Serdica
Bulgariacae mathematicae publicationes

Сердика
Българско математическо списание

MARS — MULTIMEDIA ARCHIVING AND RETRIEVAL SYSTEM

HELMUT EIRUND

This paper presents a document model for multimedia documents that is specially developed for office applications. It satisfies the requirements formulated for conceptual modelling of unformatted documents, supporting efficient retrieval and flexibility in dynamic application areas. The model and its implmentation is discussed in five steps of its evolution: (a) formulation of requirements, (b) (informal) description, (c) algebraic specification of the model, (d) implementation of the MARS system that supports the model and (e) design of an application in a clinical enviroment. The MARS project is currently underway at University of Oldenburg. First application is developed for the radiologic department of a district hospital.

1. Introduction. Documents are the basic information sources in various office applications, e. g. in management, engineering or clinical environment. Electronic documents are no longer restricted to alpha-numerical data or strict formatted forms. In this paper we refer to *documents* as information units in elecronic medium, whose structure is not necessarily formular. It consists of multimedia content units, like alphanumeric character sequences (text), graphics, bit-map images or audio annotations.

Examples of documents coming from several application fields are accounting forms (strong formatted), product offers (free formatted), radiograms (image with enclosed data fields), reports (complex) or memos (simple). With the variety of structural information that can be observed, two questions arise:

(i) **What structural information is to be described?** Documents are normally manipulated as one unit by the end-user, i. e. the document is a single *user object*. Heterogenous structural information can be observed that describes different views of the document, supporting different utilizations:
- a document represented in pages, blocks etc. enables the presentation of the document content on an output-medium
- partitioning of the document content in e. g. chapters, paragraphs, subparagraphs supports the intellectual comprehension of the document content
- naming and describing semantic components of a document (independent of the layout and the logic objects) serves for accessing documents by qualifying those components.

(ii **How is the structural information to be represented?** Splitting document units into several different relations (i. e. one document represented as a set of *model objects*) according to their structural properties, needs a lot of expansive joining operations to reconstruct the *user object* each time a document is manipulated by the user. Furthermore, giving structural information implicitly through a relational schema tends to be too unflexible to describe all occurences of structural relations in office documents.

The different utilizations presented above motivate three different views of document structure: (a) the *layout structure*, (b) the *syntactic structure* and (c) the *conceptual structure*.

All these views can be represented as trees, whose inner nodes include structural information and are described by more simple objects, their subnodes. Leaf nodes are associated to content portions. The layout and logic structure of the multimedia documents can be efficiently described within the ODA standard (Office Document Architecture). The Standard is specified in [10].

In this paper a model for the conceptual view is specified. The MARS Project (Multimedia Archiving and Retrieval System) at the University of Oldenburg follows this specification. It has been influenced by the modelling approach first presented in [3] and by the ESPRIT project MULTOS that makes use of conceptual structures for multimedia document retrieval [8], [4].

The syntactic structure of this paper is as follows: The problem of representing document semantics is discussed in the second chapter and the requirements of a document model are formulated on that basis. Furthermore, an informal view of the document model employed in MARS is given in Chapter 3. In Chapter 4 some fundamentals of a usual algebraic specification formalism are introduced very briefly (readers familiar with this specification method may skip the chapter). In Chapter 5 the specification of the model is given and the semantic specifications of some selected operations are discussed. The implementation of MARS that follows this specification is presented in Chapter 6 and Chapter 7 describes a test application currently developed within a clinical environment. Chapter 8 concludes with a summary.

2. Requirements for a Conceptual Document Model. In many papers requirements have been defined for designing office models or data models appropriate for office applications with different emphasis, e. g. [1] proposes an extension of the ER model for software engeneering; many suggestions are based on the NF^2-model [21], appropriate complex operations are given in [22]; semantic information of office documents is discussed in [20]; In [13] the requirements of office archiving systems and a classification of models have been given; involving knowledge bases for document representation has been proposed in many papers, e. g. in [9], partially implemented in [2]).

Concepts for document modelling and manipulation are influenced by at least three aspects:

· database theory with practicable relational calculus,
· information retrieval with its homogenous view of the data object "document" and appropriate access methods and
· object oriented modelling with the issue of representing complex user objects through model objects in a suitable way.

The MARS document model should meet the following requirements:

(a) to provide an *abstract view* of document semantics: to describe document semantics by naming conceptual units i. e. application relevant units, and specifying hierarchical relations between them, like aggregation or specialization.
(b) to associate values (content portions) to conceptual units and permits *fact retrieval* by qualifying documents through document values.
(c) to support a systematic and consistent process of modelling document semantics: typical semantic properties of documents are generalized to *document types* that serve as templates and support query formulation as well as evaluation.
(d) to allow *flexibility* of describing document semantics and types.

3. Informal Discussion of the MARS Model. Trees structures are well suited for representing document semantic structures. In the MARS model *document conceptual structures* are defined as trees. All those components of a document that are relevant to further processing are identified and named [11].

