Project Number:	IST-2001-33570
Project Acronym:	INFOMIX
Project Full Name:	Infomix: Boosting the Information Integration
Deliverable Number:	D6.2
Title:	**Methods for data acquisition and transformation**
Workpackage:	WP6
Document Type:	Deliverable
Distribution:	INFOMIX Consortium
Status:	Final
Document file:	WP6 T2 D6.2 0202.pdf
Version:	1.0
Number of pages:	33
Due Date:	December 31, 2003
Delivery Date:	December 22, 2003
Partners contributed:	TUWIEN, RODAN
Partners owning:	TUWIEN
Man Months:	6

Short Description:

The purpose of this document is to describe methods for homogeneous access to data residing in information sources to be integrated, and to select and adapt available research results for their realization from the fields of information integration and agent technology. The data in the sources are given in the raw data formats specified in Project Report D6.1, and have to be transformed into an appropriate format for internal integration use. In the course of this, the INFOMIX Source Data Format (ISDF), which provides a uniform logical format of the source data to the user, has to be taken into account, as well as an internal integration data format, which is the one used by the internal integration algorithms. Furthermore, the usage of methods and techniques for information extraction from implicit representation in this framework will be respected.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	September 4, 2003	First draft
v.1.1	September 13, 2003	Amalgamate Warsaw minutes
v.1.2	October 6, 2003	Add RODAN input
v.1.3	November 4, 2003	Introduce IGDF, IIDF
v.1.4	November 14, 2003	Sketch data cleaning
v.1.5	November 15, 2003	Solidify wrapper descriptions
v.1.6	December 2, 2003	Detail data cleaning
v.1.7	December 2, 2003	Detail raw data to ISDF transformations
v.1.8	December 3, 2003	Detail caching and data transfer
v.2.0	December 4, 2003	Detail ISDF to IIDF transformation
v.2.1	December 15, 2003	Incorporate comments by UNIROME and UNICAL
v.2.2	December 16, 2003	Incorporate design input by RODAN

Contents

1 Introduction

The INFOMIX project is aimed at exploiting advanced reasoning capabilities in order to provide a set of techniques and tools for next-generation information integration. An important aspect of this problem is that data and information is in general not available in a homogeneous format, but rather in different formats, which range from structured formats such as relational data or object-oriented data formats, over semi-structured formats like XML to completely unstructured data in form of plain text. The INFOMIX information integration system shall be able to deal with different data formats and support their logical integration. To this end, the INFOMIX system (D2.2) has been equipped with a conceptually layered architecture, in which the lowest level, the Data Acquisition and Transformation Layer (DAT Layer), provides "low-level" access to the data sources which should be integrated (see Figure 1).

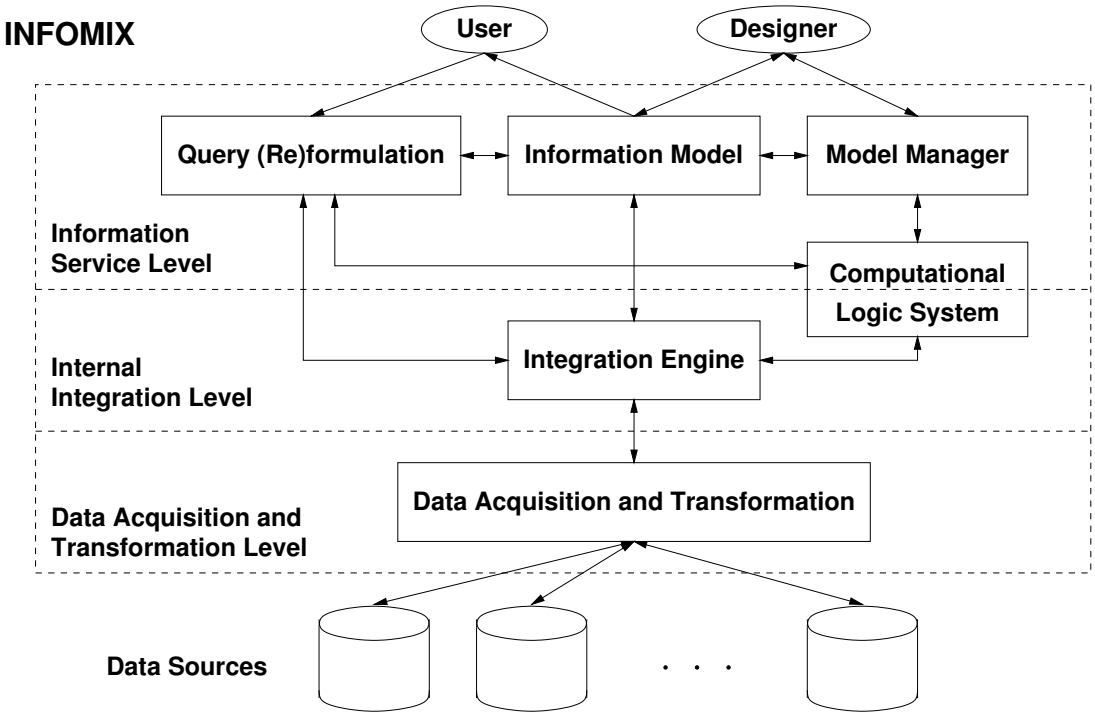

Figure 1: Conceptual layers of the INFOMIX information integration system.

Objectives. The aim of this document is to specify the methods and means for realizing data acquisition and transformation at the DAT Layer of the INFOMIX system ("Data Wrapping"), following the functionality given in Project Report D2.2 (INFOMIX Architecture) and building upon Project Report D6.1 (Heterogeneous Data Source Type Description), where the different "raw" data formats to be handled are described. Raw source data have to be transformed into the logical INFOMIX Source Data Format (ISDF) as well as into the Internal Integration Data Format (IIDF), which has been presupposed

in Project Report D4.2 (Algorithms and Implementation Techniques for the Internal Integration Formalism) and will be defined in Section 4.2 of this report. Besides these basic functionalities, different methods for specifying and generating wrappers for the DAT layer by the user, respectively the system administrator, will be described. This document will serve as a basis for the implementation of the DAT Layer (Work Package WP7.3).

