Representing Complex Semantics in Databases

Ph.D. Dissertation Summary

by

Gábor Surányi

supervised by

Dr Gábor Magyar

Budapest, Hungary
December 18, 2009
1. INTRODUCTION

Traditionally and fundamentally, databases are the common back-ends of various software systems which manage huge amount of data. Technology development has affected and affects revolutionarily the role of databases in software systems over the decades. Here I focus on two aspects of the effects only.

Object-oriented (OO) data modelling is finally available in databases. However, these data models still have some lag behind the capabilities of state-of-the-art OO modelling tools used in software engineering.

Databases are planted into all kinds of computer systems as main memory and disk storage capacities increase while the prices of a unit drop. The aim is to record all available data and use them, probably in an unforeseen way, to maximise product quality and/or profit.

Both phenomena impose new requirements on the representation capability of the databases. On one hand it should be rich to catch up with the capability of software engineering tools, on the other hand they should be open to incorporate not foreseen model elements or to store data elements which are not conform to the pre-established model.[22, C7] This is required by the (data) semantics which gets more and more complex along with the increasing intelligence of software systems. The other side of the coin is the retrieval capability, which has as well to be present and match the representation capability — it does not suffice to store something without being able to retrieve it.

The data representation is determined at the design time of databases. During database design the following steps are carried out:

1. selecting a data model,
2. designing the external/conceptual schema,
3. laying down the physical organisation.

All of them are targets of my seeking representation methods for complex semantics.

1.1 Rich, Object-Oriented Data Models

Although databases based on the relational data model are still very common, OO databases are widely employed in new software systems. The reasons are well
known: the OO paradigm offers a high abstraction level while retaining intuitiveness. Moreover, since new software applications are almost exclusively OO, there is no discrepancy in the representation of live (in-memory) and stored (on-disk) entities.

There exists a standard for object persistence in databases, The Object Data Standard (latest version is 14) created by the Object Data Management Group (ODMG). However, OO metamodels\(^1\) tend to become richer and richer in order to describe the model-world more precisely. One of the modelling capabilities OO data models (including the object model of the ODMG standard) miss is the universal use of constraints. The universal use means more than the enforcement of the traditional integrity constraints. It should cover all areas OO models used in analysis, design, implementation and testing of software do.

In the need for a sole OO metamodel which is adequate for most purposes, the Unified Modeling Language Specification (UML)\(^{37, 38}\) and Object Constraint Language (OCL)\(^{31}\) of Object Management Group, Inc. (OMG) emerged. They influenced all other, less commonly used OO models as well as the ODMG standard.

### 1.2 Ontologies: Open Database Schemata

Indeed, existing data models are already capable of representing database schemata which are open due to the earlier realised need to manage semistructured data\(^9, 1\). But DBMS\(^2\) do not give any further help in retrieving the semantics which is complex in the following sense: all data to be entered into a database have to be disassembled into basic units (e.g. records) but once stored, is it really necessary to retrieve only the same disassembled units? The answer is definitely no, but this is how DBMS\(^2\) (including those managing semistructured data, see e.g. \(^10, 1\)) have worked.\(^2\)

For instance, information retrieval (IR) systems are affected by this behaviour. In this case, huge databases are employed to manage terms, resources (i.e. documents) and their relationship. The retrieval method may seem trivial: returning resources which are related to the given terms. However, hits obtained by this method are likely to be high in number and not to contain all resources the user is interested in. Other, sophisticated methods exist which overcome these deficiencies. Amongst all, ontology-based IR (OBIR) systems are nowadays the most popular ones.

An ontology is a specification of a conceptualization.\(^{23, 27}\) The elements of the ontology are index terms in an OBIR system. The various relationships between the ontology elements (OE’s) are used to judge the similarity of OE’s, which serves as the basis of looking up resources relevant to the user query (also

\(^1\) A metamodel defines elements to describe models, i.e. concrete descriptions of important properties of entities. In the common use, metamodels are often just called models since from the context it is usually clear if a metamodel or a strictly meant model is referred to. Here I retain this tradition unless it causes ambiguity.

\(^2\) In fact, since stored procedures were introduced into DBMS’s it is possible to implement sophisticated query methods. However, the elementary retrieval method behind stored procedures still operates on small units.
1. Introduction

composed of index terms). That is, the OBIR system is to answer queries like

**Which records are described by similar records as given?** \( \text{Q}_{\text{similar}} \)

Clearly, this is more sophisticated than allowed by DBMS' since the word 'similar' is not a query primitive for them. By the business logic it is eventually translated into a query the DBMS can process. ('Described by' is just a many-to-many relation understood by all DBMS').

Open schemata are the generalisation of simple (i.e. logic-free) ontologies since they just represent various relations between records. Here the same problem arises: there exists no built-in method to interpret relationships for querying.

1.3 Enhanced Physical Databases

Physical data organisation deals with the layout of data units on storage media with the sole goal to improve response times to queries. Traditionally, exact results are delivered to queries. However, approximate results are gaining significance.\[^2\] For instance, outside the relational world set values are quite common\[^3\] and often an exact match cannot be expected but an approximate (closest) match suffices.

The logical proximity of data elements is determined by the data elements themselves, after all. The challenge in this layer is therefore to grasp the distance between data elements and represent it in the physical database.

\[^3\] Relational database design usually involves normalisation, which requires all attributes be atomic. Therefore set values are often split into elements and additional relations in relational databases.
2. RESEARCH OBJECTIVES

The objective of the dissertation was to enhance the representation of data semantics in databases. Having identified related open issues in all the database design steps, I tackled the following problems.

**A data model supporting the features of UML and OCL is desired.** This basically means there is a need for an OO data model which universally supports constraints. Furthermore, the data model should have a solid type system. The fundamental purpose of type systems is to prevent the occurrence of errors during the execution of programs merely by analysing their code. Here I aimed at designing a type system which ensures error-free operation w.r.t. value constraints set forth by object invariance, operation and state-based role\(^1\) specifications.

