Query Evaluation in an Object-Oriented Multimedia Mediator

Joachim Biskup, Jürgen Freitag, Yücel Karabulut, Barbara Sprick
Fachbereich Informatik, Universität Dortmund
{biskup | jf | karabulu | sprick}@ls6.informatik.uni-dortmund.de

Abstract. A multimedia mediator aims at providing a well-structured gateway to some application dependent part of a federated multimedia system. Our specific design employs proxy objects for external multimedia items and introduces a new concept of semi-structured and self-describing types for multimedia items. Query evaluation and optimization hide all details of communication with external sources and explore the external parallel computation capacities, the selectivity of local preprocessing, and the impact of materialization.

1 Introduction

A multimedia system can be considered as a distributed, federated, extensible information system that supports a wide range of object types and allows a high degree of autonomy of its components. A *multimedia mediator* aims at providing a well-structured and controlled gateway to some application dependent part of the whole system. Our design of such a mediator is based on multimedia schemas and on object-oriented concepts. In particular we employ proxy objects for external multimedia items and introduce types for semi-structured multimedia items. High level query processing hides all details of external communications, and query optimization takes advantage of parallel processing capacities of external sources and of materializations of previous query answers in the proxy state.

Presenting our design of query evaluation in the multimedia mediator, we

- discuss the role of proxy objects in representing external multimedia items,
- describe two basic approaches for querying proxy objects,
- show how communications with external sources can be hidden transparently,
- outline optimization methods for query processing dealing with the time tradeoff between local selectivity and external parallelism,
- explore the impact of persistent materialization of query results, and
- reconsider types for semi-structured multimedia items and introduce a new concept of semi-structured and self-describing types.

Our presentation is organized as follows. Section 2 summarizes the general architecture of our multimedia mediator. Section 3 presents basic approaches to query evaluation with external communication. Section 4 studies optimization of query evaluation. Section 5 deals with persistent materialization of attribute values. Section 6 deals with semi-structured multimedia items. Finally, Section 7 briefly compares our design with alternative approaches.

2 Architecture of the multimedia mediator

A multimedia system can be considered as a distributed, federated, extensible information system that supports a wide range of object types and allows a high degree of autonomy of its components. Seen as a *distributed* system, a multimedia system is based on a communication infrastructure that, at least potentially, allows worldwide communication. A mediator provides a transparent, application specific view on the whole system. Seen as a *federated* system, a multimedia system allows access to heterogenous information sources by providing appropriate translations between the various data models and protocols for interaction of its components. A mediator implements such translations and protocols in a transparent fashion. Seen as a *type system*, a multimedia system supports a broad spectrum of types. A mediator integrates the various types required for the specific application within a uniform type system, which includes appropriate types for semi-structured data. Seen as an *extensible* system, a multimedia system can be extended with respect to the federation structure and with respect to the type system. A mediator easily adapts any such extension.

Our general approach for a multimedia mediator is based on two established technologies for database systems: using *multimedia schemas* as a self-description of a specific application, in order to achieve well-structuredness and control on the one side, and employing *object-oriented concepts*, in order to federate heterogenous, distributed and evolving information sources on the other side. The resulting overall architecture is sketched in Figure 2-1, and the basic features are explained below.

A *multimedia schema* has three layers. The *view layer* offers interfaces for the local and external users. The *conceptual layer* constitutes the instantiation of the me-