Relations to other documents. This report is directly related to the documents D1.3 (Review of Techniques and Systems for Acquisition and Transformation of Heterogenous Data), D2.1 (Functional Specification of the INFOMIX System), D2.2 (INFOMIX System Architecture), D4.2 (Algorithms and Implementation Techniques for the Internal Integration Formalism), D6.2 (Heterogeneous Data Source Type Description). It is furthermore relevant for the implementation of the INFOMIX prototype, in particular for the Information Integration Management Layer prototype (Deliverable 7.1) and for the Data Acquisition Layer prototype (Deliverable 7.3).

Intended audience and usage guidelines. This report is accessible to the public.

Notational conventions. Nothing particular.

2 Preliminaries

In this section, we briefly recall some conclusions and findings from previous documents of the INFOMIX project which are relevant to this document.

2.1 DAT functionality and architecture

The functionality of the DAT Layer, and a rough architecture, has been described in the INFOMIX system architecture (D2.1), in which the Design-Time and the Run-Time architecture of the system has been described, shown in the Figures 2 and 3.

The DAT Layer is, since it is a conceptual layer, not explicitly outlined in this functional system architecture.

The data acquisition and transformation in INFOMIX will be realized through wrappers, which retrieve the data from the sources in the raw data format and transform them into the logical source format, ISDF, as well as into the internal integration format, IIDF. The particular conversion methods from raw format to ISDF and from ISDF to IIDF (resp., from raw format to IIDF directly) will be dealt with in Section 4.

The Design-Time part, however, comprises the *Wrapper Generation Interface* and the *Wrapper Generator*, while the Run-Time part comprises the *Wrapper Executor*. The functionality of the Wrapper Generator and the Wrapper Executor will be recalled and detailed in Sections 3 and 4, respectively.

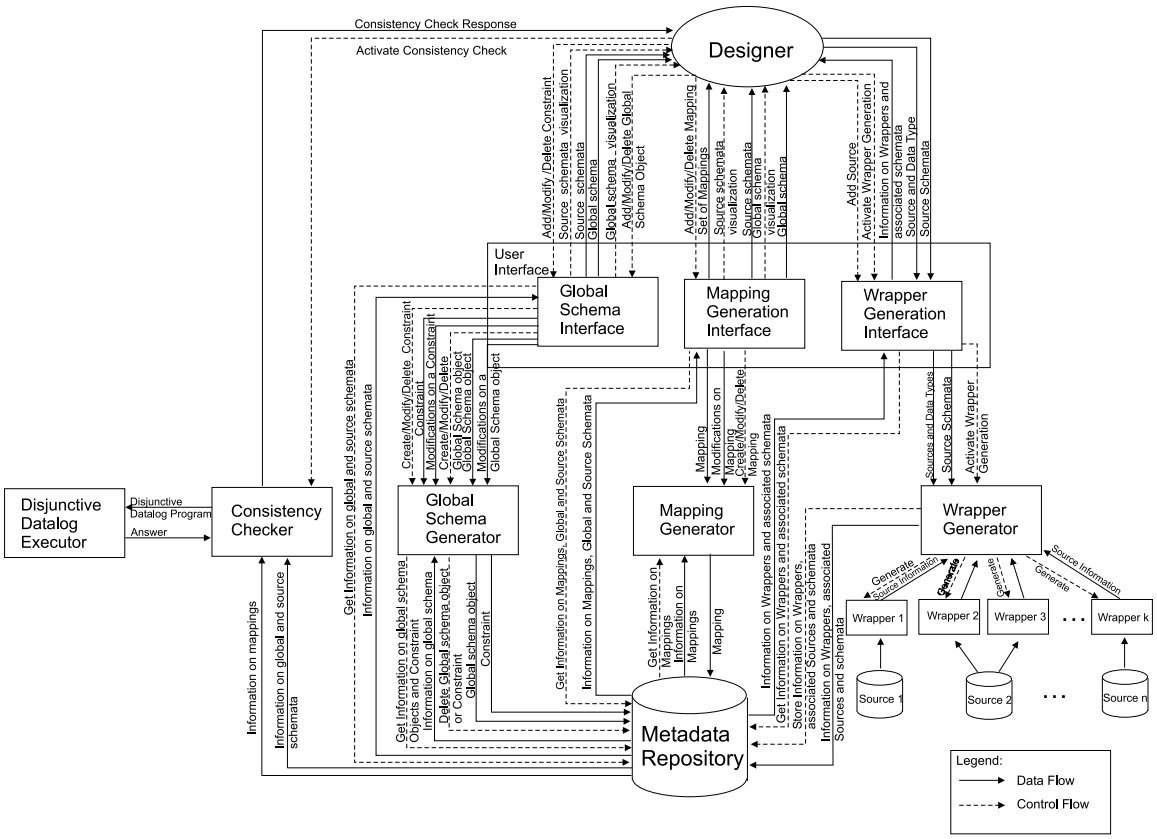

Figure 2: Design-Time Architecture of the INFOMIX system.

