The Extended MdF Model

Ulrik Petersen

September 22, 2003

Contents

1 Introduction

In this article, I present the Extended MdF model, or the "EMdF model". It was originally proposed in [2]. The EMdF model builds on the MdF model as proposed by Crist-Jan Doedens in his 1994 PhD thesis [1]. For a summary of the MdF model, see [3].

The EMdF model arose out of my efforts in implementing the MdF model. I realized that the MdF model in itself is too abstract to be directly implementable. Thus I came up with a number of concepts and distinctions which make it easier to talk about an implementation. The EMdF model contains these concepts. Thus the EMdF model does little beyond adding some useful concepts to the MdF model. It limits the MdF model in only one respect, namely in restricting the codomains of features. This was necessary in order to be able to implement the MdF model in practice.

2 Sequences

In an EMdF implementation, a big part of keeping the internal data integrity depends on IDs. For example, objects have ids ("object id_ds"), object types have ids ("object type_ids"), etc. In the EMdF model, these IDs are drawn from two sequences of integers. One sequence is for generating object id_ds, and the other sequence is for all other kinds of IDs. Thus, in an EMdF database, these two sequences are present:

- SEQUENCE_OBJECT_ID_DS
- SEQUENCE_OTHER_IDS

These two sequences provide two separate "namespaces" for generating IDs.

3 Monads

3.1 Introduction

The EMdF model has a concept of "largest possible monad_m". This is called "MAX_MONAD". This is distinct from the monad "max_m," which is the largest monad present in any object in a particular database.

3.2 MAX_MONAD

In order to be implementable, the EMdF model introduces the concept of "largest possible monad_m". This is an integer called "MAX_MONAD". In any given implementation, this integer can be chosen arbitrarily large, only limited by what the underlying implementation-primitives allow.

3.3 max_m

Each database has a monad called max_m. It is the largest monad in the database in the sense that it is the last monad of the object(s) in the database that extend(s) farthest upwards into the monad sequence. Thus max_m is a particular monad in a particular database which is the largest monad in use.

3.4 min_m

By anaology, each database has a monad called min_m. It is the smallest monad in the database in that it is the first monad of the object(s) in the database that extend(s) farthest downwards in the monad sequence. Thus min_m is a particular monad in a particular database which is the smallest monad in use.

4 Objects

4.1 Introduction

The EMdF model makes a distinction between "objects in the abstract", called "object_ms", and "objects in an implementation", called "object_dms". The first two sections deal with this distinction.

In the EMdF model, objects are no longer unique in their monads. This has a number of ramifications for object ids. The third section deals with this issue.

In the last two sections, some notation is introduced for accessing the monad information in an object.

4.2 Object_ms

In order to be able to distinguish between "objects in the abstract" and "objects in an implementation", I have renamed the old notion of objects to "object_m". Thus an object_m is exactly the same as what Doedens calls an "object".

4.3 Object_dms

An object_dm is an object_m that has been concretely instantiated in a concrete EMdF database.

The special object types all_m, pow_m, and any_m, do not have any object_dms in the database, since these object types are not meant for storing in an MdF database.

4.4 object ids

4.4.1 Objects are not unique in their monads

The MdF model stipulates that,

> "No two objects of the same type may consist of the same set of monads. The reason for this restriction is that it allows us a simple and clear criterion for what different objects are." ([1, page 59])

In the EMdF model, this restriction has been done away with. Instead, each object is given a unique ID, its "object id_d." Doing away with this restriction also has ramifications for object id_ms, and for the linear ordering of objects per type. Object ordinals and object id_os are not changed, however.

4.4.2 object id_d

An object id_d is an integer which functions as a unique object identifier. Only object_dms have object id_ds. Objects of the special object types, all_m, pow_m, and any_m, do not have an object id_d.

Notation: For the object_dm O, $O.id$ denotes its object id_d.

4.4.3 object id_m

The Object id_m's are not redefined in the EMdF model. However, it is necessary to realize that, since objects are no longer unique in their monads, the object id_m need not be an id for any other object types than the special object types all_m, any_m, and pow_m. Thus object id_m's should only be used for these types. For application-specific object types, the concept of object id_m has been supplanted by the concept of object id_d.