**A methodology for time efficient retrieval from databases with open schemata considering relationships between data items has to be founded.** Artificial intelligence made effective retrieval already possible by succeeding in grasping relations between records. Such methods are based either on single relationships of record pairs or on the similarity of a pair of record sets. Methods of the latter class naturally have a more global overview of the semantics and therefore provide better results. Their drawback is that they can be applied pairwise only, which does not scale well. I aimed at developing a methodology to integrate the pairwise method into an efficient query processing system. Once this objective is reached, it can be expected that DBMS\(^2\) will offer query primitives like \((Q_{\text{similar}})\) (see page 4) built-in.

**Efficient physical organisation for values with partial order needs to be found to support closest as well as exact match queries.** Partial orders\(^2\) have not attracted much interest in physical database organisation despite the fact that queries over partial orders model frequent problems [33].

---

1. The term role is used throughout this work as in UML 1.5: the named specific behavior of an entity participating in a particular context [36]. See Section 4.1.1 for details.
2. A partial order is a binary relation which is reflexive, transitive and antisymmetric. [5]
3. METHODOLOGY

The data model and the type system to develop are mathematical structures which had to be conceived. However, it is still true that re-inventing the wheel is not a wise idea. Therefore my mathematical structures are based on earlier work, Fernandes’ axiomatic data model\cite{fernandes} and Castagna’s OO calculus, λ\cite{castagna}. This part has solid theoretical background, which enabled analytical evaluation.

On the contrary, efficient retrieval with ontologies is rather an engineering problem. Thus re-use and adapting results of related areas of computer science which I was aware of was a must. This is palpable in the related section, e.g. in the construction of the blackboard system\cite{blackboard1,blackboard2}. Validation was empirical here, i.e. a group of target users tested the IR system which employed my method and the users’ opinion was collected.

Finding efficient physical organisation is in a way a mixture of the above approaches: re-use and invention, engineering and science. By formalising the task I have transformed the problem into mathematics and realised that it is widely explored there already. Despite the disappointing generic results coming from mathematics, there existed already applications in a few domains but domain-neutral databases were not amongst them. Thus my new physical organisation can be considered as an adaptation of the earlier theoretical and practical results plus an optimisation thereof. The optimisation was partially enabled by other earlier results of mathematical analysis and applications in computer science. Furthermore, mathematics enabled my result to be extended to pre orders\footnote{A pre order is a binary relation which is reflexive and transitive.\cite{preorder}} since a partial order can be derived from each pre order \cite{partialorder}.

\footnote{A pre order is a binary relation which is reflexive and transitive.\cite{preorder}}
4. NEW RESULTS

4.1 Leveraging Constraint-Enhanced OO Models

4.1.1 The Constraint-Enhanced Axiomatic OO Data Model

There exists already an axiomatic OO data model, which is described in detail in [18]. Although that model could support, it does intentionally not deal with application-specific integrity constraints so that queries can be extensively optimised.

As already mentioned in the Introduction, this standpoint is no longer adequate. Herewith I show that the support of application-specific integrity constraints in a data model is theoretically possible and fruitful. However, I leave some of the practical issues (such as decidability) now unresolved, they have to addressed in the respective applications just as done in Section 4.1.2.

Claim 1. I have defined an axiomatic OO data model which supports application-specific constraints [J1, C6].

Definition 1. Let $L$ be a logic language which consists of

- an infinite set of variable symbols,
- a set of constant (nullary function) symbols which stand for class, object identifiers and atomic constants,
- a set of predicate symbols: $\mathcal{P}$.
- a set of non-constant function symbols: $\mathcal{F}$.
- the auxiliary symbols ( and )
- the logical connectives $\neg, \land, \lor, \Rightarrow$.
- the quantifiers $\forall$ and $\exists$.

The elements of $\mathcal{P}$ are:

- unary symbols for each atomic type (e.g. integer, string),
- basic predicate and relation symbols needed for the atomic types (e.g. =, >).
• the unary symbols class, object,
• the binary symbols specialize, instance,
• binary symbols for each attribute name (e.g. name).
• \((n+1)\)-ary or \((n+2)\)-ary symbols for the names of each operation taking \(n\) arguments.

First-order logic (FOL) with any \(L\) characterised above is the axiomatic OO data model which supports application-specific constraints.

Let \(A\) be the set of the following formulae:

\[
\forall c \text{ class}(c) \Rightarrow \neg \text{object}(c) \quad (4.1)
\]

\[
\forall o \text{ object}(o) \Rightarrow \neg \text{class}(o) \quad (4.2)
\]

\[
\forall c_1 \forall c_2 \text{ specialize}(c_1, c_2) \Rightarrow \text{class}(c_1) \land \text{class}(c_2) \quad (4.3)
\]

\[
\forall c \forall o \text{ instance}(c, o) \Rightarrow \text{class}(c) \land \text{object}(o) \quad (4.4)
\]

\[
\forall \text{ class}(c) \Rightarrow \text{specialize}(c, c) \quad (4.5)
\]

\[
\forall \text{ specialize}(c_1, c_2) \land \text{specialize}(c_2, c_1) \Rightarrow c_1 = c_2 \quad (4.6)
\]

\[
\forall \text{ specialize}(c_1, c_2) \land \text{specialize}(c_2, c_3) \Rightarrow \text{specialize}(c_1, c_3) \quad (4.7)
\]

\[
\forall o \exists c \text{ object}(o) \land \text{instance}(c, o) \Rightarrow \text{instance}(c, o) \quad (4.8)
\]

\[
\forall c_1 \forall c_2 \forall o \text{ specialize}(c_1, c_2) \land \text{instance}(c_1, o) \Rightarrow \text{instance}(c_2, o) \quad (4.9)
\]

A set \(\Pi\) of closed formulae of \(L\) is a database schema if it is consistent and \(\Pi \vdash A\). A structure \(S\) corresponding to \(L\) is a database if \(S \models \Pi\). Any closed formula \(\varphi\) of \(L\) is a query. Upon querying it needs to be indicated as well whether it is to be evaluated as

• \(S \models \varphi\), i.e. whether the query formula is currently true in the database or

• \(\Pi \vdash \varphi\), i.e. whether the query formula is always true in all possible states of the database.