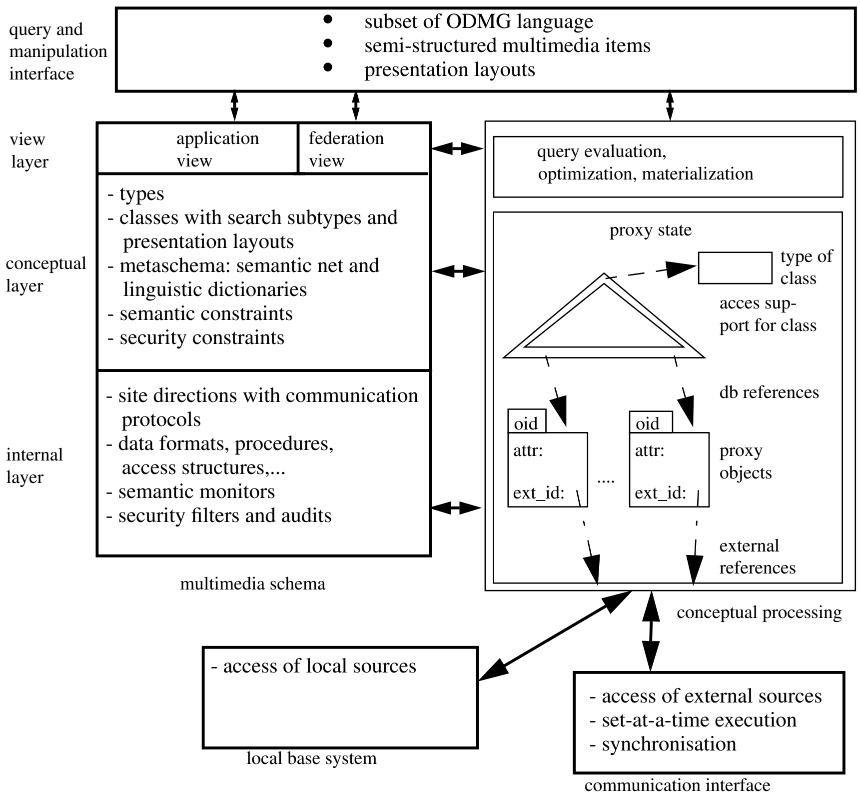

Figure 2–1 Architecture of the multimedia mediator

diator for the specific multimedia application. It contains a full structural and operational description, including multimedia *types*, *conceptual classes*, *semantic constraints*, *security constraints*, and a *metaschema* as abstract knowledge representation tool. The *internal layer* provides the details of how the conceptual items are implemented. Most importantly, it comprises *site directions*, for indicating whether a conceptual multimedia item is locally available and, if not, for providing communication protocols for accessing that item elsewhere in the multimedia system.

We employ *object-oriented concepts* as in any traditional object-oriented information system. Besides that, we take advantage of object-orientation for hiding the heterogeneity and distribution of the information sources within the whole multimedia system.

Firstly, any multimedia item visible in the mediator‘s view is represented by a uniquely identified object that is under the control of the mediator. If the represented item is locally stored, we call that object a *local object*; otherwise, if it is externally stored, that object is used as a *proxy object.* We also allow that a single object is used in a mixed way. Thus the mediator maintains a full *proxy state* of the application driven view of the whole multimedia system, which consists of all created objects of any kind. Accordingly, a user of the mediator gets the illusion of dealing with just this proxy state and he is not at all aware of the storage sites.

Secondly, the class specific procedures for types hide all the subtle details on how to actually access the items represented by the local and proxy objects.

Our multimedia mediator is designed to serve as an integration tool within an ambitious project for building a high performance multimedia query server, HPQS. The overall goal of the HPQS is to provide high-level, user-friendly, and interactive access to an application domain. This domain is represented by both classical knowledge representation techniques, in particular semantic nets and linguistic dictionaries, and various kinds of multimedia items, in particular texts and images.

In order to achieve high performance, the HPQS is based on a specialized parallel computing system which can be tailored to the most frequently required elementary search requests on texts and images of an application. In principle, there can be more than just one underlying specialized parallel system, and these systems can be remote from the user site. Therefore, given a semantically meaningfull query issued by some application oriented user, our multimedia mediator has to extract appropriate sets of elementary subtasks to be processed by the possibly external parallel systems. It is important to note that the mediator has to forward *sets* of elementary subtasks which are *as large as possible* in order to take advantage of the parallel processing capacities of the underlying parallel systems.

On the one hand, we assume mass of data, i. e. texts and images, processed by underlying parallel systems, and on the other hand, we recognize semantically elementary but computationally complex search requests on texts and images as a dominant cost factor for query evaluation. Thus, we designed our multimedia mediator to employ not only *parallelism* but also *materialization*, in order to avoid recomputations of elementary requests whenever possible and useful.

3 Query evaluation with external communication

From the user´s point of view the multimedia mediator should appear like an ODMG compliant database, which has been augmented by additional features, in particular with transparent access to external multimedia items and with types for semi-structured multimedia items.

Let us consider the type of a class. We assume that for any structural component of the type, i.e. an attribute, there is a class specific standard procedure for reading the attribute value for a given object. We distinguish three cases:

1) The attribute value is locally stored and thus can be simply fetched.
2) The attribute value must be determined from an external item by means of a communication.
3) The attribute value is redundantly available both by fetching from local storage and by querying an external source.

The last two cases only apply if the attribute belongs to a proxy object and represents a property of an external item. Basically, the procedure for reading an attribute value first tries to find the value locally and if this attempt fails, it generates an appropriate command to query the external source. The details of such procedures are discussed below.

When evaluating a query, we have to call the procedures for reading attribute values, typically while iterating over the population of some class. There are two simple brute force approaches to handle the calls of these procedures: the one-pass approach and the two-pass approach.These brute force approaches can be combined and refined to optimized approaches.

