F2Concept, a Database System for Managing Classes' Extensions and Intensions

FALQUET, Gilles, LEONARD, Michel Paul, SINDAYAMAZE, Jeanne
F2\textsuperscript{Concept}: a Database System for Managing Classes’ Extensions and Intensions.\footnote{This research was supported in part by the Swiss-FNRS under grant 12-33958.92.}

Gilles Falquet, Michel Léonard, Jeanne Sindayamaze

Centre Universitaire d’Informatique
Université de Genève
24, rue Général-Dufour
CH-1211 Genève 4, Switzerland
(falquet | leonard | sindayam)@cui.unige.ch

Abstract
This paper introduces the F2\textsuperscript{Concept} database system that is an extension of the F2 object-oriented system. The main feature of F2\textsuperscript{Concept} is its ability to represent intensional information about database classes. Intensional information, usually found in terminological databases, is useful for database modelling, operation, and evolution. A description of how this kind of information is represented and stored is given as well as how to use it in conjunction with extensional information represented by objects. This constitutes a new approach for databases that consists in managing not only objects of classes but also descriptions of the exact semantics of classes.

1 Introduction

One of the most active research area in the field of databases is the enrichment of databases with knowledge [13]. Our approach consists in enriching databases with intensional information, it is inspired by practice and research in the fields of terminology and databases. Terminology [12] and terminological knowledge representation [3], [4], [11], [14] concentrate on the definition of terms designating concepts of a domain and on relationships between these terms, while the database area concentrates mainly on representation and storage of objects that are concepts’ instances.

The notion of concept in terminology and the notion of class in object-oriented data models are closely related. Indeed, classes are used to classify and represent instances of real-world concepts. The objects of a given class have attributes and methods to respond to incoming messages. This set of attributes and methods, shared by all the objects of a class, is similar to the intension of a concept. The objects themselves represent instances of the concept and form its extension.

Databases management systems are intended to efficiently manage classes’ extensions but existing database models are not adapted to represent the intension of classes. Thus a large part of the intensional knowledge that leaded to the definition of the database structure is lost. If an organization wants to have intensional information as well as information on every object of a class, it is obliged to have two databases: a terminological database which allows to answer to questions such as “how do we call a furniture with a back and two armrests?” and get the answer “armchair” and a traditional” database to store information such as “serial-numbers of armchair sold during
the month of October”. This leaded us to define the F2\textsuperscript{Concept} model, an extension of the F2 object-oriented data model, that permits to store in the same database extension of classes as well as their intension (i.e. definition of the concept corresponding to a class).

The remainder of this paper is organised as follows. The definition and the importance of classes’ intension for databases are discussed in section 2. Section 3 presents the F2\textsuperscript{Concept} model. Section 4 explains how to express intensional and extensional queries on a F2\textsuperscript{Concept} database. Section 5 details the semantics of inheritance and multivaluation. Sections 6 describes the F2 object-oriented data model and the implementation of F2\textsuperscript{Concept} based on the reflective architecture of F2. Section 7 compares F2\textsuperscript{Concept} with other approaches. Finally, we present our conclusions in section 8.

2 Intension of Concepts in Databases

2.1 Definition

The intension of a concept is the set of all its characteristics. Concepts intensions are often expressed by definitions given in natural language (e.g. in dictionaries or in documentation). An object (abstract or concrete) that possesses all the characteristics of a concept's intension is called an instance of the concept, and the set of all instances of a concept is called its extension. An instance may have additional characteristics that are not in its intension. For example, the intension of concept “chair” is “a seat with one back and no armrests”. The extension of the concept is formed by all the objects that are seat, have one back, and don’t have armrests. But a particular instance of “chair” may have other characteristics like being black, costing $45.00, etc.