A conceptual structure is composed of *conceptual components* (also called *concepts*). It contains *basic* conceptual components as leaf nodes, and *complex* conceptual components constructed by subcomponents. With the concept identifiers a "natural" understanding can be associated (e. g. the concept named "offer" identifies the description of an offer in a business document). Thus we get an abstract view of the semantic units of the document. The association of content portions as values to concepts, as well as the valuation of properties like price of a product or medical examination results on a patient card can be expressed.

The construction mechanism may be either *aggregate* so to describe a concept by those concepts participating in its definition, or *specialize* in order to describe a concept as a whole. The next figure vasualizes[1] a document conceptual structure that can be observed in a clinical environment.

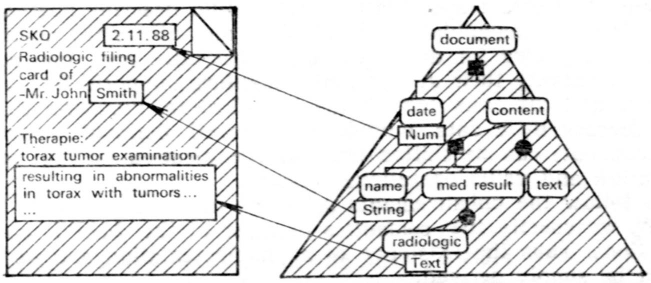

Fig. 1. A document with its conceptual structure

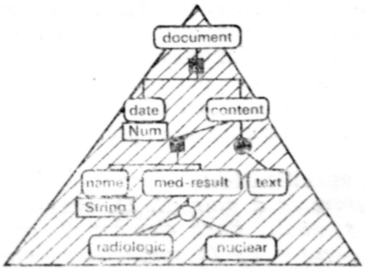

Fig. 2. Type conceptual structure

Classes of documents with a similar structure can be described by document types. A type conceptual structure is a generalization of the document conceptual structures associated to it. Thus, basic concepts in type conceptual structures are not further described by associated values but have value *domain specifications*. Furthermore, a variety of concepts specializing one concept can be formulated *(choice-specialization)*. Document conceptual structures are instances of the appropriate types. This enables a content modelling process of documents with similar semantics. Figure 2 visualizes[2] a possible type conceptual structure with the document structure of fig. 1 as one of its instances.

Document conceptual structures of one type are not required to be identical, because not all the semantic properties can be foreseen at the type definition time. This is called *weak typing* opposed to strong typing. A later refinement of types also can produce new types (subtypes). According to this refinement relation that specifies inheritance of type conceptual structures to subtypes the set of types can be organized in a directed acyclic graph. Both, weak typing and multiple inheritance provide the flexibility demanded above.

[1] The graphic representations are: — aggregate constructor (black square, specialization) black circle, value (arrow to value box, value domain) associated box.
[2] The graphic representation for choice-specialization is an unfilled circle.

These data structures together with the set of operations defined for creating, modifying and selecting them, form the basis of the MARS document model.

4. Some Basic Statements on the Specification of the MARS Model. There are several specification techniques for abstract data types, e. g. specification by intuition, example or implementation. These are all straight forward, but often things become complicated when investigating some more sophisticated questions.

We present an abstract specification that should describe how document semantics is represented and manipulated. In [14], [15] and others the algebraic specification formalism is recommended for describing data structures and the appropriate operations. It helps to present their syntax and semantics, supports the verification of the model properties and the consistent model specification. Furthermore, properties difficult to be expressed by a data model (e. g. "well defined" conceptual structure, "value" of a concept) can be shifted into the specification of the appropriate operations (modification and value operations, resp.).

Trying to follow the enhancements proposed in the algorithmic specification formalism by [17], we specify the semantics of most of the MARS operations through recursive functions. Thus the existence of an implementation and its freedom of contradictions can be simply derived.

Next we introduce two basic definitions and a small example to make the formalism employed in Chapter 4 more understandable.

An algebraic specification consists of two parts: *a syntactic specification* and the description of the *semantics of the syntactic elements.* The syntactic specification is given by a *signature* of the formal language or the abstract data type. Similar to the interface specification the signature determines the descriptors of the relevant data domains and the operator symbols, and defines the type of the operators. The basic elements of the formalism used hereafter are presented. A more exact background is given in [16].