In the *one-pass approach*, the original query is processed nearly as usual. However, whenever a call of a reading procedure starts an external communication, the query process is suspended until the external value has been supplied. More precisely, we assume that for any attribute *A* there are two standard procedures *one_pass_read_A* and *write_A*, which serve for reading and writing, respectively, an attribute value. Moreover, the type of attribute *A* necessarily has to contain a special value *extern* that is different from any regular value. If for attribute *A* only the special value *extern* is locally stored, then the reading procedure calls a function *communicate* the purpose of which is to supply the (real) attribute value from an external source. Then the standard procedures can be sketched as follows:

- *procedure one_pass_read_A( ): attribute_type;*
begin
    if A = extern
    then write_A(communicate(external_source, A_op, oid, external_ident));
    return A
end one_pass_read_A;
- *procedure write_A(new_value: attribute_type);*
begin
    A := new_value
end write_A;

Here *communicate* is a globally known function that implements the communication with an external source, say by an RPC-like protocol. The function *communicate* needs appropriate parameters, say of type *communicate_input_type*, which are retrieved from the currently considered object:

- *external_source* uniquely identifies the external agent that must be queried for supplying the required attribute value;
- *A_op* is used as an operation symbol supposed to be known by the external agent;
- *oid* uniquely identifies the object under consideration within the mediator;
- *external_ident* uniquely identifies the multimedia item that is represented by the object under consideration, where now identification is within the external agent.

If the function *communicate* sends the last three parameters to the external agent identified by the first parameter, *external_source*, then this agent interpretes the second parameter, *A_op*, as an operation which is applied to the item denoted by the fourth parameter, *external_ident*. The result of this operation is sent back to the function *communicate*, which in turn delivers it as its return value. Of course, instead of just sending an operation symbol, *A_op*, we also could send an executable program, say written in Java, to the external agent. All such details are declared in the internal layer of the multimedia schema. As a side effect, after a call of the reading procedure the (real) value of attribute *A* is always locally materialized. Depending on a materialization strategy, such values can be made persistent.

In the *two-pass approach*, the original query is processed in four steps:

a) The original query *Q* is syntactically analysed to determine, which calls for reading procedures are required. Based on this analysis, a new transformed query *Q_external* is generated. As a query result, it just delivers the set of all parameters to be passed to the function *communicate* later on, and finally to be forwarded as commands to the external sources.
b) The transformed query *Q_external* is actually executed.
c) Its output is appropriately sent to the external sources. Afterwards, the returning answers, containing the attribute values, are interpreted as insert commands that materialize the attribute values locally. Thus, when all answers have been returned, then all attribute values required by the original query are locally available.
d) Finally the original query *Q* is executed. Since now all required attribute values can be locally fetched, query processing can be done as usual.

For the two-pass approach we need a slightly modified reading procedure for an attribute *A*. The modified procedure delivers either a parameter tuple for external communication, i.e. of type *communicate_input_type* or a (real) value of type *attribute_type*:

- *procedure two_pass_read_A ( ) : attribute_type union communicate_input_type;*
  begin
  if A = extern
  then return tuple_of(external_source, A_op, oid, external_ident)
  else return A
  end two_pass_read_A;

Whereas in the one-pass approach the global function *communicate* is immediately and directly called with appropriate parameters, now the reading procedure only has to supply the appropriate parameters, but the actual call of the function *communicate* is deferred to step c). Thereby, we can use a variant of the function *communicate* that accepts a whole set of parameters which is sent as a bulk to the external agent (or even to several external agents if the quadrupels in the set differ in the first component). The external agent then interpretes all commands that it has received, preferably in parallel, and after completion it returns the set of all results. Of course, now any single result value must be tagged with the pertinent object identifier and the pertinent attribute, i.e. instead of just any *value* we get a triple of the form *(value, oid, A_op)*. Once the returned result set is available at the mediator, the results can be materialized in the proxy state by executing appropriate writing procedures.

In order to illustrate the two-pass approach, we present a very simple example. Let the original query *Q* be as given in Figure 3–1. Then the evaluation of *Q* consists of the following four steps:

a) The original query *Q* is transformed into the query *Q_extern* that is intended to deliver the set of all parameters to be used by the global function *communicate*. *Q_extern* is determined as given in Figure 3–2.
b) Then the transformed query *Q_extern* is executed (as usual). The query result is denoted by some identifier, say *commands.*

select $x.A$
from x in *population*
where $x.A \leq$ "*const*"

Figure 3–1 query Q

select $x.A$
from x in *population*
where $x.A$ = *extern*

Figure 3–2 query *Q_extern*

c) Subsequently, the global function *communicate* is executed with the actual parameter set *commands.* The returned set of tagged values is denoted by some identifier, say *results*. By iterating over *results* we can materialize the (real) attribute values as sketched by the following fragment of pseudo code:
for all (value, oid, A_op) of results do oid.write_A(value) end
d) Finally, executing the original query Q (as usual) we get the overall answer.

