IST R&D PROJECT

SHARED-COST RTD PROJECT

THEME: INFORMATION ACCESS AND INTERFACES
COMMISSION OF THE EUROPEAN COMMUNITIES
DIRECTORATE GENERAL DG INFSO

PROJECT OFFICER: DR PAT MANSON

Resource Selection and Data Fusion for Multimedia
International Digital Libraries

Test-bed architecture specification

D1.1

May 28, 2001, UNIDO/WP1/Task 1.1/Version 3

Henrik Nottelmann

Contents

Chapter 1

Definitions

This chapter gives some brief definitions and descriptions of often-used terms (like proxy, dispatcher, document, schema, query). It should make it easier to understand this document.

1.1 Databases, proxies and the dispatcher

MIND combines multiple databases (also called digital libraries, resources), each using an own schema, query language and communication interface. Due to this heterogenity, much database-specific work has to be done, encapsulated in a proxy. A proxy is a representative for one database or for two or more homogeneous databases (which use the same schema, query language and communication interface). Each database can be identified by the proxy and a proxy-internal number. The corresponding pair *(proxy ID, internal ID)* is called the database ID.

The dispatcher controls the query process, calls the proxies when necessary and performs database-independent work.

1.2 Content, schemas and media types

Documents are structured through schemas. A schema is a set of attributes *(name, data type)*. A document is a set of values, one value for each attribute corresponding to the data type. A data type defines the set of valid values, the set of valid predicates and the media type (text, fact, image or speech).

Attributes are used for two purposes: to model metadata (e.g., title, author, publishing year) and to contain the document content. Metadata attributes have the media type "fact", content attributes have the media type "text", "image" or "speech". Of course, a document can contain more than one content attribute with different media types as well.

If the schema wants to allow for partial documents (e.g., a chapter of a book), it should provide an additional metadata attribute (`is-part`) indicating if it is the original value or a part (and if, which part).

A document can be identified by a pair *(ID of the corresponding database, database-internal ID)*, the "document ID" (which is stored in the document). If the database does not support database-internal IDs, this number has to be generated by the proxy. The purpose of this ID is to retrieve a document from the result set when the query process only returns metadata or summaries (see below). Then, the proxy has to cache the result documents.

Each document has associated the corresponding schema.

1.3 Queries

There are two kinds of queries in MIND:

- user queries which use a standard schema, and
- proprietary queries which are database-specific.

For the user and the dispatcher, only user queries are visible. The probabilistic mapping from user to propietary queries (called "query transformation") is performed by the proxies and is completely transparent to the dispatcher.

A user query contains

- the number n of relevant documents to retrieve,
- a set of conditions *(type, weight, attribute, predicate, value, global condition information)*, where *type* indicates if this condition is mandatory, prohibitive or optional, and *global condition information* is a set of pairs *(name, value)* containing global values (e.g., idf for text attributes) for this condition,
- relevance feedback data, i.e. a set of pairs *(document ID, relevance label in* $[0, 1]$*)*,
- the databases (identified by their IDs) to be considered,
- user-specified cost parameters, and
- global condition-independent information, i.e. a set of pairs *(name, value)* (e.g., the number of documents summed over all databases).

Global information will be collected in the first query step ("query modification") and is used by resource selection and data fusion.

There are two syntactical difference between proprietary and user queries:

1. Proprietary queries don't contain databases to consider.
2. Conditions in proprietary queries also have a precision value indicating how precise the transformation of the corresponding user condition into this proprietary condition is. These precision values are available at all other proxy components.

1.4 Result

The result of a query process is a list of pairs *(document, original RSV from database, new RSV)* (both weights may be the same).

The "result type" (given by the dispatcher) determines the actual content of the content attributes of the result documents. They can be

- empty (i.e., the document contains only metadata),
- short summaries of the actual content (produced by the proxy) or
- complete.

The rationale is that returning all complete documents to the user is often too expensive. Thus, possibilities to reduce the data volume are integrated.

It is possible to retrieve the complete document to a result document (specified by the document ID) with empty or summary content attributes. If the corresponding database does not support such queries, the proxy can cache the complete result documents for this querying based on the document ID.

Chapter 2

Survey

This chapter gives a brief survey of MIND. MIND consists of two parts:

1. the query process ("Using MIND" in the Technical Annex) which is visible to the user and
2. resource gathering ("Preparing MIND" in the Technical Annex) which is invisible to the user.

2.1 Query process

This section gives a rough overview over the query process. Details are described in chapter 4.

The query process is organized as followed:

1. The user enters a user query. In the initial run, the relevance feedback data is empty. In the subsequent runs, the relevance feedback data can contain documents with a relevance label in $[0, 1]$. This label can be explicitly entered by the user or calculated by the user interface (e.g., from a binary relevance judgement and some other measures). The relevance feedback data also can contain pseudo relevance feedback data (first retrieved documents chosen). The cost parameters are determined by the user interface, e.g. by user input or calculation based on user interfaces.
2. The first task is to modify the user query w. r. t. the given relevance data. This task has two subtasks:
   (a) (Condition-dependent and condition-independent) global information values have to be computed and stored in the query. The proxies support this subtask by generating database-specific features ("global information features") and sending them to the dispatcher. For text conditions, a global information feature can contain the df values w. r. t. the database (i.e., the number of documents in that database containing the condition term). The dispatcher collects the global information values and uses them to create the actual global values. Of course, these values can be used to modify the query conditions, too.
   (b) Furthermore, relevance feedback data can be used to modify the query (e.g., to add new conditions or reweight the original query conditions). The proxies once more support this subtask by generating database-specific features ("relevance feedback features") and sending them to the dispatcher, which modifies the user query.
3. Second, the dispatcher determines an optimal resource selection, i.e. the number of documents to retrieve from each resource so that the expected overall cost is minimized. Each proxy has to estimate the expected costs retrieving s relevant documents ($1 \leq s \leq n$). As there might

be duplicates in the retrieval results (so that the user will not get as many documents as he requested), the dispatcher might overestimate the number of documents of some databases w. r. t. statistical document similarities. It is possible to let the user change the resource selection.