Elements of algebraic specification:

Definition. *A Signature is a pair $\langle S, \Sigma\rangle$, where S is a set of sorts, Σ is a family of sets of operator symbols. Each element O_k of Σ is linked to a sequence $(s_{1k}, \ldots, s_{n+1k})$, $s_{ij} \in S$. An operator op is syntacticly defined by its symbol $op_s \in O_k$ and the association of the sequence of argument sorts $(s_{1k}, \ldots, s_{n_k})$ and its target sort s_{n+1k}.*

The second part of the algebraic specification determines the semantic properties. For each sort a carrier set is defined, i. e. the set of its values, and for each operator symbol a mapping from the carrier sets of the argument sorts to the carrier set of the target domain is given. This part is called a *Σ-algebra.*

Definition: *Let $\langle S, \Sigma\rangle$ be a signature, (C^s) denotes the carrier set of s and $C=(C^s)_{s \in S}$ is a family of sets. Let $C^{\varepsilon}=df\{\}$ and $C^{ws}=dfC^w \times C^s$, with $w \in S^*$ $s \in S$. For each op_s in each $O_k \in \Sigma$ a mapping op_sC is associated with $C(s_{1k}, \ldots, s_{n_k}) \rightarrow C^s{}_{n+1k}$. The pair $(C, \{op_sC \mid op_s \in S\})$ defines a Σ-algebra.*

Some extensions should be presented informally. We introduce **variables** by giving the corresponding *term-algebra* where the *terms* are: each element of a carrier set, each operator expression and each variable. A set of **equations** implies equivalence classes of different syntactic elements with the same semantics. Restricting operator specification to those classes abbreviates the specification process.

Example: -algebraic specification of the abstract data type "Set"-

The **signature** $\langle S, \Sigma\rangle$ is given by:

S= {Set, Int, Bool}

with **constructors**[3] for Set: emptyset: $\rightarrow$ Set, and insert: Set, Int $\rightarrow$ Set

[3] Constructors can be seen as unevaluable operators. The semantics of a constructor term is just the character sequence built according its definition.

$\Sigma = \{\text{InSet}: \text{Int}, \text{Set} \rightarrow \text{Bool}\}$[4], tests the membership of an Int-element in a set
The **Σ-algebra** comes with:

$C^{\text{Bool}} =_{\text{df}} \{\text{true, false}\}$
$C^{\text{Int}} =_{\text{df}} N$
$C^{\text{Set}} =_{\text{df}}$ {"the set of words defined by constructors"[5]}
$\text{InSet (i, s)} =_{\text{df}} \mid$ true, if $\exists e1, \ldots, e_n \in C^{\text{Int}}, s_{n+1} \in C^{\text{Set}}$ with
$s = \text{insert} (..(\text{insert} (s_{n+1}, 1), e_n)..), e1)$
$\mid$ false, else

Equations:

insert (insert (s, j), i) = insert (insert (s, i) j), with $i, j \in N$, $s \in$ Set
insert (insert (s, i), i) = insert (s, i), with $i \in N$, $s \in$ Set

An algorithmic notation of the example above would change the definition of the mapping given for the operator by a recursive function, e. g.

InSet (i, s) =df | true, if s = insert (s′ i)
| false, if s = emptyset
| InSet (i, s′), if s = insert (s′, j), $j \neq i$

5. Data Structures and Operators of the MARS Model. This section outlines the signature and the Σ-Algebra of the document model employed by MARS, as introduced informally in the third chapter. First we present the **set of sorts** together with the description of their carrier sets as comments (in brackets "(* .. *)"). Next for the sake of brevity we will identify a sort with the respective carrier set.

S = { CC, (* descriptors of the conceptual components, defined as concepts *)
Dom, (* set of sort descriptors of the employed value domains: in MARS: (atomic:) "Num", "String", (non-atomic:) "Text" *)
Num, Text, String (* value domains, also denoted by Dom_i, the carrier set of Text is a set of arbitrary sequences of String-elements *)
Bool, (* = {true, false} *)
CS (* conceptual structures (e. g. document-, type-, sub-structure *)
}

Elements of the sort CS are tree-like structures with concepts as nodes and **constructors** that define all the relationships between the nodes. The latter describe the structural properties of the document semantics and will be given in the following list:

CS = df (CC; (* single node structure *)
agg (CC,CS+[6]) (* concept with its aggregates and their substructures *)
spec (CC, CS+) (* concept with a set of its specialisations and their substructures *)
c-spec (CC,(CS+)+) (* concept with possible variants (choice-) specializations, applicable only in type structures, to make them more general *)
conc (CS+) (* combination of different kinds of structure with same root-thus it is used for technical reason and does not express any semantics of the document content *)
iterate (CC, Num) (* identical iteration of one concept with its substructures, applicable only in document structures *)
domain (CC, Dom) (* concept with associated value domain — establishes an attribute *)
value (CC,Dom,Dom_i) (* concept with linked value from given domain *)
)

[4] The target sort is prefixed with a "→".
[5] The syntactic definition of a sort with constructors implies a context free grammar; here: G = (Set, {emptyset, insert, N}, {Set}, {Set ::= emptyset | insert (Set, N)}) and $L(G) = C^{\text{Set}}$.
[6] The "S+" names a sort S that specifies all non-empty sequences of elements of sort S — to avoid introduction of new sort descriptors.

The constructive definition of CS allows the definition of different structures although describing same properties. The following **equations** E help to minimize the redundancy[7].