4 Optimization of query evaluation

The one-pass approach treats all external communications sequentially, and thus it is unlikely to be efficient. The two-pass approach treats all external communications simultaneously, but it has three other obvious drawbacks. Firstly, it does not allow parallelism between local processing and external communication. Secondly, it cannot take advantage of any optimization technique to reduce the number of attribute accesses or related features (for instance semi-joins or side-way information passing). Finally, it actually only works for a restricted kind of queries that allow to precompute the set of all objects for which an attribute value must be determined. Therefore, besides the brute force approaches there is room for a large variety of optimized evaluation strategies. The query optimizer of the mediator should be able to select an appropriate strategy for any given query.

As an example, we outline an optimized evaluation strategy for conjunctive (object) queries. Such queries have the following general syntactical form:

select *xi.Ai*,... from *xi* in *Ci*,... where *xk.Ak* = *xl.Al* and ... and *xj.Aj* = *const* and ...

Here, as above, *Ci* is a class (identifier), *xi* is an object variable, the clause "*xi* in *Ci*" binds the object variable *xi* to range over the population of class *Ci*, and *xi.Ai* denotes the value of attribute *Ai* of the object denoted by *xi*.

The optimized evaluation strategy is based on the classical heuristic to perform the selections, which are required by the where-clause, as early as possible in order to minimize the size of intermediate results. In particular, the selections required by conditions of the form *xj.Aj* = *const* are executed before any other operation, and selections required by conditions of the form *xk.Ak* = *xl.Al* are performed immediately after the set of all pairs of objects (*xk*,*xl*) that must be examined is actually available. In order to pursue this heuristic, the original flat query is syntactically transformed into a nested query in which intermediate results of subqueries are explicitly denoted by identifiers. The nested query in turn can also be expressed as a sequence of simple flat queries. Starting with innermost nested queries, any single flat subquery is treated as in the two-pass approach. As a result, we get something like a "multiple two-pass approach". This outline is illustrated by an example query.

select $y.D$
from x in *firstpop*, y in *secondpop*
where $x.A$ = "*multimedia*" and $x.B = y.C$

Figure 4–1 conjunctive query Q

Consider the conjunctive query *Q* shown in Figure 4–1. Following the optimization heuristic, we can rephrase the flat query *Q* by an equivalent sequence of queries as follows. The outermost query *Q3* is shown in Figure 4–2, the definition of *intermediate_2* is given by query *Q2* shown in Figure 4–3, and finally the definition of *intermediate_1* is given by query *Q1* shown in Figure 4–4.

```
select y.D
from y in intermediate_2
```

Figure 4–2 query *Q3*

```
select y
from x in intermediate_1,
     y in secondpop
where x.B = y.C
```

Figure 4–3 query *Q2*

```
select x
from x in firstpop
where x.A = "multimedia"
```

Figure 4–4 query *Q1*

The original query *Q* is equivalent to the query sequence *Q1*, *Q2*, *Q3*, where each subquery *Qi* can be treated separately by the two-pass approach. In Figure 4–5 to Figure 4–7 we only present the transformed queries *Qi_extern*.

```
select x.A
from x in firstpop
where x.A = extern
```

Figure 4–5 query *Q1_extern*

```
select x.B
from x in intermediate_1
where x.B = extern
   union
select y.C
from y in secondpop
where y.C = extern
```

Figure 4–6 query *Q2_extern*

```
select y.D
from y in intermediate_2
where y.D = extern
```

Figure 4–7 query *Q3_extern*

The outline for an optimized evaluation strategy for conjunctive (object) queries does not completely specify, how to precisely decompose a query into appropriate fragments. Obviously, as usual in query optimization, for each issued query we have to explore various possibilities for decomposition, thereby estimating the expected evaluation costs, and finally determining the expectedly best available option. Basically we are faced with a fundamental time tradeoff between

- *selectivity* of the *mediator´s* local operations on the proxy state on the one side,
- *parallelism* of the *external agents* operating on external multimedia items on the other side.

Here, the selectivity *sel* measures the impact of the mediator´s heuristic to perform selections early, where *sel* with $0 \leq sel \leq 1$ is the ratio of actually selected objects to inspected objects. And the parallelism, *par*, describes the impact of parallel computing capacities of external agents, where *par* with $1 \leq par$ is the number of items that can be processed in parallel. If *card* denotes the number of objects, or equivalently of represented external items, in the search space of a subquery under consideration, then the evaluation time, *time*, of the subquery is roughly estimated as follows (where proportional constants are omitted for the sake of succintness):

$$time \approx \frac{sel}{par} \times card$$

Unfortunately, the essential parameters sel and par of a decomposition are not independent. Additionally, the actual selectivity *sel* does not only depend on the current instance, the characteristics of which could sometimes be determined from the schema, but also on the materializations stemming from preceding queries. And finally, the parallelism can also be affected by the decomposition, since any conjunctive query can be considered as containing some kind of m-ary (object) join operation, which has to be translated into a tree like structure of binary joins. For an issued query, some translations may result in a tree that allows parallel evaluation of subtrees whereas other translations do not.

