The Standard MdF Model

Ulrik Petersen

December 3, 2002

Contents

1 Introduction

The MdF model was developed by Crist-Jan Doedens in his 1994 PhD dissertation. It is a database model which is exceptionally well suited to storing linguistic analyzes of text. The MdF model gives a high-level view of text databases, where a text database is viewed as text plus information about that text. The MdF model is mathematically clean, simple, intuitive, and elegant, which makes it well suited to conceptualization of solutions to problems which can be solved by a text database.

This article gives an introduction to the standard MdF model. The introduction is based heavily on Chapters 2 and 3 in Doedens' book, and follows the same structure.

The bibliographic information for Doedens' PhD dissertation is:

> 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. Extent: xii + 314 pages. ISBN: 90-5183-729-1.

2 On text-databases generally

2.1 Text-dominated databases, expounded text

Two concepts are a key in understanding the state of the art in text databases. I here give two quotes from Doedens:

Text-dominated databases are "collections of data, predominantly composed of characters, in which we can perceive structure" (p. 18). This is the current viewpoint on databases of text.

Expounded text is "An interpreted text, i.e. a combination of text and information about this text, stored in a computer, and structured for easy update and access" (p. 19). The MdF model embodies the idea of expounded text.

2.2   What is a database model?

Doedens defines a **data model** or **database model** as

> "a toolbox of concepts which can be used to describe the handling by the computer of data in certain domain(s). The concepts should allow easy formulation by humans of the structuring and handling of the data in the domain(s). The concepts can be grouped as follows:
>
> - The data structures supported by the model
> - The access language. This language, or set of languages should allow the definition of the structure and types of the data and allow creation, insertion, change, deletion and retrieval of the data." (p. 23)

The MdF model is not a full database model, in that it does not specify an access language, but only the data structures supported by the model. This, a database model without its access language component, is what Doedens calls a **static database model**.

2.3   Demands on a text database model

Doedens says:

> "The fundamental requirement for a text database model is that it should be able to support the structural description of a text and its associated annotations." (p. 25)

He then lists thirteen demands which he perceives should be set on text database models:

D1. Objects: We should be able to identify separate parts of the text.

This can be realized with instances of the concept of objects.

D2. Objects are unique: Each object should be uniquely identifiable.

Otherwise, we may not know what we are talking about.

D3. Objects are independent: Each object in the database should exist without direct reference to other objects.

The advantage of having this is that we can have the best of two worlds: Independence and dependence, isolation and referentiality. The independence comes from D3 being met, and the dependence can arise through D4-D7 being met.

D4. Object types: We need 'object types': we should be able to assign the same generic name to like objects.

For example, we would like to be able to identify certain parts of a book stored in a text database as "paragraphs", other parts of a book as "chapters", and other parts as "pages".

D5. Multiple hierarchies: It should be possible to have different hierarchies of types.

For example, we might have a hierarchy of types which form a textual hierarchy ('character', 'word', 'line', 'page', 'book'), and a hierarchy of types which form a logical hierarchy ('word', 'sentence', 'paragraph', 'chapter').

D6. Hierarchies can share types:

See D5 for a useful example.

D7. Object features: We should be able to assign features and values for these features to objects.

D8. Accommodation for variations in the surface text: E.g. variations in spelling should be attributable to the same words in the surface text.

D9. Overlapping objects: We need objects of the same type to be able to overlap.

For example, to describe recursivity in linguistic phenomena. Take for example the string of words, 'A noun phrase and a conjunction phrase': How do we analyze this? As two noun phrases ("A noun phrase" and "a conjunction phrase") conjoined by a conjunction ("and"), or as one compound noun phrase (the whole thing)? We want to be able to represent both choices. Overlapping objects of the same type can help us in doing this.

D10. Gaps: We need objects with 'gaps'.

For example to describe clauses which are discontiguous, as in "John, who was having a cow, freaked out.", where "John ... freaked out" is a discontiguous clause.

So far, the demands have dealt with the data-structures to support the model. The following demands reflect the demands which a full model (one in which there is also an access language) must meet:

D11. Type language: We need a type language in which we can specify object types and the types of their features.

D12. Data language: We need a strongly typed data language in which we can specify the creation, insertion, change, deletion, and retrieval of data.

D13. Structural relations between types: It should be possible to specify standard structural relations between objects of different types.