$E =_{df}$ (conc (Tree, Node) $=$ Tree, if Tree $= \Phi$ (Node, Arg$^{\rightarrow}$))
conc (Tree$_1$, conc (Tree$_2$, Tree$_3$)) $=$ conc (Tree$_1$, Tree$_2$, Tree$_3$)
Φ (Node, ε) $=$ Node, if $\Phi \in$ {agg, spec, c-spec}
Φ (Node, (Sub$_1^{\rightarrow}$, . . ,Sub$_n^{\rightarrow}$)) $= \Phi$ (Node, (Sub$_2^{\rightarrow}$, Sub$_1^{\rightarrow}$, . . ,Sub$_n^{\rightarrow}$)), if Φ as above
conc (Φ(Node, Sub$^{\rightarrow}$), Φ(Node, Sub$^{\sim\rightarrow}$)) $= \Phi$(Node, (Sub$^{\rightarrow} \oplus$ Sub$^{\sim\rightarrow}$)), if Φ as above
).

As we distinguish document conceptual structures from the generalized conceptual structures, a rule which is neither expressed in the data structure specification nor hidden in the operation specifications should be given informally:

Type-Instance-Rule:

(a) Type conceptual structures do not contain "iterate" or "value" constructors
(b) Document conceptual structures (as instantiations of a type structure) have:
-a "value" constructor for each "domain" constructor in the type (a value is liked to each attribute-concept)
-no choice of specialization ("c-spec" constructor)

The notation of a (simplified) conceptual structure is given in the following example. It refers to the structures given in fig. 1 and fig. 2, respectively.

Example: — referring to figures 1 and 2 above —
(a) **type** conceptual structure (→ fig. 2)
Let cs $\in C^{CS}$ be defined:
cs = agg (document, (domain (date, Num), conc (spec (content, text), agg (content, (domain, (name, String), c-spec (med-result, (radiological, nuclear)))))))
(b) **document** conceptual structure, instance of type in (→ fig. 1)
Let cs $\in C^{CS}$ be defined:
cs = agg (document, (value (date, Num, 881102), conc (spec (content, text), agg (content, (value (name, String, Smith), spec (med-result, (value (radiological, Text, ⟨offset, length⟩)))))))

The Syntactic Specification of the Operators Σ. The set Σ of sequences of argument and target sorts with the appropriate operator symbols are presented in the following list. Again an informal description of the operator function is given in comments:

Σ = {(CS, CC->CS)	view	(*provides a superstructure with a given leaf node *)
	project	(*provides a substructure for a given root node *)
(CS, CC->Bool)	is-concept	(*existence of a concept in a structure *)
(CS->CC)	root-concept	(*the root concept of a (sub-)structure *)
(CS, CS ->Bool)	is-substruc	(*existence of a substructure in a structure *)
	is-ref	(*compares two structures for refinement *)
(CS, CS, CS->CS)	subst	(*substitutes substructure in given structure *)
(->CS)	create	(*initializes a new structure *)
(CC, CS, CS -> CS)	ref	(*refines a given structure in a specified concept by a new substructure ("refinement-step") *)
(CS+, CC-> Dom$_i$*)	val	(*provides all the values related to a concept *)
(Dom$_i$, Dom$_i$ ->Bool)	<, >, =, ≠, contains[8]	(*relations on values[9] *)

[7] Notations: · Arg→ denotes the rest of an argument list
· ⊕ denotes concatenation of sequences.
[8] "t contains s" denotes inclusion of s ∈ String in t ∈ Text.
[9] Further relations depend on the employed value domains (e. g. image-qualifying relations, etc.).

(CS, CC, $Dom_i \rightarrow$Bool) select$_\rho$ (*compares according to $\rho \in \{<, >, =, \neq$, contains$\}$ the values of a concept with a given value *)

(CS$+\rightarrow$CS) merge (*combines a refined structure from given structures*)

Semantics of the Operators. In this section the semantics of the operators defined within the model is described. As structure manipulation is only supported by these sets of operations, much of the data model semantics is implied by the operations.

The operators can be grouped into three groups: we distinguish operators for

- creating (i. e. initializing) and modifying structures (create, ref, merge),
- restricting structures to subparts (project, view, value) and
- comparing values and structures (is-ref, select and the value relations: $\{<, >, =, \neq$, contains$\}$).

Furthermore, some basic operators are used so as to define the operators above (is-concept, is-substructure, root-concept, substitute). The operations are mostly defined recursively. We do not want to discuss all of them in this paper, but we will give some of the most essential ones:

The "create" and "ref" Operators: In the specification of the sort CS all possible structures are defined. Although, not all of these elements describe **well defined conceptual structures** of documents or types. These consistency constraints are hidden in the "ref" operator. Its definition describes all cases of allowed refinement steps leading again to a well defined structure. All well defined structures have the root concept in common (named: root^0), initialized by the operator "create":

create () $=_{df}$ (root^0)