2.2 Why Intensional Information in Databases?

The conceptual schema is at the heart of a database since it describes an abstract representation schema, i.e., it specifies how real world entities and their relationships are represented in the database. As we already mentioned, real world concepts are often represented by database classes. The database modelling process consists in defining a database schema that is suitable for the intended applications. During this process the designer acquires considerable knowledge about the concepts of the application domain. But this knowledge, which was necessary to build the schema, cannot be represented in the database schema due to the absence of intensional information structure. Indeed, the intensional knowledge actually stored in a standard database schema consists of attributes names and domains, methods names and signatures, and classes names and hierarchies. Thus a large part of the intensional knowledge that leaded to the definition of the database schema is eventually lost.

The description of classes’ intensions may greatly help designers, programmers, and users interacting with a database since they often need to know the precise meaning of the classes they are dealing with. One can distinguish several situations in which intensional knowledge plays an important role.
High Level Database Queries

Consider for instance a furniture shop, it has customers, employees, chairs, armchairs, etc. A database schema for that shop could contain the following class definitions:

```java
class Chair(prodId : Int, prodName : String, width : Real, price : Real)
class Armchair(prodId : Int, prodName : String, width : Real, price : Real)
class Employee(empId: Int, name: String, salary: Real)
class Customer (custId: Int, name : String, adress : String, telephone: String)
```

This structure permits the management of objects that are member of the armchair, chair, employee, and customer concept’s extensions. It is possible to store information about objects, such as “the chair with product id 554 costs $34.00” or “the phone number of the customer named Fred is 566-5401”.

It is also possible to answer to questions like “what is the price of the chair named Nikita-k12” or “what are the product id of tables having less than 70 centimetres of width”. However there is an important class of question the answer to which cannot be found in the database. These are intensional questions like “what is a chair?” or “what is the difference between a chair and an armchair?”. The database is of no help to answer these questions because it lacks intensional information about these concepts. The only thing that we know about chairs is that they have a product id, a product name, a width, and a price; but these are not essential characteristics of chairs.

In general, the description of classes’ intensions is necessary to determine which class to access to get the required information, i.e. in which class the required data is located so that the appropriate query can be formulated.

Synonyms and homonyms.

In database schemata classes are designated by a single term (that is not always one of those used in the common language to designate the corresponding concept). This does not correspond to the reality of an organization where different terms are used to designate the same concept. For instance a type of chair may be called “furniture-x34” by the production department, and “comfort-6” by the sale department.

Database maintenance and evolution.

The database designer possesses, at design time, considerable knowledge about classes’ intensions. This knowledge, used during the design process, is of primary importance when making the database schema evolve. Since the database administrator, who is in charge of database maintenance, is generally not the database designer there must be a way to store and communicate the designer’s knowledge to the administrator.

Reusability.

Intensional information is stable knowledge since concepts do not change so often. Hence, classes’s intensions are potentially more reusable than traditional database schemata. For instance, two organisations often manage the same concepts’ intension but have different structure for rep-
resenting concepts’ extensions. Thus an organization could reuse the intensional structure of a similar organization and define its own structure for extensions.

*Integration and interoperability.*

That stable information is also important in the process of integrating different databases. It permits to discover correspondences between objects of different databases that are conceptually similar but have different designations and structures.

## 3 The \( F_2^{\text{Concept}} \) Model

The \( F_2^{\text{Concept}} \) model is a data model for databases that are intended to store information about the definition of concepts (as they exist in terminological knowledge bases) together with instances of these concepts (as in traditional databases). In database terminology this means to store objects of a class as well as a formal description of the concept corresponding to this class. \( F_2^{\text{Concept}} \) is basically a three levels model. Objects of these level represent classes of concepts, concepts, and concepts’ instances respectively, as shown in Figure 1.