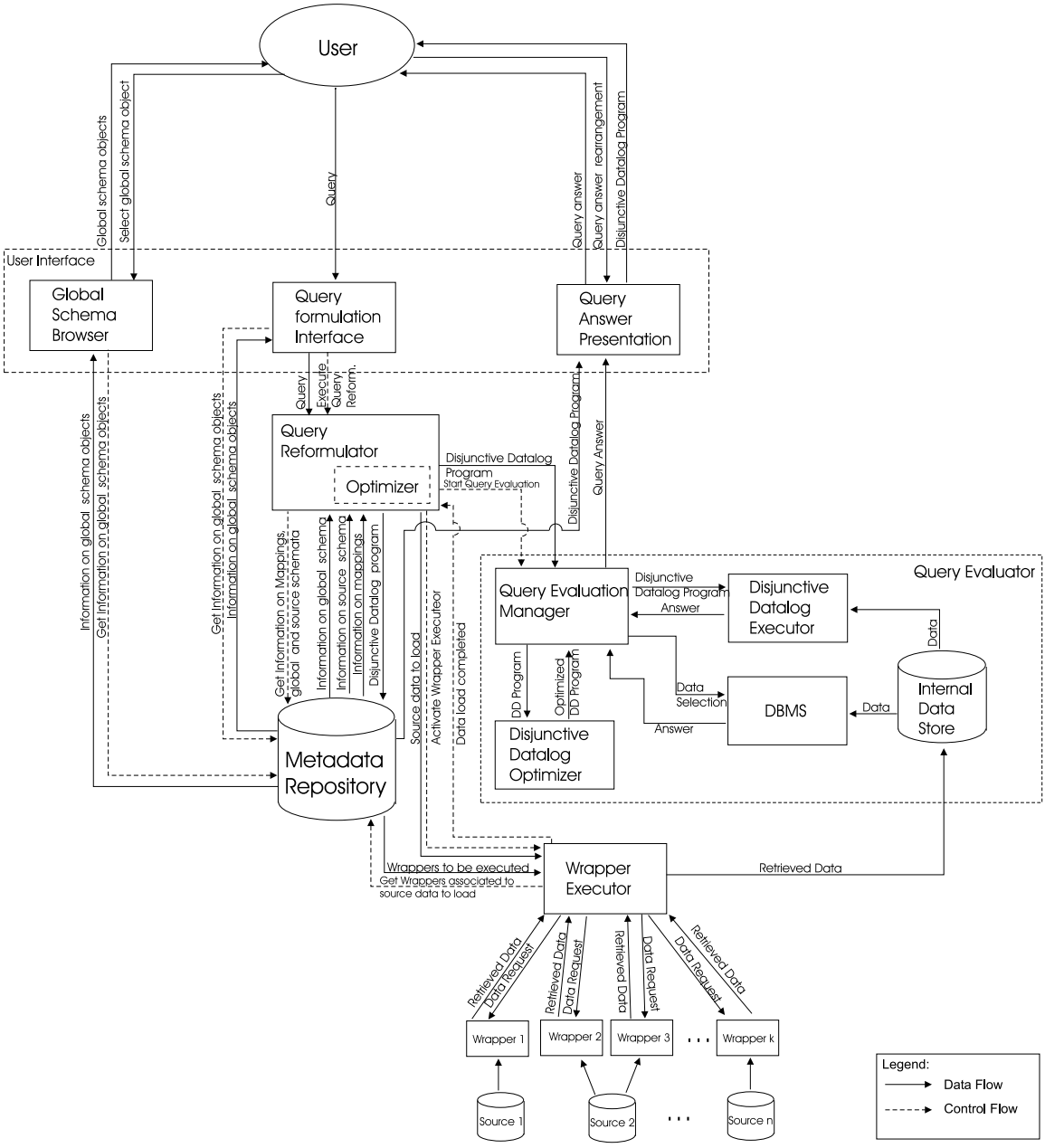

Figure 3: Architecture of INFOMIX system - Run-Time part.

2.2 DAT tools

In document D1.3, we have provided a review of techniques and systems for acquisition and transformation of heterogenous data from the perspective of the envisaged DAT layer of the INFOMIX system. In it, the data from the (physical) sources should be mapped into a logical (virtual) format which amounts to a restricted fragment of XML. From the wide range of DAT tools which are available, the report D1.3 concluded that the following two tools should find application in the development of the INFOMIX system prototype:

1. The OfficeObjects Portal, in particular the *OfficeObjects Repository* together with *OfficeObjects Data Extractor*, of RODAN, for wrapping structured and also semi-structured data; and

2. LiXto Suite, comprising *LiXto Visual Wrapper* and the *LiXto Transformation Server*, of LiXto GmbH for wrapping unstructured, poorly structured, and semistructured data.

2.3 DAT-relevant data formats in INFOMIX

The DAT layer of INFOMIX is concerned with different data formats, which correspond to the data flow in the INFOMIX system (see Figure 4).

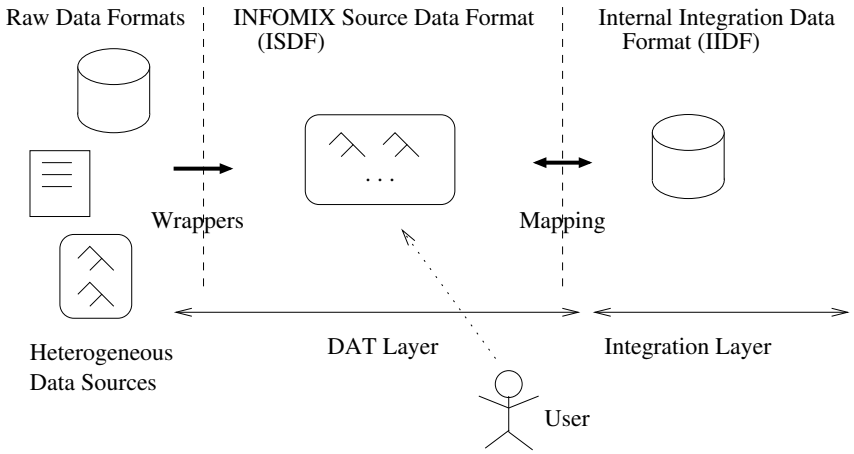

Figure 4: DAT-Relevant Data Formats.