Some of the constraints given in the "ref"-operator, are presented below:[10]

ref (RefNode, Ref Step, Tree) $=_{df}$

```
    (1) | Tree [RefNode/Ref Step],
                  if project (RefNode, Tree) = RefNode
    (2) | Tree [domain (RefNode, dom) / value(RefNode, dom, v)],
                  if project (RefNode, Tree) = domain (RefNode, dom)
                  and RefStep = value (RefNode, dom, v) and $v \in C_{dom}$
    (3) | Tree [$\Phi$ (RefNode, ($Sub_1, \ldots, Sub^n$)) / $\Phi$(RefNode, ($Sub_1, \ldots, Sub_n$)$\oplus$
                  RefTrees$\rightarrow$)],
                  if project (RefNode, Tree) = $\Phi$(RefNode, ($Sub_1, \ldots Sub_n$)) and
                  RefStep = $\Phi$(RefNode, RefTrees$\rightarrow$), für $\Phi \in$ {agg, spez, c-spez}
    (4) | Tree [c-spec (RefNode, (($Sub_{1_1}, \ldots, Sub_{1_n}$), .., ($Sub_{m_1}, \ldots, Sub_{m_n}$))) /
    conc (c-spec (RefNode, (($Sub_{2_1}, \ldots, Sub_{2_n}$), .., ($Sub_{m_1}, \ldots, Sub_{m_n}$))
          spec (RefNode, ($Sub_{1_1}$) )) ]
              if project (Ref Node, Tree)
                  = c-spec (RefNode, ($Sub_{1_1}, \ldots, Sub_{1_n}$), .., ($Sub_{m_1}, \ldots, Sub_{m_n}$))
              and RefStep = spec (RefNode, ($Sub_{1_i}$))[11]
  (5) | Tree [$\Phi_1$ (RefNode, Sub$\rightarrow$) )/
    conc ($\Phi_1$ (RefNode, Sub$\rightarrow$), $\Phi_2$ (RefNode, Sub$\rightarrow$) )],
              if project (RefNode, Tree) = $\Phi$1 (RefNode, Sub$\rightarrow$)
              and RefStep = $\Phi_2$ (RefNode, Sub$\rightarrow$)
              with $\Phi_1$, $\Phi_2$ not unifyable (e. g. $\Phi_1$ = agg, $\Phi_2$ = c-spec)
  (6) | Tree [$\Phi$ (RefNode, Sub$\rightarrow$) / iterate ($\Phi$ (RefNode, Sub$\rightarrow$), Num)],
              if Ref Step = iterate (RefNode, Num) and $\Phi \notin$ {value, iterate}
  (6) | undefined, else.
```

[10] Notations: · Arg$\rightarrow$ denotes the rest of an argument list
· $\otimes$ denotes concatenation of sequences

[11] Without restrictiond according to equations.

Table 1

Φ RefStep							
Tree	CC	dom	value	agg	spec	c-spec	
iterate							
CC	n	s 1	s 1	s 1	s 1	s 1	s 1
dom	n	c	s 2	c	c	c	s 6
value	n	—	—	—	—	—	—
agg	n	c	—	s 3	c	c	s
6 spec		n	c	—	c	s 3	c
s 6	c-spec		n c	—	c		s 4
s 3 s 6							
iterate	—	—	—	—	—	—	—

The following table summarizes the substitution rules given in the "ref"-specification. The labels are "n" (no substitution), "si" (substitution according to rule i), "c" (concatenation with "conc" constructor, same as application of rule 5) and "-" (not defined).

The val, select and is-ref Operators:

```
val (Tree, Node) = df
        | {dom, v) },   if project (Tree, Node) = value (Node, dom, v)
        | ∪(SubTree ∈ Sub) val  (SubTree, root -concept (Sub Tree))
              if Sub = {SubTree | is-substruc (SubTree, project (Tree
                 Node) and SubTree ≠ Tree}
        | {},    else

select  (Tree, Node, c) = df
        | true,    if (domi ≠ Text
                      and ∃(v_i, dom_i) ∈ val (Tree, Node): v_i ρ  c, and ρ ∈
                      {<, >, =}
                      or  ¬∃(v_i, dom_i) ∈ val (Tree, Node): v_i = c, and
                      ρ = "≠")
                   or (∃(v_i, dom_i) ∈ val (Tree, Node): v_i = c, and dom_i ≠
                      Text
                      or  ∃(v_i, Text ∈ val (Tree, Node): v_i  ρ  c)  and
                      ρ = "contains"
        | false,   else.
```

As stated before, document structures are refinements of appropriate types (and associated to those types), and the type catalogue itself defines a directed acyclic graph according to the refinement relation. Refinements are defined by applying the "ref" operator-thus, structures can be equivalently specified by an appropriate sequence of "ref"-applications. The "is-ref" operator uses this notation for specifying two structures to be in refinement relation:

```
is-ref (Tree Spec, Tree) = df
        | true,    if ∃ {N_1, .., N_n}, {RefStep_1, .., RefStep_n} with n ≥ 1 and
                      Tree Spec = ref (N_n, RefStep_n) ref (N_1, RefStep_1)
                      (Tree)
        | false,   else.
```

Query-Language. The query language can be defined by the operations above. The features of the query language are: qualifying documents by
(a) typical properties, given through their type association, (*type clause*) and/or,
(b) the existence of a concept, (*existence clause*) and/or,
(c) the value of a concept (*value clause*)[12].