4. Forth, the dispatcher asks each proxy to query its database to retrieve the calculated number of documents ("database query run"). The result is a list of sets of pairs (document, weight).

5. The fifth task is data fusion. For every document, the corresponding proxy calculates a new RSV based ont the original RSV to consider global knowledge (e.g., global idf values). Furthermore, a short summary for each content attribute has to be generated (if this result type is chosen). The dispatcher then tries to detect duplicate documents in the several result lists. Duplicates are eliminated. At last, the dispatcher reranks the documents w. r. t. the new RSV weights.

6. If the dispatcher detected and eliminated duplicates, the merged result list might contain fewer documents than requested by the user. If this happens, the query process has to be repeated (starting with resource selection) with a higher number of documents to be retrieved.

7. The user interface (not yet considered in the proposed model) displays the result (metadata, content and RSV). The user can select an entry to retrieve a summary or the full document from the corresponding proxy (depending on the chosen result type).

2.2 Resource gathering

Resource gathering is the process which creates or updates a resource description. Resource gathering is perfomed as follows:

Every proxy periodically calls itself for resource gathering. This is invisible for the dispatcher and user interface. The proxy applies query-based sampling: In a loop, it calls its database with a constructed query. The result is used to update the resource description. This loop is to be continued until a stopping criteria is reached. E.g., this can be "m documents returned by the database" or "q queries send to the database".

Chapter 3

Structure

This chapter gives a brief overview over the static aspects of MIND: the layers of the system, the class structure and the user interface.

3.1 Layers

MIND is organized in four layers (see figure 3.1):

1. On the bottom (layer 0), there are the (existing) databases.
2. Layer 1 is formed by the proxies. Each proxy represents in general a group of homogeneous databases (same schema, same query language, same communication interface). Every database-specific work has to be done by the proxies.
3. Layer 2 contains the dispatcher. The dispatcher controls the query process, calls the proxies and perfoms database-independent work.
4. The user interface forms layer 3. The user interface is not described in this document as it is part of later work (WP 6.1). It can be a stand-alone application or a web-based interface.

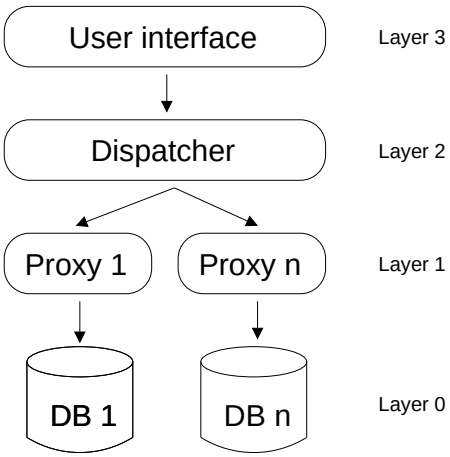

Figure 3.1: MIND layers

3.2 Class architecture

The architecture contains the following classes:

- `DLProxy` and the associated classes model the representative of one (or more homogeneous) databases to allow database-dependencies.
- The class `Dispatcher` and the associated class `DispatcherMediaComponent` (for media-specific work; there is one subclass for every media type) controls the query process. For this work, the dispatcher repeatedly calls the proxies to delegate database-specific subtasks.
- `DLInterface` is a wrapper for the database, allowing for accessing the database using an unique query syntax and an unique communication protocol for the proxies. The `DLInterface` transforms the proprietary query (which uses the schema of the database but still an unique query syntax) into a query which can be understood by the database, and calls the database using its proprietary communication protocol.
- Except for the database query run, each task has database- and media-specific parts which are carried out in `DLMediaComponent`.

  Once more, there is one subclass for every media type. It is clear that database-specific (anonymous) subclasses of this classes can be created and attached to a proxy.
- `DLProxy` and `DLMediaComponent` both need resource descriptions for their work und thus contain an attribute `resourceDescription`.

For details, see figures 3.2 and 3.3.

3.3 User Interface

The user interface is not yet considered in this model. It communicates with the dispatcher through XML, thus there could be multiple user interfaces.

One simple solution is just to provide a formular-based WWW-Interface.