None of the text database models available today satisfies these 13 demands. The MdF model satisfies D1-D10. In contrast, SGML does not, and neither do any of its derivatives, e.g.. XML.

3 Gentle introduction to the MdF model

3.1 Key concepts

The MdF model has four key concepts:

1. Monads: These are the basic building blocks of the database. They are simply integers. Monads are ordered relative to each other, such that a string of monads emerges.

2. Objects: Objects are made of monads. An object is a set of monads.

3. Object types: Objects are grouped in types. An object type determines what features an object has.

4. Features: A feature is a function on objects. A feature takes an object as its argument and returns some value usually based on that object.

3.2 An example

An example of an MdF database can be seen in figure 1 on the following page. It has five object types: Word, Phrase, Clause_atom, Clause, and Sentence. Object type Word has two features: surface, and part_of_speech. Object type Phrase has one feature: phrase_type. Object types Clause_atom, Clause, and Sentence have no features. The first Phrase object consists of the set of monads $\{1, 2\}$, the third of the set of monads $\{4\}$, and the fourth of the set of monads $\{5, 6, 7\}$. The first Clause object consists of the monads $\{1, 2, 8, 9\}$. Note that the first Word object *consists of* the set of monads $\{1\}$, and *is* not monad 1, and likewise with the rest of the Word objects.

4 Monads

4.1 General

The MdF model was developed for text databases, but can be used for storing anything that is linear in nature, e.g., DNA sequences. The backbone of an MdF database is a linear string of minimal, indivisible elements, called monads. The precise nature of the entities which the monads represent is of no importance to the model. The only thing that matters is the relative ordering of the monads.

A monad is simply an integer. It represents the rank number in the string of monads, starting from 1. Since monads are integers, we can apply all the usual relational and arithmetic operators to them. For example, if we have monads a and b, we can test whether $a < b$, $a = b$, or $a > b$. Or we can test whether $a + 3 = b$.

	1	2	3	4	5	6	7	8	9
Word	1	2	3	4	5	6	7	8	9
surface	The	door,	which	opened	towards	the	East,	was	blue.
part_of_speech	def.art.	noun	rel.pron.	verb	prep.	def.art.	noun	verb	adject.
Phrase	1		2	3		5		6	7
phrase_type	NP		NP	VP		NP		VP	AP
Phrase					4				
phrase_type					PP				
Clause_atom	1		2					3	
Clause	1		2					1	
Sentence	1								

Figure 1: MdF example

4.2 Application of monads to text flows

In everyday thinking about text, any given text is conceptualized as a one-dimensional string. The text may be laid out on a two-dimensional medium, e.g., paper, but there is still only one string. The elements in this string have an ordering called the **reading order**. For example, the English reading-order is left-to-right, line-by-line, downwards. The Arabic reading-order is different, and the Japanese reading-order is different yet again. From the point of view of the MdF model, the reading-order is dictated by the monads.

The MdF model has no concept of parallel or unrelated text flows, e.g., footnotes or margin notes vs. the main text. Fortunately, this is not a problem, since the object types allow us to direct attention to any text flow at will. For example, the implementer of an MdF database might decide to intertwine the main text and the footnotes. The text is then called "footnote" or "main text" simply by defining appropriate object types and defining its objects appropriately. For an MdF database with several books, it might be more intuitive to place each book after the other, defining an object type called "book" and defining its objects appropriately.

5 Objects, object types

An MdF object is a set of monads. The monads in this set need not be contiguous. This is a great advantage, since it allows us to have objects with gaps.

Objects are grouped in types. For a sample list of object types which might occur in an MdF database, look at table 1.

It is an important characteristic of objects that no two objects of the same object

- testament

- book

- chapter

- verse

- word

- phrase

- clause

- sentence

Table 1: Sample Object types

type may consist of the same monads, i.e., there must not be two objects of the same object type for which there is set equality between the two sets of monads. The reason for this restriction is that it gives us a simple and clear criterion for what different objects are: An object is unique in its set of monads. On the other hand, two objects of different types may consist of the same monads. And two objects of the same type may share monads, so long as their sets of monads are not identical.

Note that objects do not consist of other objects. Instead, objects consist of monads. This is a great advantage, since it allows us to specify multiple hierarchies.