Raw Data Formats: At the input side to the system are the source data in their individual formats, called *raw data formats*. They comprise relational data, object-oriented data, XML data, HTML data and to some extent embedded data (e.g., postscript), some of them under certain restrictions; we refer to D6.1 for details.

INFOMIX Source Data Format (ISDF): The heterogeneous raw data are transformed to the homogenous INFOMIX Source Data Format (ISDF), by means of wrappers as specified in the INFOMIX architecture. The ISDF provides a (logical) view of the

Input	From	Functionality
Source and Data Type	Designer	Specification of the sources to be wrapped
Source Schema	Designer	Specification of the sources to be wrapped
Information about Wrappers and associated schemas	Metadata Repository	Visualization of information on generated wrappers and associated schemas

Table 1: Inputs of the *Wrapper Generation Interface*.

wrapped data which is accessible to the user, and is the uniform logical format of source data for defining mappings between the global data format at the user side, and the local data format at the (wrapped) sources. The ISDF has been specified in document D6.1. It has been designed as a (rather plain) fragment of XML Schema which comprises in essence complex value data that are relational in the core.

Internal Integration Data Format (IIDF) The source data needs to be transformed into an internal format which is used at the internal integration level. This format, the Internal Integration Data Format (IIDF) is by the nature of the INFOMIX integration algorithms the relational data format. The IIDF has been presupposed in document D4.2, and will be further elaborated on in Subsection 4.2 below.

3 Wrapper Generation

The Wrapper Generation Interface allows to incorporate data sources into the system, by specifying the necessary pieces of information, from which then individual wrappers for the data sources are generated.

The *Wrapper Generation Interface* module is in charge of both the generation of the wrappers for the sources participating in the data integration system and the storage in the *Metadata Repository*, of source schemas and associations between source relations and wrappers. All of these tasks are accomplished at design time, setting up the system for run time, during which user interactions on the DAT layer would be impractical and are therefore not foreseen in Figure 2.

In particular, the designer specifies, by means of the *Wrapper Generation Interface*, the set of sources and the format of the data stored in the sources; in case, the designer might also specify some logical source schemas in the ISDF, e.g. if sources do not allow to construct an ISDF automatically. The inputs and outputs of the Wrapper Generation Interface are recalled in Tables 1 and 2.

Since all user interaction with the *Wrapper Generation Interface* is done by the designer (who will in general be some kind of administrator rather than a user of the INFOMIX system), when referring to "user" in this context we actually mean the user of the *Wrapper Generation Interface*, which is the designer.

Output	To	Functionality
Set of Sources and associated data formats	Wrapper Generator	Activation of the Wrapper Generator
Source Schemas	Wrapper Generator	Activation of the Wrapper Generator
Information about wrappers and associated schemas	Designer	Visualization of information on generated wrappers and associated schemas

Table 2: Outputs of the *Wrapper Generation Interface*.

Wrapper Type	Query Formulation Support	Wrapper Generation Support
Code Wrapper	none	none
Query Wrapper	none	automatic
Visual Wrapper	semi-automated	automatic

Table 3: Ways of Wrapper specification in INFOMIX

The INFOMIX prototype will provide different ways of generating wrappers. They comprise, depending on the type of data source which should be incorporated to the data integration system and on the information needed, the possibility of graphical specification, manual specification, or low-level specification in terms of an API executing proprietary code. As far as possible, the user will be relieved from technical details and information required for wrapper execution at run time is compiled automatically, whenever this is feasible.

In more detail, the following possibilities for wrapper generation will exist, which are supplied with different high-level user support (cf. Table 3):

Code wrappers: At the lowest level are code wrappers. They provide a means to integrate, through a well-defined API, any data source into the data integration system. The wrapper is conceived as a black box, which is called via the API for wrapper execution at runtime. The code implementing the API and retrieving the data from the source, has to be provided at design time by the user. No user support can be provided for the generation of code wrappers.

Query wrappers: At the mid- to high-level support are Query Wrappers. Here, a data source is incorporated to the data integration systems via a query, formulated in some query language, which is shipped to the data source for retrieving the data. Prototypical query wrappers will use the ODBC or JDBC interfaces to relational databases. The data retrieved, if not already in ISDF, has to be converted to ISDF, at least conceptually (see the discussion in Section 4). At this level, automatic support for wrapper generation is given: The user just provides a query, the ISDF schema of the source and further configuration information; the wrapper is then generated automatically from this information. No support for the formulation of

the query will be provided. While such a support does not seem impossible, the additional complexity caused by necessary interactions with the underlying database systems is prohibitive.

Visual wrappers: At the high-end of user support, visual tool support is given. That is, graphical tools are available for defining wrappers, which basically help the user to define, by the use of examples, a query expression in an underlying query language for retrieving the desired data from the source in a convenient way. In particular, at this level Rodan's Data Extractor and LiXto's Wrapper Generator as well as Transformation Server will be used. Here, the underlying systems provide (semi-)automatic support for formulating the queries and for generating the wrappers. Thus, user support is given to a higher degree than at the level of Query Wrappers (since the user is eased from writing a complicated query expression in a query language).

The above possibilities for wrapper generation will be detailed in Sections 3.1, 3.2, and 3.3, respectively.

Wrapper Meta Data. As outlined in the INFOMIX architecture (Fig. 2), the Wrappers are stored in the INFOMIX Metadata Repository. By Meta data we refer to information which will be used by the Wrapper Executor at runtime in order to invoke and execute wrappers. The various wrapper types have different meta data associated with them. However, there is a meta data core which is common to all wrapper types. This core comprises the following items:

- The name of the data source,
- the location of the data source,
- the ISDF schema of the wrapped source data, and
- the IIDF schema of the data wrapped to the internal integration format.

We remark that in some cases, automatic ISDF schema and IIDF schema creation is needed and desired (in particular, for relational sources). However, not in all cases schema creation will be fully automatic, and user interaction will be necessary. In case of query and visual wrappers, suggestions of ISDF and IIDF schemas will be made to the user, respectively.

