THE APPLICATION OF GENERALISED CONSTRAINTS TO OBJECT-ORIENTED DATABASE MODELS

Guy de Tré
University of Ghent
Computer Science Laboratory
Sint-Pietersnieuwstraat 41,
B-9000 Ghent, Belgium
guy.detre@rug.ac.be

Rita de Caluwe
University of Ghent
Computer Science Laboratory
Sint-Pietersnieuwstraat 41,
B-9000 Ghent, Belgium
rita.decaluwe@rug.ac.be

Summary

This paper presents a formal framework for a generalised object-oriented database model that is able to cope with fuzzy and uncertain information. The generalised model is obtained as a generalisation of a crisp object-oriented database model that is compliant with the ODMG de facto standard and is built upon an algebraic type system and a constraint system.

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. An overview can be found in [2]. These models do not conform to a single underlying object data model, due to a lack of object standards.

The ODMG de facto standard data model [1] 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. The formal framework we present is ODMG compliant and overcomes the mentioned ODMG shortcomings. It also supports both fuzziness and uncertainty.

By building a formal generalised type system and constraint system we can give the formal definition of a generalised object and a generalised database scheme, both of them being the main components of our generalised object-oriented database model.

2 A formal framework for generalised object-databases

Since crisp databases can be seen as special cases of fuzzy and/or uncertain databases, it is our aim to define a generalised object-oriented database model which is a generalisation of a crisp one (rather than defining an extension of a crisp database model).

2.1 The definition of types

Types are defined as instances of a generalised type system. Our 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 [4] and completely compliant with the ODMG specifications.

2.1.1 The crisp type system

In our type system, we state the rules which define literal types, object types, reference types and the **Void** type. A literal type as 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 $T_{reference}$.

We do not limit ourselves to the syntax of the type specifications, but also endow the definition rules with formal semantics. Our 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 an "undefined" value $\perp_{ts}$.

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

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

where $T = \{\mathbf{Void}\} \cup T_{reference} \cup T_{literal} \cup T_{object}$ is the set of valid type specifications, P is the infinite set of all programs —indepedent of the language in which they are written— and f_{impl}^{type} is called the (type) implementation function:

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

f_{impl}^{type} 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) that contains all the implementations of ts.

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

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

When $ts = \mathbf{Void}$, t is called a void type; when $ts \in T_{literal}$, t is a literal type; when $ts \in T_{object}$, t is an object type; otherwise t is called a reference type.

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

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

where id is an identifier that represents the name of the object type, $\widehat{id}_i, 1 \leq i \leq m$ are identifiers representing the parent types 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 the identifier and s_i is the specification of the characteristic, which for attributes and relationships is a type specification and for methods is a signature.

Example 1: With **String**, **Integer**, **Set** and the enumeration type **Enum** $TLang(Dutch, French, English)$ being an element of $T_{literal}$, the specification ts of an object type $TPerson = [ts, f_{impl}^{type}(ts)]$ can be given as:

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

The properties as well as the behaviour of both objects and literals are formalised by the types of our 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 as a quadruple

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

where oid is a unique object identifier, N is a set of object names, $t = [ts, f_{impl}^{type}(ts)]$ 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 t, v contains a value $v_i \in dom_{s_i}$. The extent of an object type is the set of all its (persistent) instances. If an object is an instance of type t, then it has to be a member of the extent of t. If type t is a subtype of type $\widehat{t}$, then the extent of t is a subset of the extent of $\widehat{t}$.

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

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

2.1.2 The generalised type system

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

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

First of all, the set of type specifications T is generalised to a set $\tilde{T}$ by generalising the definitions of the domains and operators of the type specifications $ts \in T$. For every type specification $ts \in T$, the generalised counterpart $\tilde{ts}$ has a domain $dom_{\tilde{ts}}$ that is defined as the 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 generalised constraints as defined by L.A. Zadeh [6]: a generalised attribute or relationship is defined 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) characteristic are constrained. 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 $id : ts$ of its 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 seen 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 seen as a conjunctive (veristic) variable.

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

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

where $\tilde{T}$ is the generalised set of valid type specifications, P remains the infinite set of all programs and $\tilde{f}_{impl}^{type}$ is a mapping which maps each generalised type specification $\tilde{ts}$ of $\tilde{T}$ onto its set of implementations $\tilde{ts}$:

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

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

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

Example 3: With **String**, **Integer** and the enumeration type **Enum** $TLang(Dutch, French, English)$ being an element of $\tilde{T}$, the specification $\tilde{ts}$ of a (generalised) object type $GTPerson = [\tilde{ts}, \tilde{f}_{impl}^{type}(\tilde{ts})]$ can be given as:

Class $GTPerson($ $Name\ ise$ **String**;
$Age\ is$ **Integer**;
$Languages\ isv\ TLang)$ $\diamond$

The generalisation of the definition of an object becomes:

$$\tilde{o} = [oid, N, \tilde{t}, \tilde{v}, Pos, Nec]$$

where oid remains a unique object identifier, N remains a set of object names, $\tilde{t} = [\tilde{ts}, \tilde{f}_{impl}^{type}(\tilde{ts})]$ is the generalised object type, and the state $\tilde{v}$ is now an element of $dom_{\tilde{ts}}$. Additionally, a generalised object has a (system defined) possibility measure Pos and a necessity measure Nec, which can take values between 0 and 1, and which together express the fuzzy thruth value

$$\{Pos/true, Nec/False\}$$

of the object as an instance of the generalised object type $\tilde{t}$.