![Figure 1 \( F_2^{\text{Concept}} \): Integration of classes’ intensions and extensions](image)

### 3.1 Concept-classes

Concept-classes are used to classify the concepts of the universe of discourse. Instances of a concept-class are concepts, i.e., a concept-class is a class of concepts. Concept-classes are hierarchically organised, with the predefined concept-class \( \text{CONCEPT} \) as top element. Each concept-class possesses characteristics which will be used to define the intension of its concepts. For example, the following definition

```
concept-class Furniture < CONCEPT
characteristics
  shape : String
  size : String
```
A concept is an instance of a concept-class, its intension is defined by the values of its characteristics. If M is a concept-class with characteristics $c_1, c_2, \ldots, c_k$, the intension of a concept $C$ of $M$ is determined by the values $v_1, v_2, \ldots, v_k$ it takes for $c_1, c_2, \ldots, c_k$ respectively. For instance, the concept armchair of Seat can be defined as

```
concept armchair instance of Seat is
  use : "to sit down",
  back : [number : 1],
  armrest : [number : 2],
```

This example shows that some characteristics (like shape) are not given any value, because they are not relevant to distinguish this concept from others; or that their value is partially defined, for instance, only the number of backs (which must be 1) is relevant here, but not their height or shape.

Describing the intension of a concept by a set of characteristic-value pairs, instead of natural language sentences, is essential for automating the treatment of concepts. It enables, for instance, to select or compare concepts on the basis of their characteristics values, thus improving the semantic accuracy of these operations (compared to string matching based operations).

In order to represent objects belonging to the extension of concepts, i.e. instances of concepts, each concept is associated with a database class. This class possesses attributes which are used to describe each individual object. The following example shows a complete definition of the concept of armchair, with the attributes of its associated class.

```
concept armchair instance of Seat is
  back : [number : 1],
  armrest : [number : 2],
  use : "to sit down"
attributes
  serial_number : Int,
  colors : set (String),
  price : Real
```
It is important to notice that the characteristics values of a concept remain the same (e.g. “all the armchairs have two armrests”) for each object, while the attributes may and generally have different values for each object.

Attributes bear incidental properties of objects, their values have no relationship with the fact that the object belongs or not to that concept. For instance, the fact that a particular armchair x is white and that it costs $56.00 is not a consequence of the fact that it is an armchair. On the contrary, characteristics are essential properties, they are necessary and sufficient to determine to which concept an object belongs. For instance, if x is an armchair it necessarily has one back and two arms.

3.3 Objects

An object represents an instance of a concept, it is a member of classes associated with the concept. Each object is characterized by the values it takes for the different attributes of the class. Objects are the basic storage unit of the database, they can be created, updated, and removed from their class.

The following expression shows the creation of an object a which is an instance of concept armchair.

\[
a := \text{create armchair[serial_number: 3456, colors: \{white\}, price: 56.00]}
\]

3.4 Subconcepts

A subconcept is a concept that is derived by specialization from a more general concept (called its superconcept). Its characteristics’ values are either specific or are inherited from its superconcept.

The following expression defines concept wingchair as a subconcept of armchair

\[
\text{subconcept wingchair \subset armchair}
\]

\[
\text{attributes}
\]

\[
\text{armrest: \{number : 2, shape : “scrolled”\}}
\]

\[
\text{style : String}
\]

In this case wingchair inherits from armchair the values of characteristics use and back and has its own value for characteristic armrest.

The class associated with a subconcept is a subclass of its superconcept’s class, it inherits the attributes of its superclass and may have its own attributes. In the previous example, wingchair class has attributes serial_number, colours, price (inherited), and style (own).

An instance w of wingchair can be created as follows:

\[
w := \text{create wingchair [serial_number:7689, colors: \{black, green\}, style: “Louis XV” ]}
\]

The object w so created is a member of both wingchair and armchair classes.
4 Querying F2^Concept Databases

With such databases, queries and manipulations may be made at the extension level as well as at the intension level.

Extensional queries are used to find objects according to their attribute values. For instance, the following query retrieves all the chairs having a price lower than 45.00:

```
select chair [price < 45.00]
```

This is the kind of query one can usually express in standard database management systems.