Besides the core attributes, the descriptions of the various wrappers have, according to their type, further attributes as detailed in the subsections below.

Parameters in data access We remark at this point that with respect to optimization and performance issues, it would be desirable that wrappers can handle to some extent *parameters* in the data access. For instance, it is desirable that data selection is carried out already at the data source, i.e., at the level of raw data, if this is feasible, rather than at the level of the ISDF if the definition of a mapping between the data source and a

global relation involves such data selection. In this way, a possibly much smaller portion of data needs to be transferred.

For example, in case of a query wrapper realized with ODBC, the query to the source which retrieves the data might be adjusted in order to carry out a selection for the name "John Doe" already at the source site. Such a selection might come in e.g. in the process of query optimization, if constants are pushed towards the sources. The issues of query optimization will be addressed in Work Package WP5.

This requires, however, that such data selection capabilities are supported by the wrapper. Not all of the wrapper types outlined above do provide such capabilities (e.g., code wrappers cannot support them). Query Wrappers are, in general, flexible and allow for a rewriting of the query shipped to the data source such that selection is pushed through. For visual wrappers, the situation is different and no such easy rewriting is possible. For this reason, the use of *parameters* in the description of data access for such wrappers is suggestive, which might dynamically be instantiated or set to a default value for data access. This will be detailed below.

3.1 Code wrapper definition

A Code wrapper uses fixed code that provides data in source data format. Source configuration parameters (like e.g. a database name) can be either hardcoded or stored in some configuration file, thus viewing a code wrapper at run time as a black box which produces only output. At design time, a code wrapper is therefore to be viewed as an entity with an optional proprietary configuration parameter. Note that as a consequence (query-)optimization techniques are not applicable through code wrappers.

Way of specification: The extraction method of a code wrapper should be written in or made available through Java to be invoked by the INFOMIX Demonstration Platform (IDP) using the Java reflection mechanism.

At design time, the user will specify the code wrapper details like method name, class or package (there will be a possibility to select one of the previously defined API definitions). The user can also define an optional string parameter to instantiate a parametric code wrapper. The interpretation of these parameters depends on the wrapper extraction method (for example in object oriented method an object name can be passed as a parameter).

So strictly speaking, a code wrapper is actually a pair of the actual code along with a parameter. However, this parameter is processed only at design time, and hence at run time it is not possible to determine that two code wrappers rely on the same code base with just differing parameters. Therefore no reasoning or optimization effort can be done in this direction. This run-time-view also justifies the view of a code wrapper to be one inseparable entity. The parameter should just facilitate code re-use.

The following API serves as a specification both at design and at run time. Wrapper and API are de facto equal for code wrappers.

API:

`method(buffer,parameter)` Retrieves all the data in the wrapped source, and materializes it in ISDF into the given `buffer` area. The optional `parameter` is specified at design time and is stored with the meta data. `method` is determined by the meta data *package*, *class*, and *method*.

Meta data needed:

method The name of the method which performs the data extraction.

class The extraction method class name.

package The extraction method package name.

parameter An optional method parameter.

method:	`GetObjectsAsXML`
class:	`OOObjectList`
package:	`pl.rodan.ooportal`
parameter:	`"+43-1-[0-9-/]*"`

Table 4: Example Meta Data for a Code Wrapper

Table 4 gives an example for the meta data associated with a code wrapper. When the wrapper is invoked, `OOObjectList.GetObjectsAsXML(buffer,"+43-1-[0-9-/]*")` of package `pl.rodan.ooportal` will be executed. Here the parameter is used to specify a regular expression which matches phone numbers in Austria. The way how this parameter is used is determined by the method code.

3.2 Query wrapper definition

Query wrappers are specified in terms of a view expressed as a query in some query language. The sources they access typically are database systems, in particular such wrappers provide an interface to relational databases via ODBC or JDBC. However, in principle such a source can be any system interfaced via a query language.

Way of specification: The user has to define the query language to use, and also connection parameters, including database location and authentication information. After defining the source connection parameters, the user will be asked to provide the query to be executed. Query validation will not be performed, but in the case of the SQL query the application will check if that query starts with clause “select”, to make sure that update or insert query will not be executed accidentally.

Meta data needed:

query language Query language (a dictionary type). For the prototype, there will be one kind of query language – SQL. Later-on the number of query languages can be extended (with XPath, for example).

database type The type of the database. There will be several predefined data types for SQL (ORACLE, MSSQL, POSTGRES and ODBC), but also a new database type can be added. The ODBC is not a real database type, but it defines when the jdbc:odbc bridge should be used instead of the jdbc connection.

driver The driver to be used to access the database. Several jdbc database drivers will be provided with IDP. If the user chooses a driver not provided with IDP, she must put the proper database driver into the IDP class-path.

user Name of the database user.

password Password of the database user.

connection string A database connection string (specifying name, location, etc).

query The query to be filed.

query language:	`SQL`
database type:	`POSTGRES`
driver:	`org.postgresql.Driver`
user:	`infomix`
password:	`infomix`
connection string:	`jdbc:postgresql://192.168.1.102:5432/infomix`
query:	`select name, surname from users where name like 'T%'`

Table 5: Example Meta Data for a Query Wrapper

Table 5 contains example meta data of a query wrapper. When the wrapper is invoked, it will retrieve tuples by executing the query (possibly extended by dynamically passed additional conditions) in the specified database, using the driver and login information.

The run-time API for query wrappers contains (beside the buffer to store retrieved data) an optional condition, which can be passed down to the wrapper by a higher-level module.

API:

`query_method(buffer,condition)` Retrieves all the data from the database, specified by the associated meta data, under a possible additional condition specified as an argument. The additional parameter can serve as a handle for future query optimization.