In the former case, the DBMS shall return a (variable) assignment for the existentially quantified variables the quantors of which are not preceded by universal ones in the prenex normal form of the query.

Thesis 1.1. I have shown that the data model introduced in Definition 1 is really OO, i.e. it has equivalents of all fundamental concepts of object-orientation: class, object, generalisation, polymorphism.

In accordance with the Introduction, this thesis is proven by checking the compatibility of the model to the constructs of UML\cite{37, 38}. Although UML itself defines 'compliance levels', rather compatibility than compliance is addressed here.

\(^{1}\) Whence the adjective axiomatic of the data model comes: the database schema contains propositions which must hold for a concrete database.
because even the lowest compliance level requires all elements of the (UML) Basic package have an equivalent in the compliant model. But I investigate only if the model has equivalents of all fundamental concepts of object-orientation: class, object, method, generalisation, polymorphism. Mapping all elements of the Basic package would make my data model unnecessarily complex. With compatibility it is ensured that, if needed, the data model can be augmented to be UML-compliant.

**Definition 2** (Class and object[38]). A class describes a set of objects that share the same specifications of features, constraints and semantics. A class is a kind of classifier whose features are attributes and operations. [...] Some of these attributes may represent the navigable ends of binary associations.

A method is an implementation of an operation.

In my model, objects are entities, which are identified by constants and only for such constants *object (o)* holds. The objects have various features, including

- attributes, represented by the respective binary predicates, e.g. *name (object, string)*,

- binary associations, represented like attributes,

- *n*-ary operations, represented by the respective (*n*+1)-ary and (*n*+2)-ary predicates. The additional arguments are needed for the owner (in the context of which the operation is invoked, first argument) and, if there is any, for the return value (second argument), e.g. *bear (parent_object, integer, child_object)*, which can mean that the operation returns the new number of child objects.

Classes are, too, model entities described by constants. For such constants *class (c)* holds but they are different than objects: (4.1)-(4.2). That objects belong to classes is described by the axiom (4.8) using the predicate *instance (c, o)*. The domain of the predicate arguments is determined by (4.4). That all objects of a class have a certain feature can be formalised:

\[\forall o \forall a_1 \ldots \forall a_n \exists r \quad \text{instance}(c, o) \wedge \text{instance}(c_1, a_1) \wedge \ldots \wedge \text{instance}(c_n, a_n) \Rightarrow \text{FEATURE}(o, r, a_1, \ldots, a_n) \wedge \text{instance}(c, r) \quad (4.10)\]

where FEATURE is the predicate symbol of the feature, *a*<sub>1</sub>, ..., *a*<sub>n</sub>, are only present if the feature is an operation, not an attribute or an association. As already mentioned, *r* may be omitted for operations if there is no return value. A formula of the form (4.10) assigns the feature to the class identified by *c*<sub>o</sub> in the formula.

The previously enumerated formulae still allow an object to have features not defined by its classes. To disallow this, one can add formulae of the form

\[\forall o \forall a_1 \ldots \forall a_n \forall r \quad \text{FEATURE}(o, r, a_1, \ldots, a_n) \Rightarrow \text{instance}(c_0, o) \wedge \text{instance}(c_1, a_1) \wedge \ldots \wedge \text{instance}(c_n, a_n) \wedge \text{instance}(c, r) \quad (4.11)\]

to the database schema.

**Constraints** are any other\(^2\) arbitrary formulae which are part of the schema. For example, constraints may describe object invariance criteria, operation pre-
and postconditions, which are specific to the concrete application. However, the model introduced above is limited to constraints which can be expressed in FOL.

Constraints also include methods, which are traditionally defined as universally closed implications [27] (i.e. constraints which define the relationship between the operation input and the output):

\[
\text{BODY} \Rightarrow \text{OPERATION}(o, r, a_1, \ldots, a_n) \quad (4.12)
\]

A constraint is assigned to a class identified by \( c \) if the formula contains no other class identifier than \( c \) and

- if the formula all predicate symbols which correspond to features are assigned to \( c \).

**Definition 3** (Generalisation[38]). A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

In my model, generalisation is represented by the `specialize` binary predicate: (4.3). Its usual (partial order) properties are described by formulae (4.5)–(4.7). That an object is also instance of a more general class is formalised by the formula (4.9). In this way, whenever a feature of a generic class is referred to (in a formula, e.g.), it is ensured that the same feature of all more specific class is as well referred to: the feature is inherited.

**Definition 4** (Polymorphism[12]). The operands (actual parameters) of polymorphic operations can have more than one type.

There are two types of polymorphism: universal and ad-hoc.[12] In practice both of them are important but since ad-hoc polymorphism is just a syntactic abbreviation for a finite set of different types [12], I consider only universal polymorphism here.

Universal polymorphism can be inclusion or parametric.[12] Inclusion polymorphism w.r.t. classes was recently discussed at generalisation and it was shown to be supported by my axiomatic OO data model. Since it is universal, by definition parametric polymorphism works on an infinite number of types having a common structure.[12] Parametric polymorphism is usually realised in one of the following two ways [12]:

- by template constructs which need to be explicitly bound (instantiated) before use as in UML[38] and e.g. in the programming language C++[11].

- by generic constructs which operate on any entity fulfilling a set of requirements. This is typical of functional programming languages like ML[29] but also supported by UML via type stereotypes[38].

My data model for the sake of simplicity and because of genericness employs the latter. As in UML, the notion of class covers these type stereotypes as well. Because databases traditionally have a long lifespan, there is one additional notion of type that databases have to support: roles[21, 19, 36][32] The concept is
widely-used in general in OO analysis and design but has unfortunately many different names, not even the 1.5 and 2.0 versions of UML use the same term (see ClassifierRole vs. ConnectableElement in [36] and [38], respectively).

