proceedings of the NAFIPS'99 conference, IEEE, New York, NY, pp. 381–386.

[12] Lausen, G. and Vossen, G. (1998). *Models and Languages of Object-Oriented Databases.* Addison-Wesley, Harlow, Engeland.

[13] De Tré, G. et al. (2000). *A generalised object-oriented database model.* In: Recent research issues on the management of fuzziness in databases, Bordogna, G. and Pasi, G. (ed.), Studies in Fuzziness and Soft Computing, Vol. 53, Physica-Verlag, Heidelberg, Germany.

[14] Paredaens, J. et al. (1989). *The Structure of the Relational Database Model.* EATCS: Monographs on Theoretical Computer Science, Vol. 17, Springer-Verlag, Berlin, Heidelberg.

[15] Zadeh, L. A. (1997). *Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic.* Fuzzy Sets and Systems, Vol. 90, No. 2, pp. 111–127.

[16] Dubois, D. and Prade, H. (1997). *The tree semantics of fuzzy sets.* Fuzzy Sets and Systems, Vol. 90, No. 2, pp. 141-150.