3.3 Visual wrapper definition

Visual wrappers are used to specify the source schema and to define source view query definitions by graphical means. The tools supported by the prototype will be Office Objects QBE, LiXto Visual Wrapper, and LiXto Transformation Server. These tools will be embedded into the INFOMIX prototype, and will require some local setup information (such as the URL to LiXto Transformation Server), which will be stored in IDP configuration files.

Way of specification: The user has to select one of the three available DAT visual tools (Office Object QBE, LiXto Visual Wrapper, LiXto Transformation Server). We will first describe requirements for each of these and will then formulate a generalization in order to unify the meta data.

Office Object QBE The user has to select one of the available "extractions" in the system (defined previously using Office Object QBE). If there are some required parameters defined in the extraction, the user will be asked to enter them. The definition of an extraction in Office Object QBE is documented in the Office Object manuals.

LiXto Visual Wrapper The user has to provide the name of the "wrapper program" defined previously in the LiXto Visual Wrapper tool. Then, the user has to provide the URL of the page the wrapper program should operate on (if not, then default page specified in the LiXto wrapper program will be used). The process of creating a wrapper program is well-documented in the LiXto manuals.

LiXto Transformation Server The identifier of a pipe, which was previously defined in the LiXto Transformation Server, must be specified by the user. The access to such a pipe possibly requires authorization, in such a case the user has to specify a username and the associated password. The way how a pipe is created in LiXto Transformation Server is described in the LiXto manuals.

Meta data needed: In all three cases, an external system is used to create a resource (extraction, wrapper program, or pipe, respectively), which is later referred to by name. Some of these resources need to be instantiated by specifying some parameter or a URL. For LiXto Transformation Server some authorization to access a resource is possibly needed.

So we propose the following generalization for meta data storage: For each wrapper its type and resource name must be stored, along with its instantiation parameter. Since authorization might also be needed in the future for other wrapper types, we have decided not to view authorization as a special kind of parameter, but to represent it separately in the meta data storage. That means that currently the instantiation parameter will always be empty for a LiXto Transformation Server wrapper, while user and password information will always be empty for other wrappers.

type Indicates whether the wrapper is an Office Object QBE, LiXto Visual Wrapper, LiXto Transformation Server type wrapper.

resource name For Office Object QBE this is the extraction definition name, for LiXto Visual Wrapper it is the program file path, for LiXto Transformation Server the pipe identifier.

parameter This can be a string with extraction parameters for Office Object QBE, an URL to the page to be operated on for LiXto Visual Wrapper. Currently not used for LiXto Transformation Server.

user Specifies user information for authorization. Currently only used for LiXto Transformation Server pipes.

password Specifies password information for authorization. Currently only used for LiXto Transformation Server pipes.

type:	`Office Object QBE`
resource:	`FAMILY`
parameter:	`&name='Smith'`
user:	
password:	

type:	`LiXto Transformation Server`
resource:	`world.lila.p1`
parameter:	
user:	`infomix`
password:	`infomix`

Table 6: Example Meta Data for Visual Wrappers

Table 6 shows example meta data for an Office Object QBE wrapper (left), which passes an extraction parameter (that `name` should be `'Smith'`) to the extraction named `FAMILY`. The right part of Table 6 are the meta data for a LiXto Transformation Server pipe named `world.lila.p1`, to be accessed by user `infomix` with password `infomix`.

Finally, the API definition of visual wrappers is very simple, as no data is passed at run-time.

API:

`visual_wrapper(buffer)` This method simply invokes the associated wrapper and stores the results in the `buffer`.

4 Wrapper Execution

The execution of wrappers at run time will be taken care of by the *Wrapper executor*. The inputs and outputs of running a particular Wrapper are recalled in Tables 7 and 8.

Input	From	Functionality
Source Data to Load	Query Reformulator	Wrapper Execution
Wrappers to be executed	Metadata Repository	Wrapper Execution
Retrieved Data	Wrappers	Wrapper Execution

Table 7: Inputs of the *Wrapper Executor*.

Output	To	Functionality
Retrieved Data	Internal Data Store	Wrapper Execution
Data Request	Wrappers	Wrapper Execution

Table 8: Outputs of the *Wrapper Executor*.

4.1 Mapping from Raw Data to ISDF

Raw data is mapped into ISDF upon wrapper execution. However, there may be exceptional cases, where raw data is mapped directly into IIDF (see Section 6.1).

In general, it is the task of an administrator to specify a mapping to ISDF for the data at hand. Depending on the type of wrapper that is built for a source, the "mode" of specifying the mapping may vary, e.g by writing code in case of a code wrapper definition or by graphically defining an extraction in case of a visual wrapper definition.

Nevertheless, as for schema generation, there may be automatic support for the mapping generation for particular raw data formats. Predominantly for relational data, the mapping – like the schema – will be generated automatically following a procedure for mapping relational data to XML. Automating such mappings is discussed e.g. in [19]. Several implemented tools are available for this task, among them:

- DB2XML [30, 29]
- Oracle XML SQL Utility
- ODBC2XML [25]
- IBM DB2 XML Extender [15]
- XML-DBMS [3]
- SilkRoute [10]
- Xperanto [7]

A listing of tools for converting relational data into XML with detailed descriptions is maintained at `http://www.rpbourret.com/xml/ProdsMiddleware.htm`. During the implementation phase, a commitment to one of these tools should be made.

4.2 IIDF

The Internal Integration Data Format (IIDF) is the data format on which the internal INFOMIX integration algorithms work. This format has been presupposed in Deliverable D4.2 already, without explicit specification, as the relational data format. The reason for this is threefold:

- Firstly, the theoretical results about integration have been developed with the relational data format as core in mind.
- Secondly, the DLV deductive database system, is geared towards relational data.
- Thirdly, for scalability issues the usage of efficient database engines for subtasks or optimization in the data integration process has been recognized as a promising approach. Here, relational database technology is currently by far the most developed and sophisticated.