The diversity in terminology partially arises because there are two role representation methods: explicit and implicit. In the case of explicit representations, that an object plays a role is expressed by a ‘dynamic object’ or a ‘role object’, which is created and destroyed as needed. The corresponding terms for roles include dynamic classes (see e.g. [28]) and role types (see e.g. [21]).

The implicit role representation derives role membership from the features and state of the objects automatically, no additional objects are required. Such roles are hence also called state-based roles. The terms virtual classes (see e.g. [34]) and even just types (see e.g. [C6] and ConnectableElement in [38]) are also used for roles. The term type is justified by the fact that a role is actually no more than a set of requirements to be fulfilled by the object instances. Since an attribute may represent the existence of a role object, the implicit representation subsumes the explicit one.

It has to be clear that there is a fundamental difference between the notions interface and (implicit) role. Both of them are sets of requirements but interfaces are realised by classes and therefore all instances of those classes inherently fulfil the requirements of the interface, while (implicit) roles are populated with objects of any class if they fulfil the criteria of the role.

**Thesis 1.2.** I have shown that the data model introduced in Definition 1 supports implicit role representations via its regular notion of class.

This is achieved via classes to which formulae of the form

\[ \forall o \text{ CONDITION } \Rightarrow \text{ instance}(c,o) \]  

are assigned. CONDITION may reference features: their existence, values etc.

### 4.1.2 Proving Partial Correctness w.r.t. Constraints in OO Environments

Safety-aware design and implementation are no longer the privilege of mission-critical computer software. Constraints describing object invariants and/or pre- and postconditions of message processing are a great aid in pursuing the avoidance of design flaws and the elimination of implementation errors.

**Claim 2.** I have defined two functional calculi, \( \lambda & \) and \( \lambda & ' \) to be able to prove the partial correctness of any OO program w.r.t. value constraints set forth by object invariance, operation and arbitrary state-based role specifications. Specifications which describe relations between the input and the output of operations are not supported.

I have proven that my calculi bear the necessary properties for this purpose. [TR, J2]

I have chosen \( \lambda & \)-calculus[13] as the basis of my calculi, because, on one hand, its features include all fundamental OO phenomena (classes, operations
with multiple dispatch\(^3\), generalisation). On the other hand, a variant of \(\lambda &\) incorporates another useful feature called bounded polymorphism (for modelling e.g. type-preserving functions), and a similar technique may be applied to my results to gain more powerful OO calculi.

The basic idea behind \(\lambda &\) and \(\lambda &\) is that the notion of (pre-)type from \(\lambda &\) is extended with a constraint set. Well-formed formulae of the constraint sets are first order and are built from atomic formulae with the standard logical connectives and quantifiers. Each free variable of a constraint formula shall appear as a lower-left index of a part of the (pre-)type to which the formula belongs. As suggested, each of these variables refers to the part of the (pre-)type expression it marks. As a consequence, these index variables have all to be different within each (pre-)type.

The actual set of predicate and function symbols can be freely chosen and are usually determined by the application domain, i.e. by the pre-types. (But for computability reasons, function symbols except constants may be disallowed, see Thesis 2.4.) A few predicate symbols, namely for each atomic type a unary symbol needs to be defined, however. Their semantics are that the parameter is of that type (i.e. an element of the domain of that type) and they are used, besides in the constraint sets of (pre-)types, in the (domain-specific) axioms of logical derivability.

In \(\lambda &\) and \(\lambda &\) the definition of types, terms, the subtyping relation and the type system and the reduction rules are adapted in accordance with the constraint-extended definition of pre-types. The difference between the two calculi is the definition of types and the notion of reduction so that they treat constraints of types in a slightly different way. In \(\lambda &\)-calculus constraints are considered as parts of state-based role specifications, and as such they are obeyed during type evaluation as well as term reduction. \(\lambda &\)-calculus strictly adheres to meaning of specifications, i.e. they do not affect the semantics, the execution of programs. More precisely, \(\lambda &\) enforces constraints via its type system but it does not check constraints during term reduction. Both systems possess the properties needed to be considered useful.

**Thesis 2.1** (Soundness [J2]). I have proven that in \(\lambda &\) and \(\lambda &\): the type of a term may not change as a result of reducing it.

**Thesis 2.2** (Confluence [J2], for state-based roles also [TR]). I have proven that in \(\lambda &\) and \(\lambda &\): the final result of term reduction does not depend on the order the individual reductions are performed in.

Both of my calculi extensively use the apparatus of FOL and in FOL several problems are known to be undecidable. Therefore it had to be investigated if FOL makes impossible the practical application of \(\lambda &\) and \(\lambda &\). I have identified two impacted areas: type consistency and derivability.[TR, J2]

Types are not required to be consistent concerning their constraint set, i.e. the set may be unsatisfiable. This means, there can be types which cannot type any term. This is impractical as such types indicate modelling error and consume system resources without any advantage. So do functions which take terms of \(\lambda &\) and \(\lambda &\) the notion of (pre-)type from \(\lambda &\) is extended with a constraint set. Well-formed formulae of the constraint sets are first order and are built from atomic formulae with the standard logical connectives and quantifiers. Each free variable of a constraint formula shall appear as a lower-left index of a part of the (pre-)type to which the formula belongs. As suggested, each of these variables refers to the part of the (pre-)type expression it marks. As a consequence, these index variables have all to be different within each (pre-)type.

The actual set of predicate and function symbols can be freely chosen and are usually determined by the application domain, i.e. by the pre-types. (But for computability reasons, function symbols except constants may be disallowed, see Thesis 2.4.) A few predicate symbols, namely for each atomic type a unary symbol needs to be defined, however. Their semantics are that the parameter is of that type (i.e. an element of the domain of that type) and they are used, besides in the constraint sets of (pre-)types, in the (domain-specific) axioms of logical derivability.