When lacking detailed information for better estimates, we pragmatically propose to favour selectivity as a default, i.e. the optimizer should assume some small fixed default value for *par* and examine only selectivity in detail by using the usual query optimization techniques.

5 Persistent materialization of attribute values

Any evaluation strategy potentially supplies some attribute values that have not been locally available before. This offers the option to materialize these newly available values permanently in the local storage. If they are subsequently needed for further queries they can be simply fetched, and thereby their possibly time and resource consuming recomputation can be avoided.

Pursuing this option we have to deal with several problems, including:

- *Materialization* strategies: based on the knowledge of the application, in particular as it is represented by the linguistic dictionaries, we have to decide which attributes are both suitable and worthwhile to be materialized.
- Assumptions on *incomplete knowledge*: if an attribute is materialized we have to decide whether query evaluation should be done under the usual closed world assumption or a partial open world assumption is more appropriate.
- *Refreshment* strategies(out of the scope of this paper): if the external multimedia items, from which the materialized attribute values are computed, can be updated, then the materializations must eventually be updated also.

Materialization and assumptions on *incomplete knowledge* are decided on the basis of the type of an attribute. The full type system of our mediator is outlined in Section 6 below. Here we only need the distinctions between a scalar and a setvalued attribute, and between a structured and a semi-structured attribute. For a scalar attribute, we assume that the standard procedures are for reading and writing the scalar value, as already used in the previous discussions. For a setvalued attribute, instead we suppose that the standard procedures do not deal with the whole set but only with single members, i.e. they have the parametrisied form *is_member_A(x)*, *insert_A(x)*, and *delete_A(x)*.

A *scalar and structured* attribute is always materialized. If there is a stored value, then any reference to such an attribute is evaluated solely on the basis of this materialization, i.e. we use the usual closed world assumption.

A *setvalued and structured* attribute is materialized if the cardinality of the set of possible member values is a small finite cardinal. This situation is recognized by inspecting all components of the structure whether their possible values are bound by a small, explicitly enumerated range. Typically, such an enumeration is given by a finite subrange of cardinals or by a finite collection of constants . In this case, any materialization of the set is interpreted under a partial open world assumption: the stored values are supposed to represent definitely true statements, while the truth of statements corresponding to values that are possible but not stored in the set is assumed to be unknown. As a consequence, any query explicitly or implicitly referring to values of the latter kind must be evaluated by reinspecting the original multimedia item.

A *scalar and semi-structured* attribute is always materialized. A reference to a subcomponent that is included in the actually stored self description (see Section 6 for more details) is evaluated solely on the basis of this materialization. A reference to a subcomponent that may occur according to the semi-structured type but is not included in the actually stored self description is again treated under a partial open world assumption. And thus the reference must be evaluated by reinspecting the original multimedia item.

For a *setvalued and semi-structured* attribute we don´t have a general rule as yet. We expect that such an attribute is typically used for feature extracting from images or other highly complex and semantically rich multimedia items. Presently, we suggest to decide on materializations of such attributes case by case on the basis of the application semantics, rather than on more syntactically oriented reasoning as for the other three types of attributes. If the attribute is materialized, it is again treated under the partial open world assumptions sketched above.

Obviously, the cases and our standard decisions for materializations possibly need to be refined and revised according to a specific application. Therefore, our treatment should be merely taken as a default. Furthermore, our approaches to query evaluation must be suitably adapted to deal with our decisions on incomplete knowledge.

Persistent materialization of attribute values offers the option that the multimedia mediator uses its proxy state to *cumulatively learn* more and more aspects of its application field. This option is particularly powerful in combination with semi-structured attributes: learning may enhance the knowledge about both the *structure* and the *content* of multimedia items. We can also speed-up the learning rate by running specially chosen queries in times where the mediator and the external agents would be idle otherwise. The only purpose of such queries is to result in new materializations.

6 Dealing with semi-structured multimedia items

Traditional database management systems deal with structured data only. Accordingly, for that kind of systems we can achieve a clear separation of raw data and the structural description of raw data. This distinction is accomplished by sharply distinguishing between the time-varying database instance, the raw data, and the time-independent database schema, the structural description of the raw data. If data is only semi-structured, then, at most, only a part of its structural description can be time-independently declared in the schema, while any single data unit has to individually carry the description of its particular structure on the instance level. Then, of course, the particular structures on the instance level have to conform to the partial structural description on the schema level. Thus the partial structural description on the schema level is taken as an invariant to be maintained under update operations, like for traditional semantic constraints.