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

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

2.2 The definition of constraints

Constraints are used to enforce integrity rules on databases (e.g. domain rules, referential integrity, etc.) and to help to specify the formal semantics of a database model (e.g. null values, definitions of keys, etc.) [3]. In our approach a constraint is formally defined as an instance of a constraint system.

2.2.1 The crisp constraint system

A crisp constraint is defined as a Boolean function, which indicates whether a given object is valid within the context of a given database or not. The (crisp) constraint system CS, which allows to derive both the specification and the implementations of the constraints, is defined as the triple:

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

where C is the set of the specifications of all the valid constraints, and P and f_{impl}^{constr} denote respectively the infinite set of all programs and the (constraint) implementation function, which is defined as:

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

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

An instance c of the constraint system CS is called a constraint and is defined by a 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: a Boolean (constraint) function can be generalised to a function with co-domain $[0, 1]$ that associates a membership degree with a given object, indicating to which degree the object satisfies the constraint. As well the constraints which specify the domain rules, as the constraints which specify the transition rules are generalised in this way, which leads to the definition of the generalised constraint system GCS.

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

where $\tilde{C}$ is the generalised set of valid constraints specifications, P is the set of all programs and $\tilde{f}_{impl}^{constr}$ denotes 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

The adopted scheme definitions are the counterparts of the ones presented in [5] for the relational database model. Both the definitions of generalised object scheme and generalised database scheme rely on the definitions of types and constraints.

2.3.1 A generalised object scheme

The full semantics of an object are described by a generalised object scheme $\tilde{os}$. This scheme defines all the characteristics of the object, including the constraints that apply to the object and is defined by an object type $\tilde{t}$ (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 over the instances of the generalised constraint system GCS).

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

The meaning $\tilde{M}$ is an informal component of the definition since it will mostly be described in a natural language [5]. 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 of the object type $\tilde{t}$ is defined to be an instance of the generalised object scheme $\tilde{os} = [\tilde{t}, \tilde{M}, \tilde{C}_{\tilde{t}}]$ if and only if it satisfies (with a non-zero membership degree) all the constraints of $\tilde{C}_{\tilde{t}}$ and all the constraints of the sets $\tilde{C}_{\hat{\tilde{t}}}$ of the object schemes $[\hat{\tilde{t}}, \widehat{\tilde{M}}, \tilde{C}_{\hat{\tilde{t}}}]$ which have been defined for the supertypes $\hat{\tilde{t}}$ of $\tilde{t}$.

The possibility measure Pos and the necessity measure Nec result from the degrees to which the object satisfies the constraints of $\tilde{C}_{\tilde{t}}$ (by calculating t-conorms and t-norms).

2.3.2 A generalised database scheme

A generalised database scheme $\tilde{ds}$ describes the information which is stored in the generalised database and is defined as a triple:

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

where $\tilde{D} = \{\tilde{os}_i = [\tilde{t}_i, \tilde{M}_i, \tilde{C}_{\tilde{t}_i}] | 1 \leq i \leq n, i, n \in N_0\}$ 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 object schemes (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 $\tilde{D}$ is defined for 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 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 $\tilde{os}_i \in \tilde{D}$ is defined to be an instance of the generalised database scheme $\tilde{ds} = [\tilde{D}, \tilde{M}, \tilde{C}_{\tilde{D}}]$ if and only if it satisfies all the constraints of $\tilde{C}_{\tilde{D}}$. All the instances of the generalised database scheme $\tilde{ds}$ are elements of the extent of $\tilde{ds}$.

As is the case for the instances of an object scheme, the measures Pos and Nec result from the degrees to which the object satisfies the constraints of $\tilde{C}_{\tilde{D}}$ (by calculating t-conorms and t-norms).

2.4 The definition of a (fuzzy) database

The extent of a generalised database scheme $\tilde{ds}$ is called a (fuzzy) database and is by definition a set of conjunctive fuzzy sets of database instances (grouped per object scheme). For each database instance

$$\tilde{o} = [oid, V_{names}, \tilde{t}, \tilde{v}, Pos, Nec]$$

the membership degree of the instance within its conjunctive fuzzy set is calculated as $(Pos + 1 - Nec)/2$ and can be seen as the fuzzy thruth value of the object in the database.

3 Conclusions

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 generalise the database model. Subsequently, databases have been defined as sets of fuzzy sets of objects.

References

[1] Cattell, R. G. G. et al. (1997). *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers Inc., San Francisco, CA USA.

[2] 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.

[3] 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.

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

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

[6] 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.