In \(\lambda &\) and \(\lambda &\) the definition of types, terms, the subtyping relation and the type system and the reduction rules are adapted in accordance with the constraint-extended definition of pre-types. The difference between the two calculi is the definition of types and the notion of reduction so that they treat constraints of types in a slightly different way. In \(\lambda &\)-calculus constraints are considered as parts of state-based role specifications, and as such they are obeyed during type evaluation as well as term reduction. \(\lambda &\)-calculus strictly adheres to meaning of specifications, i.e. they do not affect the semantics, the execution of programs. More precisely, \(\lambda &\) enforces constraints via its type system but it does not check constraints during term reduction. Both systems possess the properties needed to be considered useful.

**Thesis 2.1** (Soundness [J2]). I have proven that in \(\lambda &\) and \(\lambda &\): the type of a term may not change as a result of reducing it.

**Thesis 2.2** (Confluence [J2], for state-based roles also [TR]). I have proven that in \(\lambda &\) and \(\lambda &\): the final result of term reduction does not depend on the order the individual reductions are performed in.

Both of my calculi extensively use the apparatus of FOL and in FOL several problems are known to be undecidable. Therefore it had to be investigated if FOL makes impossible the practical application of \(\lambda &\) and \(\lambda &\). I have identified two impacted areas: type consistency and derivability.[TR, J2]

Types are not required to be consistent concerning their constraint set, i.e. the set may be unsatisfiable. This means, there can be types which cannot type any term. This is impractical as such types indicate modelling error and consume system resources without any advantage. So do functions which take terms of

\(^3\) Multiple dispatch means that method selection is based on taking into account types of all arguments, not only the type of the receiver of the message.
inconsistent types as input. Lastly, in order to ensure that an unambiguous branch always exists to be selected in an overloaded function application, the definition of types may require an overloaded function to have an inconsistent branch (a function which would never be invoked because no argument can satisfy the constraints specified for the input of the function).

**Thesis 2.3** (Consistent Types [TR, J2]). *I have shown that type consistency can be enforced while my calculi retain their soundness and confluence properties.*

Derivability in FOL is one of the problems which are undecidable in general thus a decidable subclass had to be selected for my calculi. Since the domain of the OO model can be arbitrary I have opted for selecting decidable subclasses based on quantifier prefixes of the formulae and the cardinality of predicate and function symbols of the logical language. The book [6] was very useful for this purpose as it exhaustively enumerates all maximal decidable and minimal undecidable cases w.r.t. this classification.

**Thesis 2.4** (Decidable Calculi [J2]). *I have given a sufficient criterion for the practical usability of my calculi in terms of decidability.*

Subtyping is surely decidable if all formulae used in a model belong to one of the following classes.

**Bernays-Schönfinkel-Ramsey class:** In the prefix form, existential quantifiers have to precede universal ones and no function symbols except constants are allowed in the language.

**Gurevich class:** The prefix form of the formulae contains only existential quantifiers. Function and predicate symbols of any arity may occur.

**Shelah class:** The prefix form of the formulae contains a single universal quantifier and at most one unary function symbol. The number of existential quantifiers in the prefix and the number of predicate symbols are not limited.

Formulae needed for type consistency (see Thesis 2.3) rule out the two latter classes.

That the Bernays-Schönfinkel-Ramsey class disallows non-constant function symbols does not affect the expressive power of the calculi, since each atomic formula

$$p\left(f_1(f_{11}(\ldots), f_{1k}(\ldots)), f_2(f_{21}(\ldots), f_{2l}(\ldots))\right)$$

with variables and constants $z_1, \ldots, z_n$ can in general be represented by

$$p'(z_1, \ldots, z_n)$$

or

$$\forall x_1 \forall x_2 \ p_1(x_1, z_1, \ldots, z_n) \land p_2(x_2, z_1, \ldots, z_n) \Rightarrow p(x_1, x_2)$$

with appropriate semantics. In the latter case $p_i$'s are predicate symbols replacing $f_i$'s and their first argument is the return value of the original function. By the way, the restriction on function symbols was the reason for modelling operations
with predicates in Definition 1 although theoretically functions could also have been appropriate.

The above condition for decidability is only sufficient, not necessary, i.e. a particular formula set may imply a decidable system although it does not belong to these classes.

In accordance with the corresponding research objective, this section outlined how

- my constraint-enhanced axiomatic OO data model and
- my functional calculi to support proving partial correctness of OO programs enhanced with value constraints

look like. The detailed descriptions of the systems introduced in this section are available in the dissertation.

4.2 Efficient Retrieval with Ontologies

There is a simple idea to turn effective retrieval into efficient retrieval: it is not worth comparing records which obviously do not have much in common, the system should rather filter out such elements first with a 'fast' algorithm, then it should apply the costly pairwise comparison method to the rest (to the much smaller candidate result set).

This idea is actually generalised from the idea for OBIR systems from [30]. This is possible because I adapted a very general definition of logic-free ontology for this purpose (as it was already mentioned in the Introduction). In the following just as in the dissertation itself, however, I stick to the terminology of ontologies to ease understanding. One can map the terms to the ones of open schemata unambiguously.

Claim 3 (Calculating the candidate result set). I have designed a (sub)system for DBMS to support them in efficiently calculating the answer set to the query \( Q_{\text{similar}} \) (see page 4) provided that in the database

- descriptions are realised via sets of possibly weighted OE's and
- certain expansion rules are defined for OE sets as described in the following. [C5]

The subsystem delivers the candidate records for final similarity (relevance) judgement in two steps.

In the first step, the subsystem expands the query (disregarding any weight). This should be done carefully, however, to gain an efficient system.