The expressions of (a) are defined by the "is-ref" operator, (b) is defined with the "is-concept" operator and (c) is mapped to appropriate applications of the "select$_\rho$" operator. The evaluation of the operation expression is to be interpreted against a state of the MARS document archive $\mathcal{A}$ enhanced by the state of the currently defined types (catalogue) *C*. We define such an extended state s $_{\mathcal{AC}}$ as follows:

Definition: *An extended state* s$_{\mathcal{AC}}$ *of a MARS archive is a triple* (A,T, L) *with*:

· *A is a set of pairs* (ldi, cs), *with ldi* $\in$ ID$_{\mathcal{A}}$, a *countable set of internal logic document identifiers and cs is a document conceptual structure and* ldi is *unique in* A.

· **T** is a set of pairs (lti, cst), with lti $\in$ ID$_C$, a countable set of internal logic type identifiers and cs is a type conceptual structure and lti is unique in T.
In a *consistent state* the two functions L and H are totally defined:

· L is a function linking each pair (ldi, cs)$\in$ A to a *best fitting* type (lti, cs$_t$)$\in$T:
$=_{df}$ { ((ldi, cs}, (lti, cs$_t$) | (ldi, cs)$\in$A, (lti, cs$_t$)$\in$ T, and is-ref (cs, cs$_t$)
and $\neg$ ($\exists$ (lti′, cs$_t$) $\in$ T: ((ldi, cs), (lti′, cs$_t$)) $\in$ L
and is-ref (cs, cs$_t$′) and is-ref (cs$_t'$, (cs$_t$)) }

· H is a function linking each pair (lti, cs)$\in$ C to its set of supertypes (thus representing the Hasse-Diagramm[13] of the type Catalogue):
$=_{di}$ { ((lti, cs), (lti′, cs′)) | (lti, cs), (lti′, cs′)$\in$ T and is-ref (cs, cs′)
and $\neg$ ($\exists$ (lti″, cs$_t$″)$\in$ T: ((lti,cs), (lti″, cs$_t$″)) $\in$ H
and is-ref (cs, cs″) and is-ref (cs″, cs′)) }.

The **evaluation** of the three kinds of clause-expressions is defined in the following way: (a) For each type identifier lti$_t$$\in$ ID$_C$ of the type clause the set

{ (ldi$_d$, cs) | ldi$_d$$\in$ ID$_{\mathcal{A}}$ and is-ref (cs. cs$_t$)}

is evaluated, if (lti$_t$, cs$_t$)$\in$ T, {} otherwise. Thus we get the transitive closure according to the refinement relation. Sets for different type identifiers in the clause are united.

(b) For each concept c given in the existence clause the set
{(ldi$_d$, cs) | ldi$_d$$\in$ ID$_{\mathcal{A}}$ and is-concept (cs, c)}
is evaluated.

(c) For each comparison (c ρ val) given in the value clause the set
{(ldi$_d$, cs) | ldi$_d$ $\in$ ID$_{\mathcal{A}}$ and select$_\rho$ (cs, c, val)}
is evaluated.

The boolean relations AND and OR in (b) and (c) are mapped to union and intersection respectively.

An **example query** that will qualify the document given in fig. 1, may be: FIND DOCUMENTS [OF TYPE xxx] WHERE date>880101[14] AND med-result CONTAINS "tumors".

With the specifications of the operators and the state given above, complex functions like query evaluation and type catalogue updating can be developed. We distinguish two classes of functions:

[12] A further qualification of documents is supported in MARS by restricting the search scope to documents of defined collections (*collection clause*) — they may be created by former query-results. As it is not a special document qualification coming from the model it is not discussed here.

[13] The graph-representation of an ordered set (here: is-ref order on the type structures) without transitivity.

[14] Date in a normalized representation.

- functions with read only access (query evaluation, type calalog browsing, etc.)
- functions updating the state of the archive (type insertion and removal, document filing etc.).

For the first class optimized algorithms can be verified. E. g. for the evaluation of the type clause first evaluating the set of all types in the transitive closure of the given type by consulting the H-relation and then inquiring the L-relation will propably provide the specified documents more efficiently. For updating functions consistency preserving transactions with respect to L- and H-relation can be formulated.

5. MARS Software Architecture. In fig. 3 we present the layered software architecture of MARS (the full documentation is included in [6]). It is implemented in Modula2 on Sun3 computer systems running the UNIX* operating system. The **Storage Layer** includes basic file system facilities. The HOST software interface [5] provide an operating system independent library to access basic system functions. For secondary storage sequential files are used and for efficient multikey access the gridfile system [19] is employed.