More precisely, the IIDF consists of the standard, flat relational data format. The data is stored in a relational storage, the *Internal Data Store (IDS)*, which comprises database tables of the form

```
table_name(C1: type_1, C2: type_2,...., Cn: type_n)
```

where `table_name` is the the name of the relation and `C1`,..., `Cn` are the columns, which are typed with respective types `type_1`,..., `type_n`. As for the types, there are no particular restrictions imposed except for those which emerge from the particular software which implements the IDS. Furthermore, the data mapped to IIDF (which are both data from the global schema and the local sources) might incur implicit restrictions to the datatypes foreseen (cf. the ISDF definition in Deliverable D6.1).

While the IDS is not bound to a particular location, for efficiency reasons it is intended to be kept in main memory whenever feasible.

The INFOMIX internal integration algorithms, and in particular the DLV system employed shall access the IDS holding the data for integration through an ODBC interface. The ODBC interface developed for DLV to this end is described in Deliverable D4.2; we do not repeat it here.

Since DLV can handle only a restricted set of data types, basically,

- integers of bounded range, where the range is dynamically specified at runtime, and
- strings,

a type conversion of data in the IDS to the DLV types is necessary. This conversion is performed during the access of the IDS through ODBC calls, where the conversion of any data type to type integer and/or type string is feasible; we refer to Deliverable D4.2 for further details.

4.3 Mapping of ISDF to IIDF

The mapping of the semistructured ISDF to the relational IIDF requires a specific transformation algorithm. Several techniques to store XML data in relational databases have been proposed in literature; In [27] three inlining algorithms are presented that focus on the table level of the schema conversions, whereas [11] concentrates on the attribute and value level of the schema, comparing different algorithms and their performance. A method to preserve semantic constraints of a DTD is discussed in [18]. STORED [8] is a data mining based system that directly uses the XML document instead of a DTD to create the relational schema. A utility to exchange data between XML and a relational DBMS is presented in [2], where the user is able to specify a mapping by a template language.

The Mapping of ISDF to IIDF comprises two major tasks:

- Resolving the conflict between the two-level nature of relational schemas vs. the arbitrary nesting of XML schemas. To this end, the ISDF has to be "flattened" into the two-level IIDF structure.
- Dealing with set-valued attributes in the ISDF by decomposing the respective element into a separate relation.

The flattening algorithm starts with the creation of a table for the root element of the XML schema, denoting it with the element's name, and continues by processing each child element as follows:

1. A simple type child element is translated into a relational attribute.
2. A complex type child element, its attributes, and (possibly nested) elements are translated into respective relational attributes, using the "dot" notation (explained below).
3. A multi-valued complex type child element causes the addition of a primary key attribute (unless such a primary key has already been created) and the creation of a separate relation, where a foreign key is added, referencing the aforementioned primary key. For each child element of the multi-valued element, the algorithm is started recursively, using the newly created relation.

Preserving the overall structure when decomposing data in multiple relations demands private and foreign key datatypes. Basically they can be emulated by one of the existing simple types together with the specific constraints that are inherent to key attributes (i.e., values of the primary key must be unique, each value of a foreign key has to exist as a primary key value in the according relation). We will call these datatypes `primary_key` and `foreign_key`.

A foreign key has to include the relation and attribute name which it references. We will express this information via the foreign key attribute name, encoding the referenced relation name as well as the primary key attribute name (see Example 4.1).

The "dot" notation mentioned in step 2 ensures, that the entire path from the root element is preserved in order to guarantee a bidirectional mapping. This notation is

characterized by concatenated element names, according to the entire path from the root element, separated by dots (see the following example). To illustrate the application of the algorithm, we reuse Example 5.3 of D6.1:

Example 4.1 A soccer team with all its players included is given by the following schema:

```
<xsd:complexType name="SoccerTeam">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:element name="Coach">
      <xsd:complexType>
        <xsd:attribute name="FirstName" type="xsd:string"
                       use="required"/>
        <xsd:attribute name="LastName" type="xsd:string"
                       use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Player" maxoccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="FirstName" type="xsd:string"
                       use="required"/>
        <xsd:attribute name="LastName" type="xsd:string"
                       use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Applying the mapping algorithm results in the following relational schema:

```
SoccerTeam(Name: string,
           Coach.FirstName: string,
           Coach.LastName: string,
           PK: primary_key)
Player(FK.SoccerTeam.PK: foreign_key,
       FirstName: string,
       LastName: string)
```

Note how the complex type element `Coach` has been split into two attributes using the "dot" notation. This is feasible as there will be exactly one `Coach` element in each `SoccerTeam` element. For the complex type element `Player` element the situation is different because an arbitrary number of `Player` elements can occur in a `SoccerTeam` element. Therefore, `Player` is transformed into a separate relation, which is linked to the `SoccerTeam` relation by a foreign key. To this end, a primary key has to be introduced in the `SoccerTeam` relation. Note that if other elements were present in `SoccerTeam`, which can occur multiple times, e.g. `CupsWon`, the corresponding relations would re-use the primary key `PK` of `SoccerTeam`.

Consider some example data in ISDF extended from Example 5.3 in Deliverable 6.1 depicted in Table 9.