Thesis 3.1 (Agent-based parallelised query expander). To resolve this I have proposed to employ a blackboard system[16, 15] which is depicted in Figure 4.1. The so-called knowledge sources need also to be tailored to the particular ontology (to take advantage of the properties of the various relationships modelled), i.e. their definition via the expansion rules has to be part of the database.
4. New Results

The blackboard stores the current expanded query, and is initialised with OE’s contained in the query. Then each knowledge source (agent) is responsible for ensuring that the blackboard is closed under its own expansion rule. They accomplish their task by continuously examining the contents of the blackboard and adding further OE to them if required. There is no separate control shell, i.e. the expansion phase completes as soon as the blackboard is found closed under the expansion rules by all the knowledge sources. The finiteness of ontology guarantees the termination of the expansion process.

The method is inherently parallelised since the agents work independently.

Query expansion is a highly data-driven and complex task, as the inclusion of an entity may imply the inclusion of further entities in the expanded query. However, the use of a blackboard system for this purpose successfully decouples the functionality from the data. It means that only the knowledge sources, i.e. the expansion rules are specific to the ontology (or more precisely to the relations of the ontology), the architecture is ontology independent.

**Thesis 3.2** (Calculating the candidate result set from the expanded query). *For the actual filtering step I have proposed the following realisation: the candidate result set is the set of resources which are described with at least one of the OE’s contained in the final expanded query, retrieved from the blackboard after the query expansion. This step is very simple and I have shown that it is realisable with a single traditional database query.*

Of course, more sophisticated methods for this latter step are conceivable. However, the user evaluation of the realised OBIR system incorporating this method (see also Section 5) showed that this also delivers satisfactory results in terms of speed and quality.

With respect to the research objective, Claim 3 enables (even huge) databases to offer complex retrieval method based on attributes for which an ontology exists.

### 4.3 Partial Orders in Physical Databases

After the informal introduction in Sections 1.3 and 2, let me formalise the problem I dealt with in connection with physical databases.

For any attribute $A$, let $d_A$ denote the value of $A$ of the data element $d$ and $D_A$ the domain of $A$. The task to accomplish is to efficiently answer the following queries which reference

---

4. This means that the contents of the blackboard are initially equal to the query but weights of OE’s are removed if there was any.
4. New Results

- the attribute $A$ for which $D_A$ is a partially ordered set (poset) with $\preceq_A$ and
- the given parameter $v \in D_A$.

**Definition 5 (Atomic Queries over Attribute with Partial Order).**

$\max_A(v)$: Return all data elements $d$ for which $d \preceq_A v$ and there is no data element $d'$ for which $d_A \neq d_A' \land d \preceq_A d' \preceq_A v$.

$\min_A(v)$: Return all data elements $d$ for which $v \preceq_A d_A$ and there is no data element $d'$ for which $d_A \neq d_A' \land v \preceq_A d' \preceq_A d_A$.

$\similarity_A(v) = \max_A(v) \cup \min_A(v)$.

The last atomic query can be seen as the approximate equivalent of a simple lookup using equality because this requests data elements $d$ for which $d_A = v$ if there is any, otherwise some data elements we may call 'closest'. However, such a result is actually not a real closest match since it may be empty although no ambiguity is present. For instance, let a single-attribute database contain only the set-valued element $\{3, 7\}$. Then if the partial order relation is set inclusion, $\similarity(\{7, 13\}) = \emptyset$ though the sole data element of the database has an element in common with the given parameter. Complex queries (atomic queries, possibly other types than defined above, connected with logical connectives) can be answered via employing general query processing techniques (see e.g. [17]).

To efficiently answer means that secondary storage access (i.e. input/output, IO from/to media) has to be minimised; this is a basic database principle. In order to simplify calculations I assume that reaching each (data or auxiliary) element needs one storage IO. This is justified by the fact that nothing else is generally known about attribute, record sizes etc. but the number of required IO accesses (i.e. the time complexity) is somehow proportional to them.

I have laid down the following assumptions/constraints which served as a basis to search for appropriate solutions to the problem just described.

- The number of data records in the database can be extremely huge and even if more records belong to the same element of a poset, the number of poset elements can as well be very large. This means the whole poset cannot be maintained in memory. There are indeed application areas where no practical limit can be imposed on the size of the poset, such as database integration[26, C2, C4].

- No poset encoding (i.e. bitstream representation of the elements with bitwise operators to answer queries)[2] may be employed. The reason is that they are expensive to compute and maintain (and thus impractical) in databases storing huge posets since any data change may theoretically imply re-calculation of the codes of all vertices.

- No additional data may be saved during query evaluation about already visited poset elements because caching incurs increased memory footprint proportional to the size of the poset. This is undesired in databases of huge posets as the number of parallel queries can as well be very high.

All my results are to be understood in this context.
Claim 4. I have defined two auxiliary structures for databases along with query and maintenance (insert, delete) algorithms to support the above queries. I have also investigated the space requirement of the structures and the costs of the various algorithms. [C3, C4]

The first auxiliary structure, the naïve one, is basically the directed graph (digraph) representation of the partial order on the values of A, i.e. the vertices correspond to attribute values and the edges represent partial order relations. Moreover, the graph is reflexively and transitively reduced. This representation (which is called the core graph, denoted by $G_c$) is extended with additional vertices representing the data elements which have a particular value as A. Each of these vertices is connected to the corresponding one introduced earlier. [C3, C4] (Refer to the dissertation for the formal definition and for an example how this graph can be realised on block devices such as disks.)

Let $n$ denote the half of the number of vertices in the graph (which is, by definition, equal to the number of vertices in the core graph) and $N$ the number of data elements in the database. Let the greater from number of sources (vertices without any predecessor) and sinks (vertices without any successor) in the core graph be denoted by $s$ and the longest path (in the core graph) by $p$. With this notation I made the following claims.