The **Data-Handler Layer** encapsulates all fundamental data types. The module *CSDHd* for handling conceptual structures follows the specification given in Chapter 4. The specification of the conceptual components by an associated value domain in a type conceptual structure allows their being treated as attributes providing access through value qualification. To speed up query evaluation with respect to those attributes some special access structures can be defined, e. g. text-signatures given in [12] for the components with Text-value and the index- or hash access for the components with atomic domains (we employ the gridfile system for them). The *SpAccHd* (Special Access Handler) keeps track of those definitions, the *SignHd* and *AtrAccHd* support the respective filing and retrieval facilities. Each object, i. e. type or document with its conceptual structure, is uniquely referenced by an internal logic object indentifier (LOI). To handle long lists of LOIs, e. g. in the query evaluation process, the *ListHd* provides efficient setoperations on dynamic hierarchic bitvector-trees for manipulating those lists.

Complex functions operating on the state of the archive defined on top of the basic operations of the model (like query evaluation or type catalogue updating), are partitioned in tools of the **Tool Layer.** Here we placed the functions mentioned at the end of Chapter 4. These tools can be combined easily to define approporiate user applications. A machine independent *user interface manager* toolkit is realized in the module "Fantasy" [13] that permits text or graphic user interface I/O. The *type management* tool includes facilities for browsing and updating the type catalogue with preservation of consistency with respect to the refinement relation defined earlier. Furthermore the associaton of documents to types is handled here (DocSetHd). The *document access manager* controls the insertion and deletion of document conceptual structures and the updating of possibly defined special access structures. The *query manager* tool interpretes the query that may be a logic combination of type-, existence-, value- and collection clause and maps the different retrieval requests to the appropriate handlers (SignHd, AtrAccHd, ListHd, DocSetHd). Arbitrary document sets coming e. g. from queries can be stored and combined with functions from the *collection manager* tool. The *CSManager* provide functions for defining conceptual structures in a consistent way (according to the "ref" operator introduced in Chapter 4).

The tools support the development of functions for the user interface. These functions are situated in the **Application Layer.** They can be grouped as follows:

1. Type-Administrator — Type Catalogue Init, Type Insert, Type Remove, Type Modify
2. Type Browsing — Find Supertype, Find Subtype, Get Type Definition

* UNIX is a trademark of AT&T Bell Laboratories.

3. Document Handling — Document Insert, Document Remove, Get Document
4. Collection Handling — Collection Init, Insert in Collection, Remove from Collection
5. Query — Find Document, qualified by Type, Concept, Concept Value, Collection
6. Access Optimizing — Add Special Access, Remove Special Access.

To access and present documents on an appropriate output medium functions from the Picture Archiving and Communication System (PACS) (see Chapter 7) are called from different layers through an interface with the industrial standard ACR-NEMA. The SPI standard enhancement, supported in this PACS specifies the client-server communication protocol.

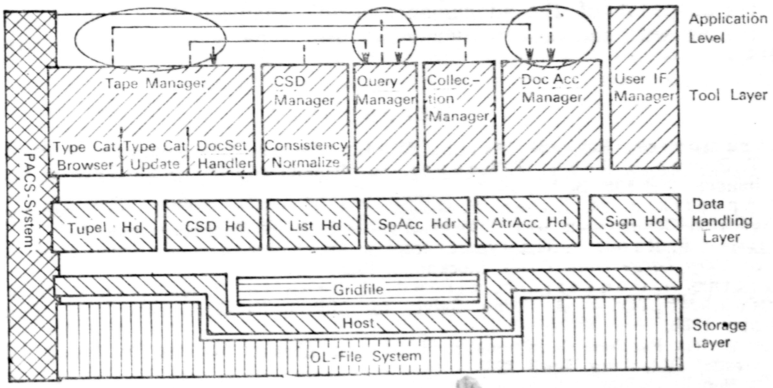

Fig. 3. Software layers in the MARS architecture

7. Test Application in a Clinical Environment. A first application of MARS that supports the manipulation of conceptual structures is developed for the radiologic department of the district hospital in Oldenburg (SKO: Städtische Kliniken Oldenburg). To provide basic storage and retrieval functions for a large amount of data (estimation: 20 GByte yearly for approximately 60.000 documents of different types) a Picture Archiving and Communication System (PACS), specially developed for clinical applications (see appropriate papers in [7]), is to be installed[15]. While the PACS provides the storage and presentation of the original documents, MARS manages the type and document conceptual structures and provide their manipulation and access.

Although the rather static "document type world" in the radiologic department of SKO is not the typical application environment of the MARS system that aims to support frequently restructuring and enhancement of the type catalogue, it gives an organizational advantage: as the flow of activities in the organization relies on the document and formular structures, the activities should not be changed, as those existing structures can be directly mapped onto appropriate type structures defined in the MARS model. This simplifies the snapshot.