The fact that objects are sets of monads gives rise to a rich set of descriptive terms, which can all be formulated in terms of the basic operators on sets, such as set equality, the subset relation, set intersection, set union, and the "member of" relation. Also, the fact that there is an ordering, $<$, on the monads gives rise to a number of interesting properties. We shall return to these properties in section 7 when discussing concepts related to the MdF model.

6 Features

A feature is a function taking one argument: An object. The object type of an object determines what features the object has. The domain (type of argument) of a feature function is the set of objects of a given object type. The codomain (type of return value) of a feature function can be anything: The MdF model puts no restrictions on the codomain. This allows the implementor to implement anything at all which he or she might feel should be a feature. For a list of sample features which might be present in an MdF database, look at table 2.

In particular, the codomain of a feature can be another function taking other argu-

Object type	Feature name	Feature function
book	book_name	maps a book object to its name
book	book_number	maps a book object to its number
chapter	chapter_number	maps a chapter to its number
word	lemma	maps a word to its lemma
word	Friberg_tag	maps a word to its Friberg grammar tag
word	part_of_speech	maps a word to its part of speech
word	case	maps a word to its case (if applicable)
word	gender	maps a word to its gender (if applicable)
word	number	maps a word to its number (if applicable)
phrase	phrase_type	maps a phrase to its type (e.g., VP, NP)
phrase	arthricity	maps a phrase to its status as being arthrous
phrase	function	maps a phrase to its function (e.g., Subj, Obj)
phrase	determinedness	maps a phrase to its status as determined or not
clause	clause_type	maps a clause to its clause type
clause	function	maps a clause to its function

Table 2: Sample features

ments, thereby in effect producing a feature with arguments.

Features can be partial functions, i.e., there can be objects for which a feature's value is not defined.

There is no such thing as a "genuine" feature. All features are considered to have equal status from the point of view of the MdF database. For example, the feature returning the surface of a word has the same status as a feature that returns the sum of the monads in its argument object.

7 Extending the basic framework

7.1 Introduction

This section is almost a duplicate of section 3.6 in Doedens' book, but leaving out some explanations, examples, justifications, and a few concepts which are irrelevant to the understanding of EMdF.

Throughout this section, we will refer to figure 1 on page 6.

7.2 Some special types: all_m, any_m, pow_m

Any MdF database implicitly defines the object types all_m, any_m, and pow_m. The reason for this is that it is convenient when talking about MdF databases.

All_m is the object type which has just one object: The one consisting of all monads in the database.

Any_m is the object type which has for each monad one object consisting of that monad.

Pow_m is the object type which has a member for each member of the power-set of the monads.

None of these three special object types has any application-specific features.

7.3   Linear ordering of objects per type

The linear ordering of objects per type is based on a 'lexicographic' ordering of the monads. This is done using the smallest monad as sort key. If this gives a tie, we go on to the next smallest, the next smallest, and so on. If at any point object O_2 has a monad which object O_1 does not, then object O_1 is stipulated to have the smaller ordinal of the two. For any object type T, the ordering relation within the type is denoted $<_T$. Thus, for example, if we have an object type T , then these relations hold:

$\{1\} <_T \{1,2\}$
$\{1,3\} <_T \{1,2\}$
$\{1,3,4,5\} <_T \{1,3,4,5,6\}$
$\{1,2,3,4,5,6,7\} <_T \{2\}$
$\{3,4\} <_T \{5,6\}$

7.4   Ordinal of an object, object id

The linear ordering of objects per type can be used to assign ordinals to objects of a given type. We just assign the ordinal 1 to the first object in the linear ordering, and go on from there.

This means that objects can be identified by their type plus their ordinal. Alternatively, since objects are unique in their monads, they can be identified by their type plus their monads. Both ways of identifying an object can be useful.

When identifying objects by their type plus their ordinal, the resulting id is called an **object id_o**. For example, objects of type T might be called T-1, T-2, T-45, etc.

When identifying objects by their type plus their monads, the resulting id is called an **object id_m**. For example, objects of type T might be called T-{1,2,3,5}, T-{4}, etc. Object id_m's are most useful for the three special object types, all_m, any_m, and pow_m, since they are specifically defined in terms of monads. There is also a practical reason for the usefulness of object id_m's: If we have a database of 138,019[1] monads, the object id_o's of the pow_m object type range from 1 to $2^{138,019}$, which would take on the order of $138,019$ bits to implement - something which is clearly intractable.

---

[1] The number of words in the Greek New Testament as published in the Nestle-Aland 27th edition is 138,019.