**Thesis 4.1 (Space allocation[C3]).** I have shown that the naïve auxiliary structure occupies $O(n^2 + N)$ space.

---

```plaintext
function min_A(v)
  return all data elements represented by any vertex connected to an element of $min'_A(v)$
endfun

function min'_A(v)
  part := []
  foreach element represented by a source as node do
    if node $\preceq_A v$ then return $min'_A(node,v)$
    if v $\preceq_A node$ then part := part $\cup \{node\}$
  return part
endfun

function min'_A(node,v)
  if v $\preceq_A node$ then return $\{node\}$
  part := []
  foreach direct successor of node in $G_c$ as next do
    if next $\preceq_A v$ then return $min'_A(next,v)$
    if v $\preceq_A next$ then part := part $\cup \{next\}$
  return part
endfun
```

Fig. 4.2: Calculating $min_A(v)$

---

As usual, an update can be realised as an insert following a delete.
4. New Results

**Thesis 4.2** (Query algorithms). I have constructively proven that a full traversal is not needed to answer the queries (unlike done by others in similar naïve representations for general partial orders). I have shown that the query realisations which I defined for my auxiliary structure and are based on the (improved depth-first traversal) algorithm described in Figure 4.2 are correct. I have also proven that it is possible to calculate the answer to the atomic queries in posets with limited number of neighbours in $O(s + p + N)$ time in the worst case [C3] with my algorithms.

The term $N$ is always due to the fact that the hits (the data elements themselves) are to be returned.

**Thesis 4.3** (Maintenance algorithms for the naïve representation). I have proven that the insertion/deletion algorithms which I defined for my naïve auxiliary structure are correct and in general as efficient as they can be, they run in $O(n^2 + N)$ time [C3].

Because querying the naïve graph representation may still be as slow as the brute force method (not only for posets of all incomparable elements, in case the successors are enumerated in an 'unlucky' order), I proposed to represent the part of the poset actually stored in the database as chains (i.e. vertex-disjoint paths). [C4] (An example realisation of this structure on block devices is given in the dissertation.)

The work [3] already represented posets as chains not on secondary storage but in memory. This difference in application implies different unit for time complexity of query realisations. Moreover, [3] proposed a binary search strategy instead of the linear scan for traditional lookup (exact match) queries and it measured its efficiency (empirical evaluation). So my contribution here was the adaptation of the chain representation to secondary storage, improvement of my query realisations with binary search and the formal complexity analysis of the algorithms.

The chain representation is based on a chain decomposition of the digraph representing the poset (i.e. of the core graph). In such a chain decomposition it suffices to store only the edges between the chains (more precisely, from each vertex of a chain there is at most one edge to each other chain) in addition to the chains themselves. The rest of the representation is not altered, i.e. the vertices representing the data elements are still present and connected to the vertices of the chains just as in the naïve graph representation.

Note that the chain representation is no different than the naïve representation in terms of edges (and of course vertices) stored. Its core graph is only decomposed into vertex-disjoint paths, which are totally ordered subsets and they are as such in turn especially suitable for simple linear organisation.

Let $w$ denote the number of chains in a chain representation.

**Thesis 4.4** (Sharper bound for space allocation of graph representations). I have shown that the chain representation occupies $O(wn + N)$ space and the same applies to the naïve representation as well but there $w$ is not explicitly known.

Since $n \geq w$ always evidently holds, this new bound is stricter than the one stated in Thesis 4.1. Furthermore, it is also known that:
• the least possible $w$ (denoted by $w_{\text{min}}$) equals to the size of a maximal antichain (Dilworth’s theorem\cite{5} for finite graphs).

• from \cite{35}: for random graphs with probability $1 > q > 0$

$$E(w_{\text{min}}) = O\left(\frac{\ln(nq)}{q}\right)$$

where $E$ stands for the expected value.

**Thesis 4.5** (Query algorithms for the chain representation). *I have adapted my original realisations of atomic queries to employ a binary search (c.f. Figure 4.3). I have shown that they are correct and that their time complexity is

$$O\left(w + w\log_2(p+1) + N\right)$$

irrespective of the poset.

For the special case of total orders, where $w=1$ and $n=p+1$, querying over chain representations is as fast as using indices.

Analogously to the naïve graph representation, maintaining the chain representation (inserting/deleting data elements) basically means maintaining the core graph, i.e. here the chain decomposition itself because the rest of the graph is very simple. The maintenance of chain decompositions is a well-elaborated area, see e.g. \cite{25, 3, 24, 7}. The related literature describes algorithms which can be adapted to my chain representation in a straightforward manner.

By comparison with the research objective, this chapter provided efficient domain-neutral physical organisation for partial orders. Closest match queries were also dealt with in this context.
4. New Results

function \( \min_A(v) \)
return all data elements represented by any vertex connected to an element of \( \min'_A(v) \)
endfun

function \( \min'_A(v) \)
part := []
for \( i := 1 \ldots w \) do
node := chains[\( i \)][1]
if node is not a source then continue
if \( v \preceq_A \) element represented by node then part := part \( \cup \) {node}
if element represented by node \( \preceq_A \) \( v \) then
return part \( \cup \) \( \min'_A(\text{\( i \)th chain},1,v) \)
endfun

function \( \min'_A(\text{\( \text{chain}, \text{item}, v \)}) \)
node := \( \text{chain}[\text{item}] \)
if \( v \preceq_A \) element represented by node then return \{node\}
part := []
last := binary_search(\( v \), \( \text{chain}, \text{item}, \text{length of chain} \))
if \( v \preceq_A \) element represented by \( \text{chain}[\text{last}] \) then return \{\( \text{chain}[\text{last}] \}\}
foreach \( c \in \text{chains not yet visited by the whole algorithm} \) do
item := neighbour(\( \text{chain}, \text{last}, c \))
if 0=\( \text{item} \) then continue
next := \( \text{c}[\text{item}] \)
if \( v \preceq_A \) element represented by next then part := part \( \cup \) {next}
if element represented by next \( \preceq_A \) \( v \) then
return part \( \cup \) \( \min'_A(\text{\( c \)}, \text{\( \text{item}, v \)}) \)
endfun

function binary_search(\( v \), \( \text{chain}, \text{first}, \text{last} \))
if \( \text{first} = \text{last} \) then return first
item := \( \text{\( \frac{\text{first}+\text{last}}{2} \)} \)
if \( \text{chain}[\text{item}] \preceq_A \text{v} \) then
return binary_search(\( v \), \( \text{chain}, \text{item}, \text{last} \))
else return binary_search(\( v \), \( \text{chain}, \text{first}, \text{item}-1 \))
endfun

function neighbour(\( \text{chain}, \text{item}, \text{next} \))
return the ordinal of the item in \( \text{chain} \) next which is reachable from \( \text{chain}[\text{item}] \) via an edge. 0 if there is no such item
endfun