In order to formalize this outline, we first reconsider an ODMG like type system. Such a system is built from some atomic types, say **boolean**, **integer** and **string**, by applying the type constructors **n_tuple_of**, for all cardinals *n* (omitting the attribute identifiers for the sake of succint presentation), **set_of**, **reference_to** (and possibly some further related ones which are not treated here). Any type σ is described by its syntax tree, where the leaf nodes denote the atomic types involved and the nonleaf nodes denote the use of type constructors. Thus a type σ can be considered as a pair $<$ *temp*, *label* $>$, where *temp* describes the pure graph theoretical structure of the syntax tree and *label* describes the annotations at the nodes of the syntax tree. Of course, the annotation of a nonleaf node must be compatible with the tree structure at this node, i.e. a nonleaf node with *n* subtrees demands for an *n*-ary type constructor.

Next we discuss a (partial) embedding order on types, where type σ_1 is embedded in type σ_2 roughly means, that the values of type σ_1 can be understood as values of type σ_2, and that values of type σ_2 can be queried as if they were of type σ_1. There are three reasons for embeddings:

- The first reason results from containments of atomic types. Here, an atomic type t_1 is contained in an atomic type t_2, $t_1 \leq t_2$, iff the set of values of t_1 can be seen as a subset of the set of values of t_2. For instance we specify that **boolean** ≤ **integer** ≤ **string**, where, for instance, the boolean value *false* is identified with the integer *0*, which in turn is identified with the string "0".
- The second reason results from refinements of constructors. Here a constructor c_2 is a refinement of a constructor c_1, $c_1 \leq c_2$, iff all components of constructor c_1 are also components of constructor c_2. For instance we can specify that for $n \leq m$, **n_tuple_of** ≤ **m_tuple_of**. It should be noted, however, that the n components generated by the tuple constructor **n_tuple_of** can also be simulated by a cascade of tuple constructors each of which may have an arity less than n.
- The third reason, not present in an ODMG like type system, results from refinements of the trees which underly the types. Here a tree $temp_2$ is a refinement of tree $temp_1$, $temp_1 \leq temp_2$, iff there is a mapping i of the nodes of tree $temp_1$ on the nodes of tree $temp_2$ such that i is injective on the set of leaves of $temp_1$ and for each edge (n,m) of tree $temp_1$ there exists a corresponding nonempty path in $temp_2$ connecting the nodes $i(n)$ and $i(m)$. There are two implicit assumptions underlying this definition of refinement. First, omitting an explicit constructor like **scalar**, we implicitly assume that a scalar component is refined by prefixing it with the **set_of** constructor. Second, we assume that any type can be refined by prefixing it with the **reference_to** constructor.

Given the containments of atomic types and the refinements of constructors and of trees, we can define an *embedding order* on types:

$< temp_1, label_1 > \leq < temp_2, label_2 >$:iff
$temp_1 \leq temp_2$ via a leaves-injective mapping i, and $label_1(n) \leq label_2(\, i(n)\,)$, for all nodes n of $temp_1$ such that $label_1(n)$ is not a tuple constructor.

As an example, we consider the type $\sigma_1 = < temp_1, label_1 >$ of Figure 6-1. Its tree structure $temp_1$ consists of a root node and three descendant leaves. And its annotation $label_1$ assigns the constructor **3_tuple_of** to the root and the atomic types **string**, **integer**, **integer** to the three leaves, respectively. This type can be embedded in the more elaborated type $\sigma_2 = <temp_2, label_2>$ of Figure 6-1. Here the tree structure has been refined by adding a fourth edge leaving the root and by substituting the second and the third edge by some nontrivial trees. And the annotation $label_2$ has been adapted as follows. It assigns the constructor **4_tuple_of** to the root which refines the original constructor **3_tuple_of**. The assignment **text** to the first child is understood as a refinement of the original type **string**. The right leaf of the new subtree for the second edge gets the type **integer** as its corresponding node in the original tree. Here we also use the convention that the original single component can be seen as part of a pair as described by the constructor **2_tuple_of** appearing in the new subtree. The assignment **string** to the leaf of the new subtree for the third edge is a refinement of the original type **integer**. Here we also use the assumption that the original scalar component can be embedded into a setvalued component as described by the constructor **set_of** appearing in the new subtree.