8. Conclusion. Multimedia documents can be seen by a layout view, a syntactic view or a semantic view. This paper focuses on the last one, defining abstract representations of documents via tree like conceptual structures. Typical properties of classes of document structures can be generalized into type conceptual structures. With those types and their associations to documents complex retrieval functions can be supported. In the MARS project currently performed at the University of Oldenburg appropriate facilities are implemented. This paper concentrates on the specification of the

[15] Currently the PACS Services are simulated.

model, i. e. both data structures and operations and gives an overview of its implementation.

This paper concentrates on the specification of the model, i. e. both data structures and operations. It outlines the approach of an algebraic specification, suitable for describing abstract data types. While the syntactic specification with an informal semantic description is given in detail, presentation of the semantics of the operators is reduced to essential definitions. With the basic operators complex functions (e. g. query evaluation, type update, etc.) that operate on the state of the archive are described.

In a clinical environment an application has been modelled. Ten document types represent a snapshot of the "document world" analysed in a radiologic department. Further testing with the first implemented prototype will allow quantitative analysis of MARS.

REFERENCES

1. K. Abramovicz, K., R. Dittrich, W. Gotthard, R. Laengle, P. C. Lockemann, T. Raupp, S. Rehm, T. Wenner. Datenbankunerstuetzung für Software Produktionsumgebungen. IFB **136**, Berlin, 1987.
2. H.-J. Appelrath, M. Ester, H. Jasper, A. Ultsch. KOFIS: ein Expertensystem zur integrierten Dokumenten- und Wissensverwaltung. *Proc· Expertensysteme* '87.
3. F. Barbic, F. Rabitti. The Type Concept in Office Document Retrieval. *Proc. 11th Conference on Very Large Data Bases*. Stockholm, 1985.
4. E. Bertino, A. Converti, H. Eirund, K. Kreplin, F. Rabitti, P. Savino, C. Thanos. MULTOS — A Filing Server for Multimedia Documents. *Proc. Euroinfo*. Athens, May 1988.
5. Interner Bericht Fachbereich Informatik. HOST: An Abstract Machine for Modula 2 Programs. Universität Oldenburg, April 1988.
6. Interner Bericht Fachbereich Informatik. Endbericht der Projektgruppe TECDOS/MARS. Universität Oldenburg. März 1989.
7. H. U. Lemke, M. L. Rhodes, C. C. Jaffee, R. Felix. Computer Assisted Radiology. *Proc. CAR 87* Berlin, 1987.
8. P. Constantopoulos. H. Eirund, K. Kreplin, F. Rabitti, C. Thanos et al. Office Document Retrieval in MULTOS. *Proc. 3rd ESPRIT Technical Week*, 1986.
9. W. C. Croft. User-specific Domain Knowledge for Document Retrieval. *Proc. ACM Conf. on Research and Development in Information Retrieval*, ABM, Pisa, 1989.
10. European Computer Manufacturers Association. Office Document Archi̇techture ODA-ECMA-101. Sept. 1985.
11. H. Eirund. Knowledge Based Document Classification Supporting Content Based Retrieval and Mail Distribution. *Proc. IFIP TC6/TC8 Symposium* — Sofia, May 1988.
12. C. Faloutsos. Access Methods for Text. *ACM Computing Surveys*. Vol. 17/1, March 1985.
13. F. Försterling. Database Requirements in the Office. Preprint of Xth Int. Seminar on DBMS. Akademie d. Wiss. der DDR. Inst. f. Informatik und Rechentechnik, Berlin (DDR). Apr. 1988.
14. J. Guttag. Abstract Data Types and the Development of Data Structures. CACM 20/6, June 1977.
15. W. Henhapl, T. Letschert. VDM — Vienna Development Method. Informationstechnik it, 29/4, Oldenbourg Verlag München, 1987.
16. H. A. Klaeren. Algebraische Spezifikation — Eine Einführung. Berlin, 1983.
17. J. Loeckx. Algorithmic Specifications: A Constructive Specification Method for Abstract Data Types. TOPLAS 9/4. ACM Oct. 87.
18. H. Lorek. Fantasy: Methods and Tools for the Development of Graphic User Interfaces. Proc. Graphik im Bürobereich. Bad Honnef. IFB 1932. Berlin. Nov. 1988 (in German).
19. J. Nievergelt, H. Hinterberger, K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM TODS 9/1 ACM March 1984.
20. G. M. Sacco. OTTER — An Information Retrieval System for Office Automation. *Proc. 2nd ACM SIGOA Conference on Office Information Systems*. Toronto, 1984.
21. H.-J. Schek, P. Pistor. Data Structures for an Integrated Data Base Management and Information Retrieval System. *Proc. VLDB Conf.*, Mexico, Sept. 1982.
22. H.-J. Schek, M. Scholl. Die NF2 — Relationenalgebra zur einheitlichen Manipulation externer, konzeptueller und interner Datenstrukturen. *Proc. Sprache für Datenbanken IFB 72*, Berlin, 1983.

Universitat Oldenburg,
FB Informatik,
P. O. Box 2503, D-2300 Oldenburg

Received 17. 05. 1989
Revised 11. 06. 1990