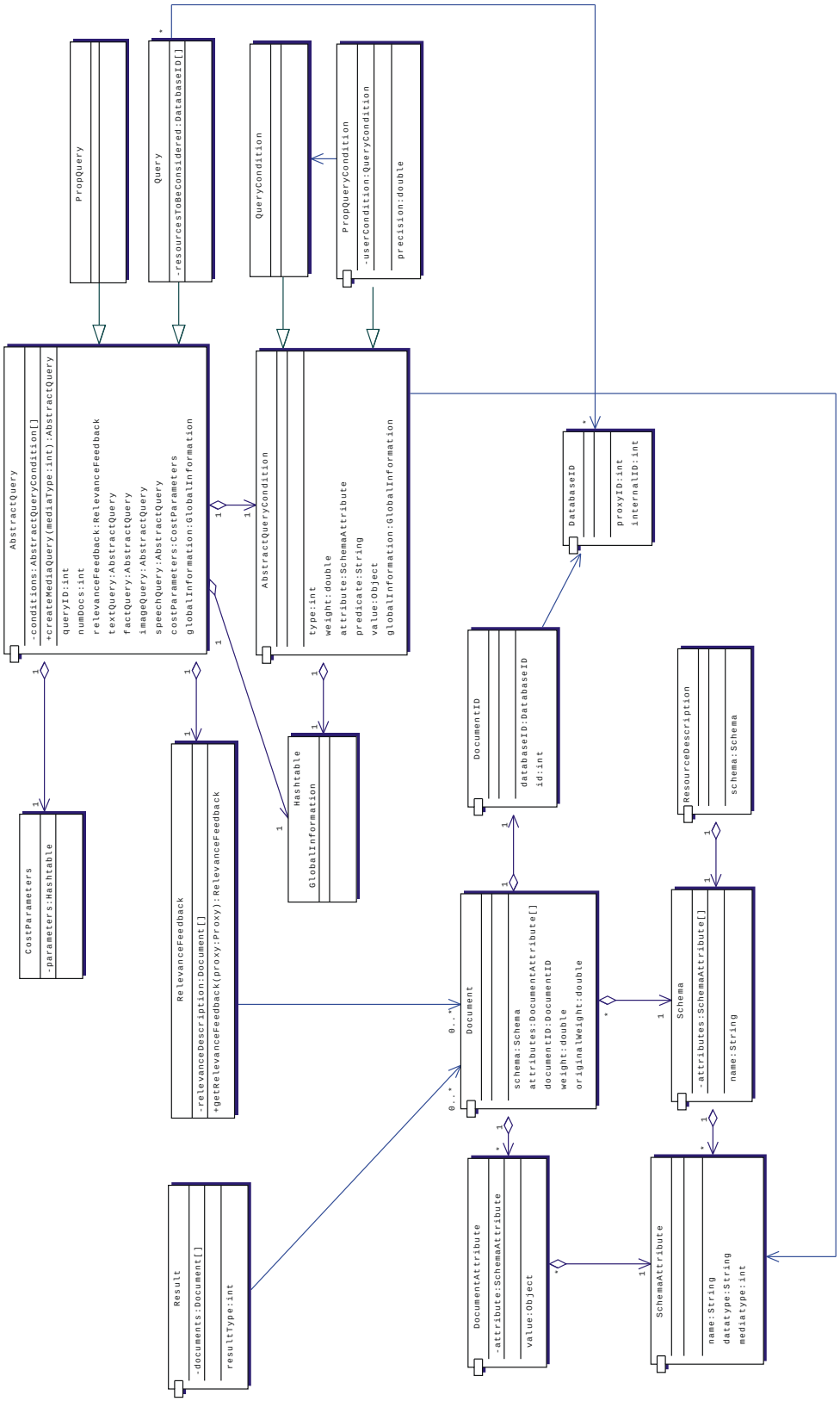

Figure 3.2: Query and document

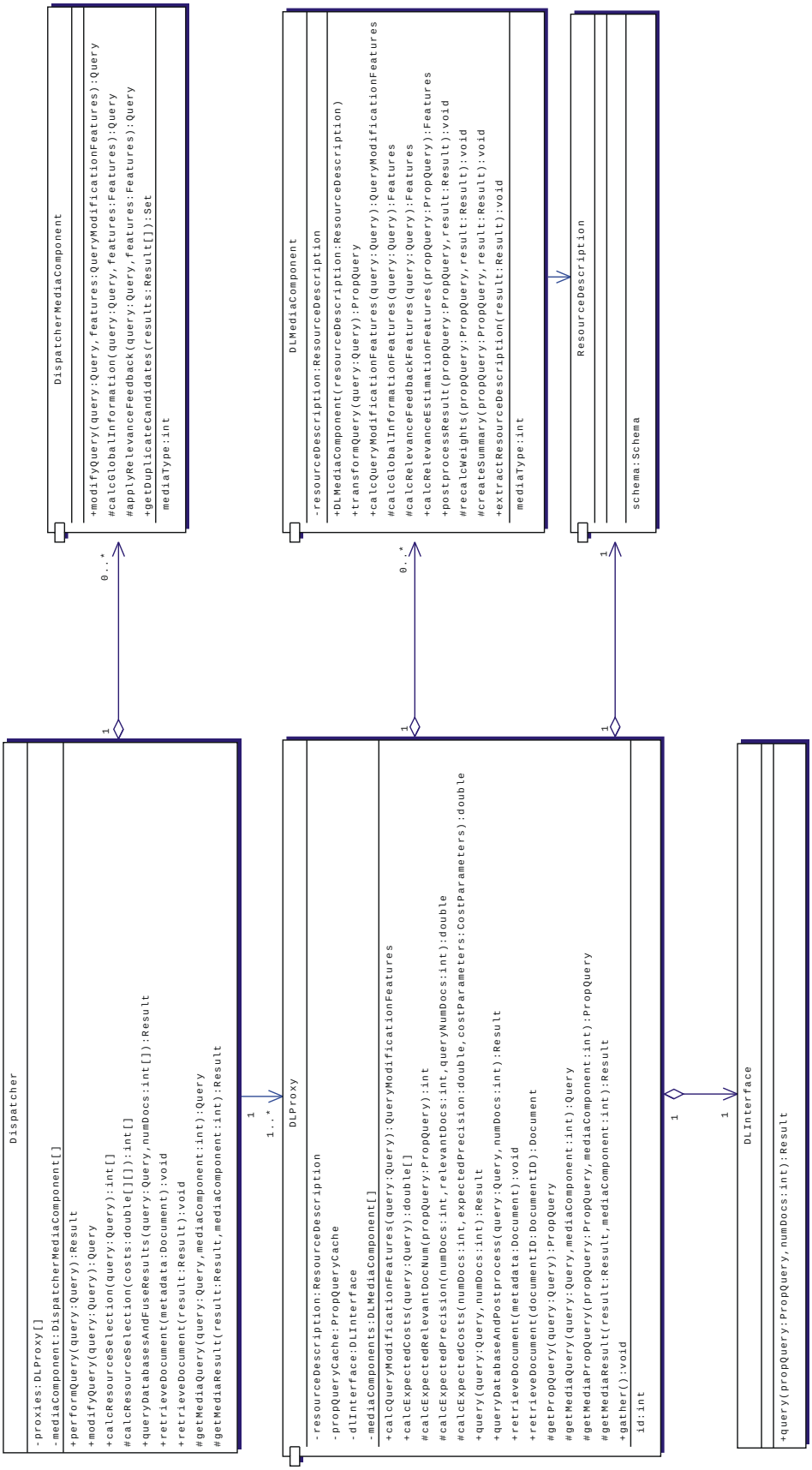

Figure 3.3: Dispatcher (layer 2), proxy (layer 1), database (layer 0)

Chapter 4

Query process

This chapter describes the tasks of the query process (including query transformation which is transparent to the dispatcher) in detail.

4.1 Query transformation

As the databases use different query languages and different schemas, user queries have to be transformed into database-specific "proprietary" queries. This transformation is done by the proxy when necessary (figure 4.1), and is completely transparent to `Dispatcher` (and, thus, to the user).

Each single condition of the user query has to be transformed into a set of proprietary conditions (and the precision of the transformation for that proprietary condition) relating to the database query language and the database schema (and possibly database-specific attribute domains). It is also possible to map from one media type into another. The union of these proprietary conditions forms the proprietary query.

Each subclass of `DLMediaComponent` transforms the conditions according to its media type. The results are collected in `DLProxy` to form the proprietary query.

4.2 Query modification

This task modifies the original user query, partially using relevance feedback data if available. This is done in the following four steps (see figure 4.2):

1. Each proxy generates
   - global information features (e.g., df values for the terms in the user query) and
   - relevance feedback features.

   The features can be generated in a media-specific way using the subclasses of `DLMediaComponent`.
2. These features (both stored in an instance of `QueryModificationFeatures`) are collected in the dispatcher.
3. Subclasses of `DispatcherMediaComponent` then generate global information values (e.g., global idf values) and store them in the user query.
4. Furthermore, subclasses of `DispatcherMediaComponent` can add new conditions or change the original condition weights based on the collected relevance feedback features.