Intensional queries can be used either to find the characteristics of a given concept, this corresponds to everyday dictionary use, or to find which concepts are described by given characteristics. For example, the following query retrieves all the concepts corresponding to seats with two scrolled armrests:

```
select Seat [armrest: [number: 2, shape: Scrolled]]
```

Combining the two levels enables to pose queries such as “what are the serial numbers and prices of all white seats with two scrolled armrests”. To answer the query we first select from the Seat concept-class the concept which is a furniture with two scrolled armrests, then we extract serial numbers and prices from the class associated with the selected concept.

```
the_concept := select Seat [armrest: [number: 2, shape: Scrolled]]

the_instances := select the_concept [colors: {white}]

display the_instances [serial_number, price]
```

5 Multivalued Characteristics and Inheritance in F2^Concept

In ordinary database modelling, multivalued attributes are used to represent a set of properties that are simultaneously true for an object. For instance, the value \{geneva, tokyo, paris, london\} given to an attribute visited_cities for an object of class person means that this person has visited geneva, tokyo, paris, and london. Thus there is generally an implicit AND connector behind such multivalued attributes.

When working on concepts representation with multivalued characteristics, it becomes necessary to qualify each value with an appropriate logical connector. For instance, the value of the armrest characteristic of the chair concept is [number: OR(0,2)], meaning that any particular chair has either zero or two armrests. This implies that if one wants to represent the actual number of armrests of a particular chair c, it is necessary to have an attribute number_of_armrests whose value can be set independently for every chair, as long as it equal to either 0 or 2.

```
concept chair instance of Seat is
  back : [number : 1]
  armrest : [number : OR(0,2)]
...
attributes
```
number_of_armrests : Int

So, in the OR case, the characteristic may give rise to an attribute defined at the concept level to store the different values taken by individual instances of the concept. The domain of values of this attribute is given by the set of values of the corresponding characteristic.

AND-valued characteristics determine a set of values that each instance of the concept must simultaneously have. For instance, a cot has two particularities: it is collapsible and it can be swung. These properties can be represented by the value AND("collapsible", "can be swung") for the particularity characteristic.

As opposed to OR-valued characteristics, AND-valued characteristics define fixed values taken by all the instances and hence do not require a concept level attribute.

5.1 Inheritance in F2

Usually subconcepts inherit characteristics’ values of their superconcept. This means that when no value is specified for a characteristic of the subconcept the superconcept’s value is assumed. For instance, wingchair inherits the value “to sit down” from characteristic use of armchair.

On the contrary, if the characteristic is given a value for the subconcept, this value supersedes the inherited value. This is what happens with armrest whose value is explicitly given for wingchair.

5.2 Inheritance and Contradictions

Usually, the value given for the subconcept is more specific, as shown in the previous example. This implies that the extension of a subconcept should be included in its superconcept’s extension. However, it may happen that a subconcept takes a value that is in contradiction with this extension’s pure specialization semantic. For example, the concept stool has back = [number: 0] while back stool, which is a subconcept of stool, has back = [number: 1].

This situation does not necessarily reflect a design flaw in the concepts’ structure but simply reveals the existence of exceptions to general rules. Otherwise stated, the strict adherence to extensions’ inclusion semantics would make the concepts’ structure much more complex and less natural. In our example it would imply that a back stool is not a stool.

5.3 Inheritance of Multivalued Characteristics

When dealing with multivalued characteristics the simple inheritance strategy cannot be applied any more. In certain cases it may be more interesting, and correct, to take the union of the proper and inherited values. For instance, if chair has characteristic parts = [{type: "back", number: 1}, {type: "legs"}, {type: "seat", number: 1}], armchair inherits these parts but it has its own part: {type: "arm", number: 2}. In this case, although a value is given for parts in armchair this does not mean that parts should not be inherited from chair. This kind of inheritance could be named inheritance with union semantics.