7.5 Part_of, overlap

7.5.1 Part_of

The subset relation gives rise to a relation between objects which is quite crucial in building hierarchies. Take two objects, O_1 and O_2.

O_1 part_of O_2 $\iff$ $O_1 \subseteq O_2$

Thus if the monads of O_1 are all in the set of monads comprising O_2, then O_1 is part_of O_2.

In our example figure on page 6, Phrase-5 is part_of Phrase-4. Phrase-2 is part_of Clause_atom-2.

7.5.2 Overlap

Objects can share monads. The notion of overlap formalizes this idea. This notion is expressed in terms of the set intersection operator. If the intersection of the monads of two objects is non-empty, then they share monads, and are thus overlapping. Take two objects, O_1 and O_2.

O_1 overlaps with O_2 $\iff$ $O_1 \cap O_2 \neq \emptyset$

In our example figure on page 6, Phrase-5 overlaps with Phrase-4. Phrase-5 does not overlap with Phrase-6. Clause_atom_2 overlaps with Sentence-1.

Object types can also said to be overlapping or non-overlapping:

An **object type is overlapping** if and only if some of its objects overlap.

An **object type is non-overlapping** if and only if it is not overlapping.

In our example figure on page 6, only the Phrase object type is overlapping. The rest of the object types are non-overlapping.

It could be made part of the type-system of a full database model based on the MdF model that one could specify that a given object type was overlapping or non-overlapping, and, in the latter case, uphold this constraint automatically when attempting to add objects of this type.

7.6 Covered by and buildable from

In some applications, we want certain object types to form a hierarchy. For example, sentences may be formed from words, and paragraphs may be formed from sentences.

Since objects are made of monads, not other objects, we need some way of specifying hierarchies. This is done using the notions covered_by and buildable_from. These notions are, in the MdF model, only extensional in nature, i.e., only by inspection can we decide whether an object type is covered_by or buildable_from another. It could, however, be made part of the type system in a full database model based on the MdF model, such that constraints could be upheld when adding or deleting objects.

Despite their names, these two notions are not opposites of each other. Rather, buildable_from expresses the same as covered_by, only with an additional constraint.

7.6.1 Covered by

An **object type** T_{high} **is covered_by object type** T_{low} if and only if the union of all the monads in all the objects of the set of objects of T_{high} has set equality with the union of all the monads in all the objects of the set of objects of T_{low}, AND for all objects O_{high} of T_{high} there exists a set of objects S of T_{low} such that the monads of the union of all the objects in S are the same as the monads of O_{high}, AND for all objects O_{low} of T_{low}, it is the case that there exists exactly one object O_{hg} of T_{high} such that O_{low} part_of O_{hg}.

Note that the name 'covered_by' is slightly counter-intuitive: It is the larger that is covered by the smaller, not the smaller that has the larger as a canopy over it.

Doedens says in his book (p. 70) that covered_by induces a partial ordering on objects types. This is not true: As a counterexample, the following setup makes covered_by fail to be antisymmetric: Suppose that we have an MdF database with just three monads and just two object-types, T_1 and T_2, and suppose that both T_1 and T_2 have only one object: The one consisting of the monads $\{1, 2, 3\}$. Then T_1 is clearly covered_by T_2, and T_2 is clearly covered_by T_1, yet $T_1 \neq T_2$.

In our example figure on page 6, Phrase is covered_by Phrase and Word, Clause_atom is covered_by Clause_atom, Phrase and Word, and Clause is covered_by Clause, Clause_atom, Phrase, and Word. Sentence is covered_by everything.

7.6.2 Buildable from

We sometimes want to say something that is a little stronger than that two object types are in a covered_by relationship. We sometimes want to specify that the two object types are non-overlapping as well.

An **object type** T_{high} **is buildable_from object type** T_{low} if and only if T_{high} is covered_by T_{low}, AND both T_{high} and T_{low} are non-overlapping.

Doedens notes in his book that buildable_from induces a partial ordering on object types that are non-overlapping. This is, of course, not true, since covered_by is not a partial order.

In our example figure on page 6, the only object type that is not buildable_from something is Phrase, since it is overlapping.

7.7 Consecutive, gaps

7.7.1 Consecutive

Basically, two objects are consecutive if they follow each other in a neat row without any gaps in between. This is the heart and soul of text representation.