Figure 4.1: Query transformation

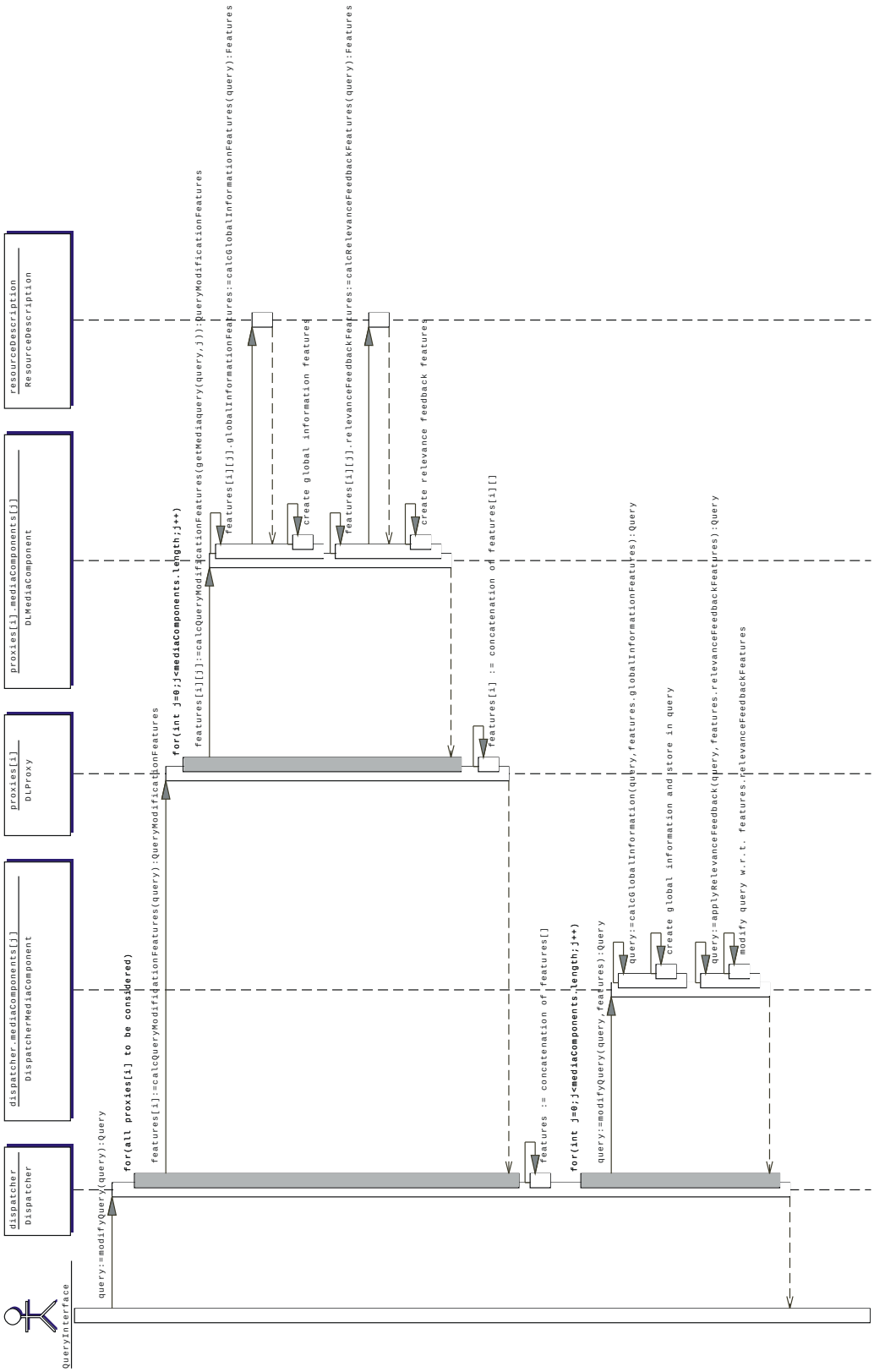

Figure 4.2: Query modification

4.3 Resource selection

Resource selection is performed in four steps (see figure 4.3):

1. First, each proxy estimates the expected number of relevant documents in its database w. r. t. the proprietary query:
   (a) Relevance estimation features are generated by subclasses of `DLMediaComponent`. It is an open question how these features look like, and how they are generated (except that the global information collected in the query modification step can be used).
   (b) With these features, `DLProxy` estimates the expected number of relevant documents in the database $E(rel|q, D)$ (using relevance feedback data if available).
2. `DLProxy` calculates the expected precision w. r. t. the recall, assuming a linear recall-precision-function. Database-specific subclasses can be used to implement other recall-precision-function as well.
3. The last database-specific step (done in `DLProxy`, too) is to estimate the costs to retrieve s documents ($1 \leq s \leq n$) from the database, using the expected precision, database-specific cost parameters retrieved from the resource description and user-specific cost parameters stored in the query.
4. The dispatcher determines an optimum selection, i.e. the number of documents to retrieve from each database which minimizes the expected overall costs.

Resource selection should consider similarities between databases (collections overlaps) and use them to overestimate the number of documents to retrieve (to reduce the problem with too few documents in the result list as described in section 4.5). It is not yet clear how this can be integrated into this model, and is left out in the diagram 4.3.

4.4 Database query run

In the database query run, each proxy sends its proprietary query and the number of documents to retrieve to the database (via `DLInterface`). The proxy receives a list of documents (metadata only, summaries or complete documents), each with an attached RSV. It returns a set of pairs *(document, original RSV from the database)*.

`DLInterface` works as a wrapper of the database providing phyiscal independence. It is clear that there is one subclass of `DLInterface` for each proxy.

The result of the database query run is a list of sets of pairs *(document, original RSV from the database)*, one set for each proxy. Of course, each document has associated the ID of the database it is derived from.

Database query run and data fusion are integrated to improve efficiency: The dispatcher calls every proxy to query the database, to calculate a new result weight (based on the original RSV) and to create a summary in one call. The dispatcher then performs the database-independent part of the data fusion work.

4.5 Data Fusion

Data fusion (figure 4.4) has three goals:

1. to calculate a new RSV for each document to consider global knowledge (e.g., global idf values), leading to a new ranking,