Fig. 4.3: Calculating \( \min_A(v) \) with chains
5. APPLICATION OF THE RESULTS

The **constraint-enhanced axiomatic OO data model** is actually a framework (in line with the latest needs) for modelling objects and constraints with axioms in databases. The framework itself does not consider practical applicability (such as decidability, complexity), they need to be addressed in the particular applications. For this purpose a great aid is the rich literature of various applications of FOL. All such optimisation problems are already tackled there.

The **functional calculi**, $\lambda\&\$ and $\lambda\&\$, serve as a basis for proving partial correctness in OO programs with constraints (see related theses for the precise list of features and limitations). Note that the calculi have a broader application potential: they may be applied outside the database domain, generally in OO programming environments, in OO software engineering solutions.

The application of the calculi for the purpose of proving partial correctness consists of the following steps.

1. Identifying the atomic types.
2. Formalising the predicate properties as axioms. In mathematical logic this is called axiomatising the theories which are attached to the predicates. Each proof ($\vdash$) will make use of these axioms.
3. (Re-)writing the classes (including the state-based roles), the specifications into pre-types and the methods (functions) into terms of $\lambda\&\$.
4. Computing the (real) type $V$ of each term and comparing it with the intended (given) type $W$. If and only if $V \leq W$ holds the term fulfils its requirements.

A concrete illustrative example is described in [J2].

As pointed out in the corresponding section of the dissertation, both the data model and the functional calculi can employ description logic instead of FOL.

The **method for efficient retrieval from databases with open schemata considering relationships between data items** was tailored for and applied in a web-based, prototype OBIR system\(^1\) for texts on European history. The system was described in detail in [C5], here its elements are briefly reviewed with focus on improving query response time.

Resource descriptions in the system (which, along with any OE set, were actually called contexts in the system's terminology) consisted of two parts: a time interval and a conceptual part, i.e. a set of OE's. Elements in descriptions were weighted with real numbers between 0 and 1. There was also a so-called Contextualisation Engine available, which performed two tasks:

\(^1\) World Wide Web homepage: [http://www.eurohistory.net](http://www.eurohistory.net)
1. generation of descriptions for new resources,

2. similarity calculation for any two descriptions.

Each resource query made direct use of the second, which was carried out only for a reduced set of resource descriptions. The selection of descriptions to compare implemented my filtering method. That is the whole query answering method (calculation of relevant resource descriptions) worked as depicted in Figure 5.1.

Fig. 5.1: Fast calculation of the relevant data elements applied in OBIR [C5]

The OBIR system was only one outcome of the Visual Contextualisation of Digital content (VICODI) project\(^2\) funded by the Information Society Technologies in the 5\(^{th}\) Framework Programme of the European Union. The goal of the project was actually to enhance people’s comprehension of digital content on the Internet. To this end, resources were automatically contextualised and the contexts were browsable and searchable. The search process too was novel: OE’s were highlighted and by clicking on them, a new resource query was initiated by altering the current context with the selected OE. At last but not least the local copy of each resource was displayed with the elements referring to any of its context elements highlighted. [C5]

In accordance with the global goal of the project, a user evaluation of the whole system (web portal) was planned. The first user evaluation was conducted before all functionalities were released but the system already incorporated the query answering system. For this reason, the user evaluation assessed not only the query answering system but the whole search process, which cannot be evaluated by the traditional IR metrics such as precision and recall\(^3\).

Nevertheless, the results of the evaluation met the expectations: the users were satisfied with the OBIR system. This basically refers to the quality of the final pairwise comparison method of query answering, i.e. that must possess good precision and recall properties but it also says that

1. its first step, i.e. the filtering/calculating the candidate result set does not affect these properties noticeably.

---


\(^3\) Precision: the ratio of relevant hits out of all hits. Recall: the ratio of relevant hits out of all relevant resources.[4]
2. the response time is sufficient.

The first point dispels any possible doubts which question if the simple step of obtaining the candidate result step from the expanded query is adequate. Of course, more sophisticated methods for this step would be conceivable.

The second point is important in the spirit of my goal to improve efficiency. It has to be mentioned however that some users were not satisfied with the retrieval speed, in fact they expected it would be in the order of transferring and finally displaying the web page as it is in the case of today's popular free-text search engines of the World Wide Web. The retrieval speed of our system actually lied in the range of a few seconds, which was acceptable both in terms of absolute values and considering the number of resources stored, the programming language used (Java\textsuperscript{20} without any stored procedure) and the available computing capacity (entry-level personal computer server).

The native physical organisation for partial and pre orders is primarily suitable for large, repository-like databases because the modification (insertion, update, deletion) algorithms are time-consuming. In such data stores data are less often altered and the performance gain at query time is significant compared to other methods due to the large size of the store. In the future it could be investigated what conditions on the pre order or the data have to hold so that data modification also becomes fast and where that could be applied.

My physical organisation also supports closest match queries. Since the subset relation is a pre order and semantic hierarchies are actually pre order relations on the elements, most of the scenarios are supported where exact matches are basically sought but if none is available, closest matches are expected.
References


Publications

Journal papers


Conference papers


[C5] Gábor Mihály Surányi, Gábor Nagypál, and Andreas Schmidt. Intelligent retrieval of digital resources by exploiting their semantic context. In Robert Meersman and Zahir Tari, editors, CoopIS/DOA/OBASE (1), volume


**Technical documents**