However, sometimes it is handy to exclude certain parts of the database from consideration. For example, in our example figure on page 6, we may wish to concentrate on "The door, ..., was blue" as embodied in Clause-1. We say that "was blue" is con-

secutive to "The door," with respect to the set of objects constituting Clause-1. We call Clause-1 the "Universe".

In order to formally define the notion of consecutive, we first need a definition of a range:

Definition: A range of monads is denoted by $a \mathrel{..} b$, and has the following meaning: If $b < a$, then the range is empty. If $b \geq a$, then the range denotes the set of monads starting at a and including all monads up to and including b.

For example, "1..3" denotes the set $\{1, 2, 3\}$, while "3..2" denotes $\emptyset$.

Definition: A set of n monads, M, where $n \geq 2$, is consecutive with respect to a set of monads, U, if, when ordering the elements of M according to '$<$', such that $m_1 < \ldots < m_n$, for all i, $1 \leq i \leq n-1$, it holds that $(m_i + 1 .. m_{i+1} - 1) \cap U = \emptyset$.

Definition: A set of n objects, S, where $n \geq 2$, is consecutive with respect to a set of monads, U, if the objects can be ordered as $O_1, \ldots, O_n$ such that for all i, $1 \leq i \leq n-1$, it holds either that $(O_i = \emptyset \vee O_{i+1} = \emptyset)$ or (i.e., exclusive or) the set consisting of the last monad of O_i and the first monad of O_{i+1} is consecutive with respect to U.

7.7.2 Gaps

As we have seen, objects need not consist of a contiguous string of monads. For example, in our example figure 1 on page 6, Clause-1 consists of "The door, was blue.". There is a gap in Clause-1 with respect to "The door, which opened towards the East, was blue.".

An object is said to have gaps if its monads are not consecutive. The monads of a gap in an object are not part_of the object. Furthermore, they are not outside of the object, i.e., all of the monads in a gap in an object O are both $\geq$ the first monad of O and $\leq$ the last monad of O. Gaps are always "maximal", i.e., we group as many monads as possible into a gap.

Definition: A gap in an object O relative to a set of monads U is a set of monads, H, such that:

1. $H \neq \emptyset$ AND

2. H is consecutive with respect to U AND

3. $\neg (H$ overlaps with $O)$ AND

4. $\forall h_i \in H . h_i \geq O.first() \wedge h_i \leq O.last()$ AND

5. $\neg \exists H'.H' \neq H \wedge H \subset H' \wedge H'$ gap in O relative to U (i.e., H is maximal)

where $O.first()$ and $O.last()$ refer to the first and last monad of O, respectively.

7.8 Border, separated, inside

7.8.1 Border

An object always has a first and a last monad (which is a consequence of the well-formedness axiom for natural numbers). The first monad of an object is called its **left border**, and the last monad of an object is called its **right border**. Together, the left border and the right border of an object constitute its **borders**. Two notions that can be defined in terms of the borders of objects are "separated" and "inside".

In our example figure on page 6, the left border of Phrase-4 is 5, and its right border is 7. The left border of Phrase-2 is the same as its right border, namely 3. The left border of Clause-1 is 1, and its right border is 9.

7.8.2 Separated

Two objects are separated if and only if the right border of one of the objects is < the left border of the other. Thus even if we add the gaps in the objects to the objects, they still do not overlap.

In our example figure on page 6, Word-3 and Phrase-3 are separated. Phrase-3 and Phrase-4 are separated. Clause-1 and Clause-2 are *not* separated.

7.8.3 Inside

An object O_1 is inside O_2 if and only if the left border of O_2 is $\leq$ the left border of O_1 and the right border of O_2 is $\geq$ the right border of O_1. Note that, even though an object is inside another object, they need not overlap, since objects may have gaps.

In our example figure on page 6, Phrase-5 is inside Phrase-4. Clause-2 is inside Clause-1. Word-4 is inside Clause-2.

8 Conclusion

In this article, we have presented, in condensed and abridged form, Doedens' work as it applies to demands on text databases and to the MdF model itself. Not much has been original in this article. We have touched upon text-dominated databases and expounded text, upon what constitutes a database model, and upon Doedens' thirteen demands on a text database model. We have given a gentle introduction to the MdF model, followed by in-depth discussions of the four key concepts of the MdF model. We have then defined a lot of useful concepts in relation to the four basic concepts in the MdF model.