Let now go into a more complex case where union semantics is not sufficient. A “voltaire” is defined as "an armchair with a low seat". Thus the concept voltaire must inherit parts = [legs, seat, armrests] from armchair but not seat since it is redefined with a more specific value [position: low].
In this case we have an union semantics with priority on the most specific value when two complex values are the same on a given subcharacteristic (type = "back").

concept voltaire instance of Seat subconcept of chair characteristics
parts : [type : "set", number : 1, position : "low"]

Another case arises when a value is given as a negation: v = NOT{a}, if v were defined at the superconcept level as v = OR{a, b, c}, the allowed values for v at the subconcept level would be: {b, c}.

6 Implementation

6.1 Structure of the F2 Model

The F2 model is an object-oriented data model based on the notions of object, class, attribute, and generalisation/specialisation.

Objects. In the F2 model an information is represented by an object. Each object has an object identifier (oid) that permits to distinguish it from other objects. An object also has a value whose nature depends on its class. Contrary to its oid, the value of an object may change during the life of that object.

Classes. The term class in F2 designate both a type of objects and a set of objects of that type. A class determines the type of its object by fixing the set of values they may take and the operations allowed on those objects.

Atomic Classes. Atomic classes are elementary classes of the F2 model. In an atomic class objects are identified by their value, i.e., they are self-identifying. For instance, in class String an object’s value is a string of characters and this same string is used to identify the object.

Attributes and Tuple-classes. An attribute R with cardinality (min... max) from a class A to a class B is a function that associates to each object o of A a list R(o) of at least min and at the most max objects of B or the unknown value (denoted by ‘?’). An attribute with cardinality (1...1) is called monovalued attribute, in this case the list R(o) will be identified with the only object of B it contains. A class C with attributes R₁,..., Rₙ is called tuple-class. The value of an object o of C is the tuple [R₁(o),..., Rₙ(o)].

Specialisation. A specialisation of a class A is a new class B, called a subclass of A. Objects of B must also belong to A. The set of attributes of B is formed of all the attributes of A (inherited attributes) to which may be added specific attributes. In other words, the extension of A contains the extension of B while the intension of A is contained in the intension of B.

The fact that an object is a member of a subclass is determined by a predicate applied on the objects of its superclass. The subclass’ extension is composed of all the objects of the superclass that satisfy the predicate and only those ones.

class Furniture (use : String, style : String, category: String)

subclass Seat of Furniture when category = ‘seat’
(back : Back)
In this example, each object of the class Seat is an object of the class Furniture which has a value equal to 'seat' for the attribute category. An object of the class Seat has the attributes use and style inherited from Furniture and the attribute back which is specific to Seat.

![Diagram](image)

**Figure 2** Classes, attributes and specialisation in F2

6.2 **The F2 Dictionary (Metalevel)**

A F2 database structure is formed of a set of classes, attributes and specialisation links. Thus, it is possible to represent this structure with an F2 structure which constitutes the metalevel or metaschema of F2. Figure 3 shows a part of the F2 metaschema.

![Diagram](image)

**Figure 3** Fragment of the F2 dictionary

As there is no structural differences between the schema and the metaschema, objects of the metaschema are manipulated in the same way and by the same procedures as schema classes. The difference between data and metadata is purely behavioural; primitive methods Create, Modify and Delete of metaclasses act not only on the metaobjects of metaclasses but also on objects described by the metaobjects.

For instance, the Create method of the metaclass TupleClass (see Figure 3) has the effect of (1) creating an object o of Tupleclass, with its own oid and values for its attributes and (2) creating an
empty class whose name is `className(o)`. Conversely, the `Delete` method applied to an object `o` of `TupleClass` must (1) delete all the objects of class `o` and (2) delete the object `o` itself.