4.4.4 linear ordering of objects per types

The concept of linear ordering of objects per type has had to be redifined slightly in the EMdF model. The reason is that the objects_dm's are no longer unique in their monads. Thus the concept of linear ordering per type is now defined as follows.

Take an object type T and two objects of type T, O_1 and O_2:

1. If O_1 and O_2 do not have the same monads, then linear ordering is decided as in the MdF model.

2. If O_1 and O_2 have the same monads, then $O_1 <_T O_2$ iff $O_1.id < O_2.id$.

4.4.5 object ordinal, object id_o

The concepts of object ordinal and of object id_o from the MdF model are not changed in the EMdF model. They are still based on the linear ordering of objects per type.

4.5 O.monads()

In the EMdF model, each object (both object_dms and object_ms) have a function, **monads**, which returns the set of monad_ms making up the object.

Notation: The function is denoted "O.monads()" for the object O.

4.6 O.first(), O.last()

In the EMdF model, each object (both object_dms and object_ms) have two functions, **first** and **last**, which return the left border and the right border respectively.

Notation: These functions are denoted "O.first()" and "O.last()" for the object O.

5 Object types

The only addition that the EMdF model makes to the MdF model regarding object types is that each object type is identified in the database by a unique integer, the "**object type_id**". This id is drawn from the sequence "SEQUENCE_OTHER_IDS", see section 2 on page 2.

6 Features

The only addition that the EMdF model makes to the MdF model regarding features is that each feature is identified in the database by a unique integer, the "**feature_id**". This id is drawn from the sequence "SEQUENCE_OTHER_IDS", see section 2 on page 2.

7 Enumerations

The EMdF model explicitly includes the type "enumeration" in the set of codomains that a feature can take on. An enumeration is a named set of pairs (identifier, value), where the identifier "stands for" the value. Thus, for example, one could have the following enumeration:

```
psp_t = { psp_NotAppliccable = -1, verb = 1, noun, proper_noun,
          adverb, preposition, conjunction, personal_pronoun,
          demonstrative_pronoun, interrogative_pronoun,
          interjection, negative_particle, interrogative,
          adjective }
```

This enumeration would be called “psp_t”, and its constant “noun” would have the value “2”, while its constant “conjunction” would have the value “6”.

Each enumeration has an “**enum_id**”. This is a unique integer drawn from the sequence “SEQUENCE_OTHER_IDS”, see section 2 on page 2.

Each enumeration always has one enumeration constant which is the default. This default is used when the user creates an object without specifying the value of a feature whose codomain is an enumeration. In the absence of a user-specified value, the default constant is used.

8 Codomains of features

In order to be implementable in practice, the EMdF model puts restrictions on what the codomains of features can be. In particular, only the following four are allowed:

1. integers
2. strings
3. object id_ds
4. enumerations

9 Inst(T,U)

In the EMdF model, a big part of the implementation of the MQL query language relies on a special function, inst(T,U). This function returns, for a given object type T, the object id_ds of all the objects which are part_of the arbitrary set of monad_ms U.

10 Conclusion

I have presented the Extended MdF, or EMdF, model. The EMdF model has added to the MdF model a number of concepts that are useful in implementing the MdF model. The EMdF model mostly adds to the MdF model, and in only one case does it restrict the MdF model, namely in what the codomains of features can be. The EMdF model is closer to an implementation than the MdF model by itself. This is its main benefit.

References

[1] Doedens, Crist-Jan [Christianus Franciscus Joannes]. *Text Databases. One Database Model and Several Retrieval Languages.* Language and Computers, Number 14. Amsterdam and Atlanta, GA: Editions Rodopi Amsterdam, 1994. ISBN: 90-5183-729-1.

[2] Petersen, Ulrik. *The Extended MdF Model*. Unpublished B.Sc. thesis from the University of Aarhus, Denmark. November 30, 1999. Electronic copies available upon request from the author at `ulrikp@users.sourceforge.net`.

[3] Petersen, Ulrik. *The Standard MdF model*. Unpublished article. 2002. Electronic copies available from http://emdros.org/docs.html