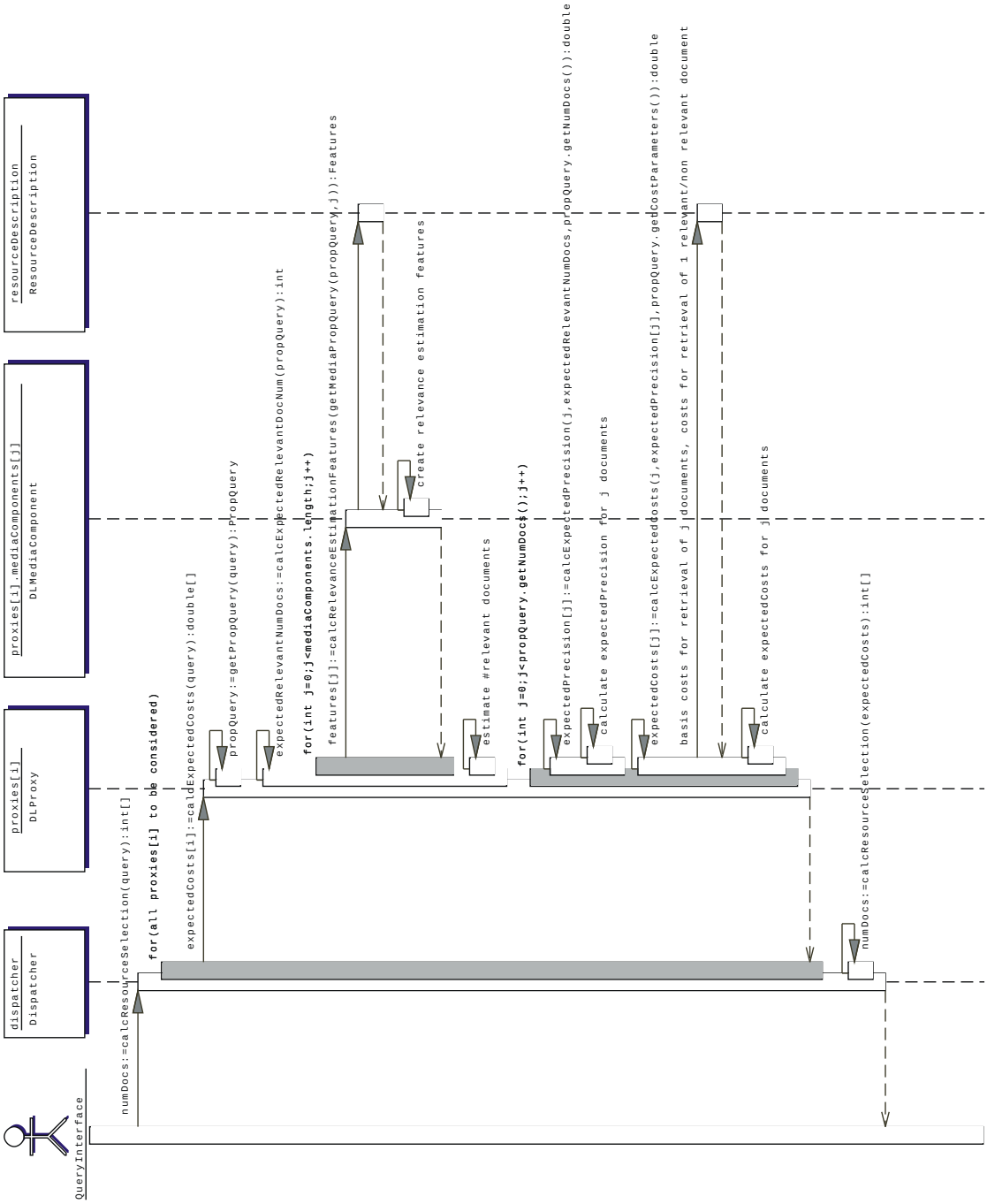

Figure 4.3: Resource selection

2. to detect and elimate duplicate documents in the result sets, and

3. to generate a show document summary for each document content attribute (if desired by the dispatcher).

Data fusion is called with the query, the result of the database query run, i.e. a list of sets of pairs *(document, RSV)* and the desired result type (as mentioned above, data fusion and database query run are integrated):

1. `DLProxy` performs these two actions with its result set:

   (a) Subclasses of `DLMediaComponent` are called to calculate a new result weight (based on the original RSV) w. r. t. to the attributes of their media type. E.g., this can be done by a standard retrieval run based on the database result, or a weighting can be applied. The global information values stored in the query can be used for this step. It is also possible to filter the result set (e.g., if the database operator is less precise than required). Also, documents can be dropped (e.g., when the new RSV is (nearly) zero).

   (b) Furthermore, the subclasses of `DLMediaComponent` create a short summary of each content attribute if the dispatcher requires summaries for content attributes.

2. Then dispatcher then has to detect duplicates. This can be done based on the database-internal document ID or on content equality (e.g., with same title attributes, bitweise equality for images or other more enhanced methods) or similarity (e.g., the case of letters can be ignored, or slight differences in images can be ignored). The subclasses of `DispatcherMediaComponent` try to detect candidates for duplicates (w. r. t. the attributes of their media type), group these candidates and return them as a set of candidate groups (this is also a set). `Dispatcher` intersects these sets and eliminates duplicates from the result set.

3. Afterwards, the sets from the result list are merged and reranked w. r. t. the new RSV.

Duplicate detection and elimination may lead to a shorter result list than requested by the user. Then, another query process run has to be initiated (starting with resource selection).