As shown in section 6, the homogeneity of data and metadata in F2 presents many qualities which facilitated the definition of the F2\textsuperscript{Concept} model:

- querying and manipulating data and metadata with the same language.
- possibility to make the model evolve continuously by successive modifications of the metalevel.

6.3 Implementation Technique

The F2 dictionary have been extended to take into account informations on the intension of classes (concepts). Since F2 is meta-circular it is possible to represent any n-levels model with the two levels (classes and objects) of F2. Thus the three levels of F2\textsuperscript{Concept} have been implemented by the two levels of F2. Concept-classes and concepts of the F2\textsuperscript{Concept} model are implemented by classes of F2 and F2\textsuperscript{Concept} instances correspond to F2 basic objects (i.e., objects that are not themselves classes). The implementation consisted in

- adding a subclass `CONCEPT` of class `TupleClass` in the F2 dictionary
- extending the F2 data definition language compiler [10] to integrate the definition of concept-classes, concepts and subconcepts.

Figure 4 Representation of F2\textsuperscript{Concept} in the F2 model

In Figure 4 one can see how concept-classes (Furniture and Seat) and concepts (armchair and wingchair) of F2\textsuperscript{Concept} are mapped to F2 classes.
The concept wingchair is a subconcept of armchair consequently the extension class Wingchair is a subclass of the extension class Armchair.

The characteristics of concept-classes Furniture and Seat are implemented by attributes of the corresponding Furniture and Seat classes. For instance Furniture is implemented as the following F2 class:

```
subclass Furniture of CONCEPT
    (shape : String,
     height : String,
     use : String)
```

Since a concept-class M is a subclass of CLASS, it inherits attributes className and classType and its instances, which represent concepts, are themselves classes. The definition of a new concept corresponds to an object creation in the class corresponding to a concept-class. For instance, the following expression creates the concept cupboard and defines its characteristic values.

```
create Furniture [className: "cupboard", classType: "tupleClass", isSubClass: 0,
    shape: "rectangular", height: "high", use: "to store objects"]
```

Attributes are then added to the new class, by creating new objects of class ATTRIBUTE. For example, the following expression adds attribute price to class cupboard.

```
create ATTRIBUTE [fromClass: "cupboard", attrName: "price", toClass: "Real"]
```

7 Related Works

Concept-classes and concepts of F2Concept could seem very close to metaclasses and classes of Smalltalk-80 [8]. Smalltalk-80 is based on a 3-levels architecture:

- metaclasses which define the behaviour of classes: mainly instantiation methods and initialisations specific to each class.
- classes which define the representation and behaviour of objects
- instances which are the objects manipulated by programs.

Smalltalk classes are defined by class and instance variables, and methods. Class variables are similar to concepts characteristics since their value is constant over all the object of the class (in fact class variables are instance variables of the metaclass). The main difference between F2Concept and Smalltalk lies in the structure of concept-classes compared to metaclasses. Concept-classes, their hierarchical structure, and their characteristics are user-defined, i.e., they are available to the database designer to structure his knowledge. On the contrary, Smalltalk metaclasses are internal implementation mechanisms, a metaclass has only one instance and inheritance links between metaclasses are imposed by the system.

Most of these Smalltalk-80 limitations disappear in ObjVLisp [5] which propose a unification of metaclass, class and object concepts. But ObjVLisp is a programming language, not a database system. It does not support persistent objects nor ad-hoc querying nor schema evolution.[1]
8 Conclusion

In this paper, we have presented the F2Concept model which permits to include in the same database the intension and the extension of object classes. The intension of a class is defined by a set of (characteristic - value) pairs that represent the intension of the underlying real-world concept. This allows to retain knowledge acquired during the modelling process.

The availability of both intensional and extensional information extends the usability of a database, it provides features to query and manipulate concepts that lead to a better understanding of the database contents.

The F2Concept implementation is based on the F2 database management system, it makes a straightforward use of the meta-circular architecture of F2 and uses the same query and manipulation tools.

Bibliography