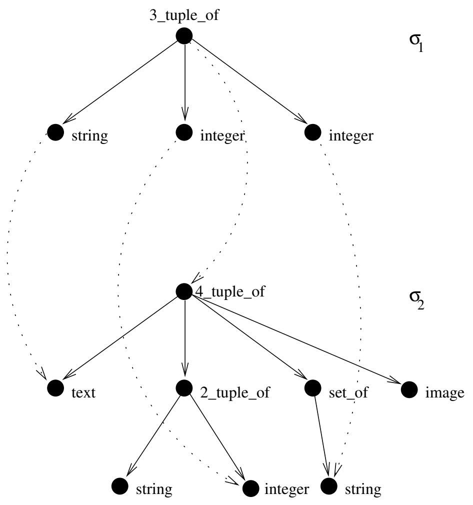

Figure 6-1 Sample embedding of types

For our multimedia mediator we designed an extensible augmentation of an ODMG like type system as follows. We add new atomic types that are relevant for multimedia applications, in particular the atomic types **text** and **image**. On the resulting set of atomic types we define the containments **string** $\leq$ **text** $\leq$ **image**. We could also add new type constructors that are relevant for multimedia applications. In particular, it appears that constructors involving some notion of sequential or parallel time, for instance for so-called "guided tours", are necessary for many applications. This topic, however, is out of the scope of this presentation. On the resulting set of type constructors we would have to define appropriate refinements. If a type constructor is based on time and some timeless types, then tuple constructors may be refined by a sequential time constructor, and a set constructor may be refined by a parallel time constructor.

Finally, we introduce the type constructor **self_describe** with two parameters which is used for multimedia items that are semi-structured as well as self-describing. The first parameter must be a usual type, as for the other type constructors, and it specifies the time-independent minimal constraints on the structure of allowed multimedia items. The second parameter must be a *meta-type* the values of which are types. It specifies which types are allowed as a time-varying self description for the multimedia items that are allowed by the first parameter. The first parameter, a type, and the second parameter, a meta-type, must be *compatible* in the sense that the first parameter can be embedded in all types given by the second parameter. For any type σ occuring as a first parameter, we can use the corresponding meta-type Σ, consisting of all types τ such that σ can be embedded into τ, as a default for the second parameter, meaning that all compatible types are allowed as self description. For each type σ and for each compatible meta-type Π, we define the embedding $\sigma \leq$ **self_describe**(σ, Π).

Considering the example types σ_1 and σ_2 of Figure 6-1 once again, we can specify $\sigma =$ **self_describe**(σ_1, Σ_1) for semi-structured items. Here, Σ_1 is the meta-type corresponding to σ_1, where the type σ_2 is an element of Σ_1. As an application, suppose that the three leaves of σ_1 denote the attributes identified by *name*, *year (of birth), and phone*, respectively. If we declare the type σ for some class, then we require that these three components are present for any item of this class. Accessing them, however, might demand to follow some paths, as for example in σ_2, and to convert the component types appropriately. For that purpose, any specific item of the self-describing type σ carries its particular type $\tau \in \Sigma_1$, together with the leaves-injective mapping *i* that maps the structure of σ to the structure of τ.

7 Comparison with other approaches and conclusion

The paradigm of a mediator has first been advocated by G. Wiederhold [20], and since then it has attracted many researchers, see for instance [18, 19].

Taking advantage of database technology for multimedia applications, in particular using and extending an ODMG like object oriented database management system has also been explored by several projects, see for instance [2,10,11,12,16,17]. The advanced work of [2] is focused on using so-called structuring schemas to neatly represent and efficiently query tree like SGML documents, featuring among others union types and paths. The experimental system of [16] highlights the combination of object management, a user-defined function library and massively parallel computations on very high data volumes. Our emphasis is on representing a partially materialized view on possibly heterogenous and external multimedia sources, featuring both transparent and optimized set oriented access to external sources. In particular, our optimization approach is directed to employ both parallelism, as provided by external multimedia sources, and materialization, as provided by the proxy state.

There are already many studies and proposal on dealing with semi-structure data, see[1] for a recent survey. Our particular proposal for embeddings of types, which includes the new type constructor **self_describe,** combines aspects of previously discussed multimedia types such as in [17], of structuring schemas and tagged union types [2,3], and of graph schemas [8]. Aspects of temporal constructors are treated for instance in [4,6,13,14].

There are also many studies on materialization of database views in general and some suggestions to employ materializations for multimedia systems, see for instance [5,15,16].

Since high efficiency is of great importance, also multimedia query optimization has been studied under various aspects, see for instance [2,5,7,9,19]. One special interest for our multimedia mediator has been to exploit the potentials of parallelism of external agents and its tradeoff with materialization.

Our design of the multimedia mediator is currently started to be implemented in a prototype fashion. The prototype is intended to be used within the HPQS project. The implementation tries to use generally available systems as far as possible. For the database functionality we base the mediator on the O_2 system. For the communication functionality we base the mediator on the CORBA proposal.