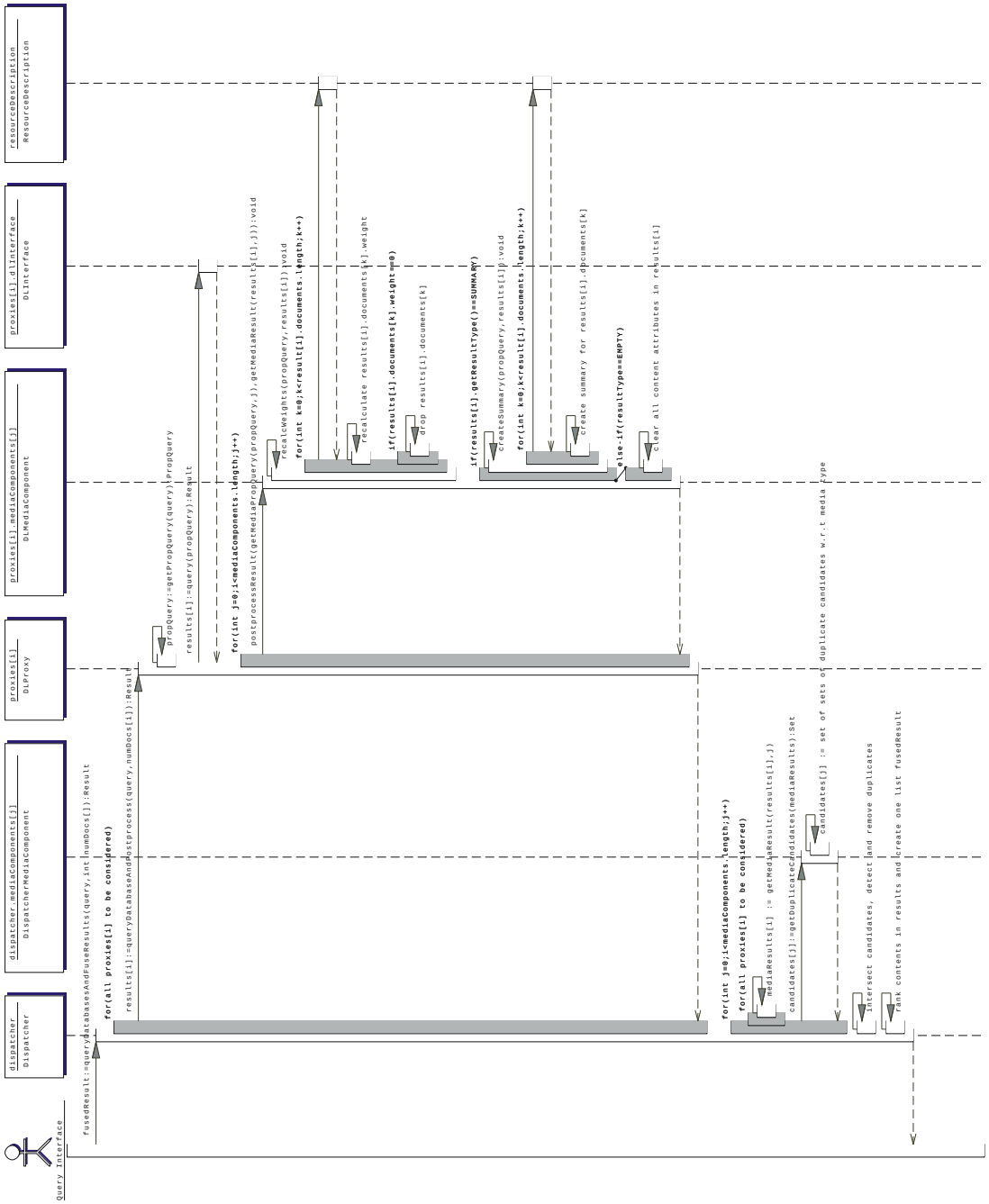

Figure 4.4: Data fusion

Chapter 5

Resource gathering

Resource gathering is the process which creates or updates a resource description. This must be done periodically, because important characteristics of a database stored in the resource description change over time.

Thus, every proxy must update its resource description from time to time. This should be done independently from other proxies. Thus, every proxy chooses a time interval and calls itself for resource gathering periodically. Therefore, resource gathering is invisible for the dispatcher and user interface.

For resource gathering, query based sampling is applied. In a loop, the following steps are performed:

1. The proxy constructs a proprietary query:
   - The initial query can be a randomly chosen condition.
   - Subsequent queries use the already learning resource description.
2. The proxy sends the query to `DLInterface` to retrieve documents. The proxy has to decide how many documents should be returned by the database.
3. Subclasses of `DLMediaComponent` are called to extract resource description features (e.g., document frequencies) and to update the resource description.

This loop is to be continued until a stopping criteria is reached. E.g., this can be "m documents returned by the database" or "q queries send to the database".

For resource gathering, also see figure 5.1.

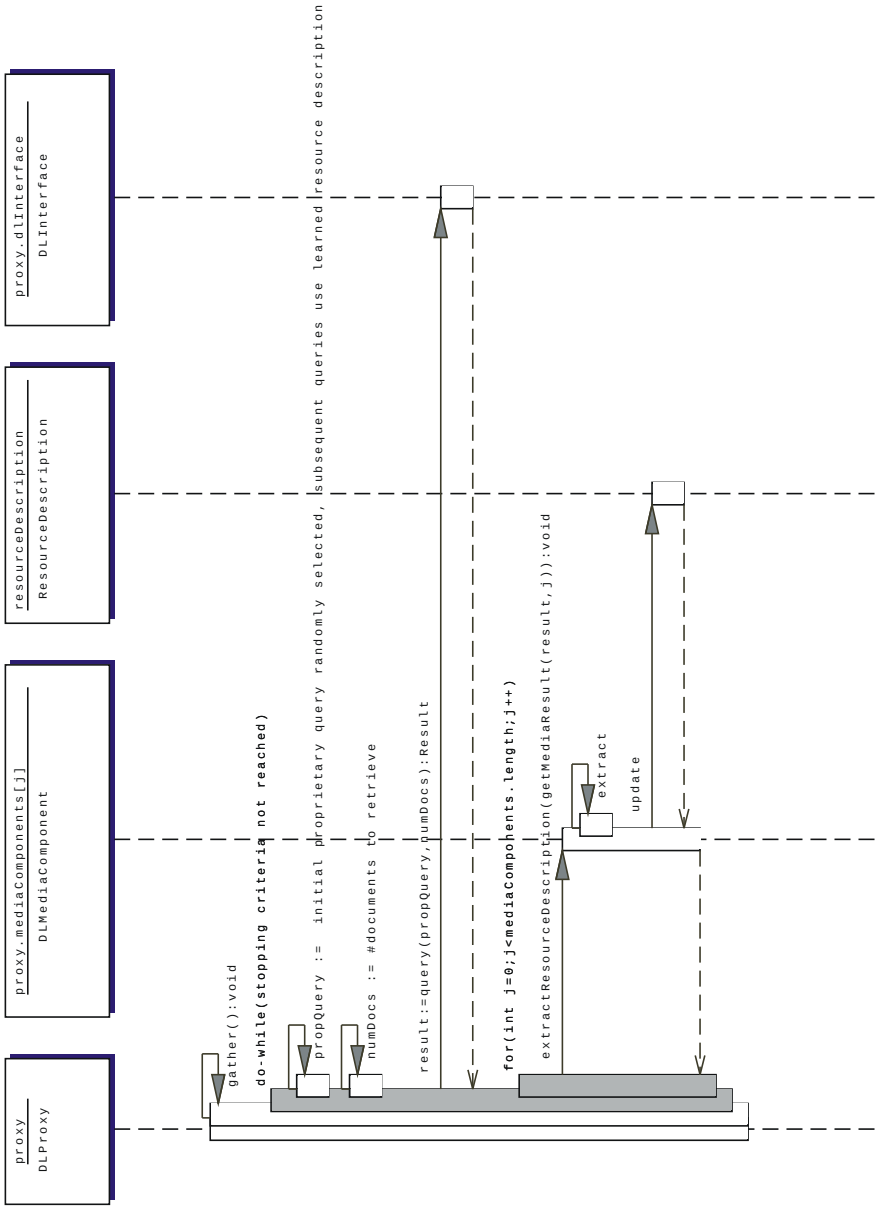

Figure 5.1: Resource gathering

Chapter 6

Programming infrastructure

This chapter covers infrastructure issues (which are apart from the "pure" model) like the development platform, the language to use, the physical distribution and the communication protocols.

6.1 Physical distribution of the components

MIND is a distributed System:

- The dispatcher runs on one single central machine.
- Proxies (i.e., database-specific subclasses of `DLProxy`) can run on different computers. The dispatcher has to know where the proxies are located.
- The media-specific parts of the dispatcher and proxies are distributed, too. Every part w. r. t. text is done on the systems located at the partner who is responsible for text. The three other media types are handled in the same way.

One example is depicted in figure 6.1. There, a rectangle contains all modules running on one computer. "Speech 1" is the speech part of "Proxy 1" (speech-specific subclass of `DLMediaComponent`)., and "Speech" is the speech part of the dispatcher (speech-specific subclass of `DispatcherMediaComponent`). The arrows denote the communication between the components.

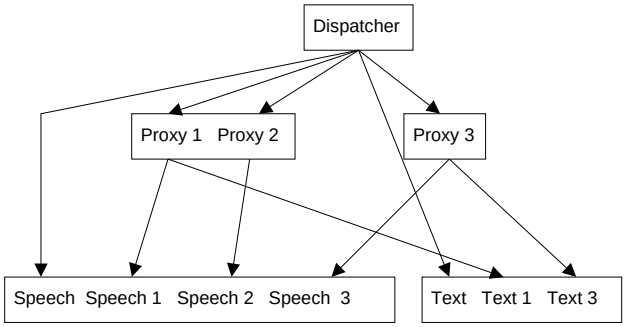

Figure 6.1: Distribution of the MIND components

This solution eases the integration of modules developed by the different partner. Furthermore, it is possible to use media-specific hard- and software in only one place.

6.2 Platforms and programming languages

As described in section 6.1, the media-dependent parts of MIND are distributed on several computers. Furthermore, the proxys (the media-independent parts of them) are distributed as well. Communication is done through XML as described below. Thus, it is possible to use multiple platforms and multiple programming languages.

The major platform and major programming language (e.g., the class framework used for the parts which do not depend on a specific database) are developed under Linux using Java.

6.3 Communication protocol

Communication inside the components is done using simple method calls (and objects). The several components of MIND communicate through XML. This eases testing, debugging, changing the communication interface and distributing this components on different computers. Furthermore, it make it very easy to exchange compontents or to add a new user interface. XML communication is not covered in the model diagrams to keep them simple and easy to read.

To embed binary data (images and speech) in XML, MIME (Multipurpose Internet Mail Extensions) is used. MIME is specified by the IETF (Internet Engineering Task Force) in RFC 1521 (`http://www.ietf.org/rfc/rfc1521.txt`). Binary data like images or audio are mainly encoded in "base64" (see section 5.2 of the RFC). In base64, the encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters, leading to an overhead of 33%. The result string can be embedded in an XML as follows:

```
<attribute result-type="summary">
   <name>name</name>
   <value mime="true" content-type="image/jpeg" content-transfer-encoding="base64">
     base64 encoded image data here
   </value>
</attribute>
```

This example shows the definition of an attribute in a document.

Of course, other MIME types like `image/gif`, `audio/basic` or `text/plain` are possible as well.

Texts can be encoded as "8bit" or "Quoted-printable" to avoid the overhead of "base64".

Communication between the components is done via TCP/IP. The proxies have to register themselves at the dispatcher (specifying their IP addresses and port number). The media-dependent parts of the proxies have to register, too. They have to call the dispatcher specifiying their IP addresses, port number, the proxy ID and the media type. The dispatcher then passes this to the corresponding proxy who has to store these information so that it can later call the component. The media-dependent parts of the dispatcher have to register at the dispatcher specifiying their IP addresses, port number and the media type.

List of Figures