There are many avenues for future work. Most importantly, our design of a multimedia mediator has to be evaluated and tuned with respect to performance characteristics. The impact of maintaining a proxy state should be determined also experimentally, in particular the tradeoff between selectivity, which is effected by the materialization strategies, and parallelism of the external agents. Conceptually, query optimization should be studied in more detail, in particular taking into consideration the impact of the extended type system. A depth-in study of a our proposal of the type constructor for semi-structured data would also be worthwhile.

8 References

[1] S. Abiteboul, Querying semi-structured data, Proceedings *6th International Conference on Database Theory*, ICDT ´97, Delphi, Greece, Lecture Notes in Computer Science 1186, Springer, Berlin etc., 1997, pp. 1 - 18.

[2] S. Abiteboul, S.Cluet, V. Christophides, T. Milo, G. Moerkotte, J. Siméon, Querying documents in object databases, Journal of Digital Libraries , to appear.

[3] S. Abiteboul, D. Quass, J.McHugh, J. Widom, J.L. Wiener, The Lorel query language for semistructured data, Journal of Digital Libraries , to appear.

[4] Y. Abiza, A. Leger, M. Crehange, Conceptual modelling for information filtering in broadcast interactive video applications, In: *Multimedia Modeling - Towards the Information Superhighway* (eds: J.P. Courtiat, M. Diaz, P. Sénac), World Scientific, Singapore etc., 1996, pp.35 - 50.

[5] S. Adali, K.S. Candan, Y. Papakonstantinou, V.S. Subramahnian, Query caching and optimization in distributed mediator systems, Proceeding *1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD Record (1996) 25:2, pp. 137-148.

[6] M. Adiba, STORM: an object-oriented multimedia dbms, In: *Multimedia Database Systems* (eds: K.C. Nwosu, B. Thuraisingham, P.B. Berra), Kluwer, 1996, pp. 47 - 88.

[7] E. Bertino, F. Rabitti, S. Gibbs, Query processing in a multimedia document system, ACM Transactions on Office Information Systems (1988) 6:1, pp.1 - 41.

[8] P. Buneman, S. Davidson, M. Fernandez, D. Suciu, Adding structure to unstructured data, Proc. *6th International Conference on Database Theory*, ICDT ´97, Delphi, Greece,Lecture Notes in Computer Science 1186, Springer, Berlin etc.,1997, pp. 336 - 350.

[9] S. Chaudhuri, L. Gravano, Optimizing queries over multimedia repositories, Proceeding *1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, SIGMOD Record (1996) 25:2, pp. 91 - 102.

[10] C.Y.R. Chen, D.S. Meliksetian, M. Cheng-Sheng Chang, L.J. Liu, Design of a multimedia object oriented dbms, Multimedia Systems (1995) 3, pp.217 - 227.

[11] N. Fuhr, Object-oriented and database concepts for the design of networked information retrieval systems, Proceedings *5th International Conference on Information and Knowledge Management*, 1996, pp. 164 - 172.

[12] K. Groenbaek, J.A. Hem, O.L. Madsen, L. Sloth, Cooperative hypermedia systems: a Dexter-based architecture, C. of the ACM (1994) 37:2, pp. 64 - 74.

[13] L. Hardman, D.C.A. Bulterman, G. van Rossum, The Amsterdam hypermedia model: adding time and content to the Dexter model, C. of the ACM (1994) 37:2, pp. 50 - 62.

[14] S. Hibino, E.A. Rundensteiner, A visual multimedia query for temporal analysis of video data, In: *Multimedia Database Systems* (eds: K.C. Nwosu, B. Thuraisingham, P.B. Berra), Kluwer, Boston etc., 1996, pp. 123 - 159.

[15] J.J. Lu, G. Moerkotte, J. Schue, V.S. Subrahmanian, Efficient maintenance of materialized mediated views, Proc. *1995 ACM SIGMOD International Conference on Management of Data*, San Jose, California, SIGMOD Record (1995) 24:2, pp. 340 - 351.

[16] W. O´Connell et al, A teradata content-based multimedia object manager for massively parallel architectures, Proceeding *1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, SIGMOD Record (1996) 25:2, pp. 68 - 78.

[17] M.T. Özsu, D. Szafron, G. El-Medani, C. Vittal, An object-oriented multimedia database system for a news-on-demand application, Multimedia Systems (1995) 3, pp. 183 - 203.

[18] V.S. Subrahmanian, HERMES: a heterogeneous reasoning and mediator system, submitted for publication (see http//www.cs.umd.edu/hermes).

[19] J.D. Ullman, Information integration using logical views, Proceedings *6th International Conference on Database Theory*, ICDT ´97, Delphi, Greece, Lecture Notes in Computer Science 1186, Springer, Berlin etc., 1997, pp. 19 - 40.

[20] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Computer (1992) 25:3, pp. 38 - 49.