```
<SoccerTeam>
  <Name>Bayern Munich</Name>
  <Coach  FirstName="Ottmar" LastName="Hitzfeld"/>
  <Player FirstName="Oliver" LastName="Kahn"/>
  <Player FirstName="Giovane" LastName="Elber"/>
</SoccerTeam>
<SoccerTeam>
  <Name>Reggina</Name>
  <Coach  FirstName="Franco" LastName="Colomba"/>
  <Player FirstName="Emanuele" LastName="Belardi"/>
  <Player FirstName="Shunsuke" LastName="Nakamura"/>
  <Player FirstName="David" LastName="Di Michele"/>
</SoccerTeam>
```

Table 9: Example ISDF data.

The corresponding data mapped into IIDF is given in Tables 10 and 11.

Name	Coach.FirstName	Coach.LastName	PK
Bayern Munich	Ottmar	Hitzfeld	001
Reggina	Franco	Colomba	002

Table 10: Relation SoccerTeam.

FK.SoccerTeam.PK	FirstName	LastName
001	Oliver	Kahn
001	Giovane	Elber
002	Emanuele	Belardi
002	Shunsuke	Nakamura
002	David	Di Michele

Table 11: Relation Player.

Note that the implicit key constraints are fulfilled: Each tuple in SoccerTeam has a unique primary key value PK, and each foreign key value of FK.SoccerTeam.PK in Player exists as a primary key in SoccerTeam.

4.4 Conversion of IGDF to IIDF

While strictly speaking not pertinent to this document, we mention here that also the data in the global schema, which conform to the INFOMIX Global Data Format (IGDF,

which is in essence relational data format), must for integration purposes be in IIDF. However, since the IGDF is a plain relational format, no special conversion is needed.

4.5 Data Cleaning

In a general setting, the data retrieved from the sources have to be reconciled, converted and combined in order to make them fit into the structures of the global schema. This is relevant if the sources, as is expected to be the case, are independent and they are not controlled by the integration system. *Data Cleaning and Reconciliation* refers to a number of issues arising when considering integration at the extensional/instance level.

A first issue in this context is the (low-level) interpretation and merging of the data provided by the sources. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by various sources need to be converted and merged to provide the data integration system with the requested information. Accordingly, data cleaning problems can be grouped in two main categories: *differences in representation of the same data* and *invalid data*.

Different Data Representation When gathering information from different independent data sources, it is likely to happen that the same information is represented in different ways in different sources. [24] identifies three main categories of such conflicts. *Naming conflicts* arise when the same name is used for different objects, or when different names are used for the same object. *Structural conflicts* are more general, and occur when the difference in the representation lies in the structure of the representation itself. Examples of structural conflicts are the *use of different data formats* in different sources *for the same field*. A common example for this is the different representation of dates, e.g., by strings "Nov 13, 2003" vs. "November 13th, 2003". Another example is the fact that *data that are represented by more than one field in a source may be represented in a single field in another one*; [1] presents an approach to give a field structure to free-text addresses.

Other structural conflicts, in the relational model, are due to different relational modeling in different sources, e.g., attribute vs. relation representation, different data types, etc. Finally, *data conflicts* appear only at the instance level, and are mainly related to different value representations. For example, the currency may be expressed in Japanese Yen in a source and in Euros in another. Another typical case is the different representation of the same value; for instance, "Simon M. Sze" vs. "Sze, S. M.".

Invalid Data Invalid data can be caused by extracting data from multiple sources, or they can exist in a single source, due to incorrect data entries. For example, a "City" field may contain the value "Italy", or a simple misspelling may occur, e.g., the field "Country" contains the value "Brazik" instead of "Brazil". A slightly more complex problem is due to inconsistencies among different fields of the same record; for example, a record regarding a person may have the value "12 December 1973" for the date of birth and the value "12" for the age. Violation of functional dependencies within a table is another typical example of such inconsistencies. Dealing with invalid data at this level, however, requires to take a "semantic" point of view and, in

a more sophisticated version, to perform reasoning tasks which involve richer semantic knowledge about the background of the application. However, such a "semantic approach" based on heuristics and characterized by a more pragmatic spirit [5], may exceed plain data cleaning capabilities and interfere with an inconsistency handling layer on top of it.

Another problem in data cleaning is that of overlapping data [13, 21, 22, 26, 23], also referred as *duplicate elimination problem* or *merge/purge problem*. This problem arises when different records of data representing the same real-world entity are gathered in a data integration system. In this case, duplicates have to be detected and merged. Most efforts in data cleaning have been devoted to the solution of the duplicate detection problem, which is a central issue.

In [13, 22] methods are proposed which are based on the use of a *sliding window* moving along an ordered list of records to be analyzed: a key is associated to each record and records in the windows having the same key are merged into a single record. Obviously, these methods heavily depend on the sorting, and therefore on the key, both based on heuristics.

An alternative to window scan is to partition the records into *clusters*, where in each cluster records are stored, that match each other according to a certain criterion (see, e.g., [22]). Duplicate elimination can be performed on each cluster separately, and in parallel.

Finally, *knowledge-based approaches* have been used in data cleaning [12, 20]. This approaches favor the use of declarative specifications for the matching operations. However, human validation and verification of the results is in general needed at the end of the data cleaning process.

4.5.1 Data Cleaning in INFOMIX

As for the INFOMIX DAT layer, it is desirable to allow for some data cleaning capabilities. For the purpose of the overall system, data cleaning is needed because otherwise a number of (inessential) inconsistencies might surface, which make the integration problem computationally complex and lead to semantical worse integration results.

Data cleaning can be viewed as task which should be accomplished by the wrapper. However, also some extra piece of "data cleaning" software might be used, which receives the data from a wrapper as input, together with some data cleaning directives, and outputs the cleaned data. We thus have an *extended wrapper* accomplishing this task. The capabilities of LiXto's Transformation Server, which also allows for data cleaning, can be utilized in this way to a certain extent.

However, the development and implementation of data cleaning methods is not on the agenda of the INFOMIX project. For the purpose of the prototype development, some elementary data cleaning capabilities should be realized at the wrapper level, which are tailored for the specific applications considered in the course of the project. These capabilities need not comprise comprehensive and advanced services, and may focus on the aspects of common data representation within single attributes, and the correction of values because of spelling errors.

In particular, for the intended model application in the domain of organization administration, data cleaning support for common representation of date formats, correction of misspelled names and constituents of addresses etc. should be given.

5 Auxiliary Functions

At the design time (cf. Figure 2), the Wrapper Generation Interface (WGI) should be provided by the DAT layer, as discussed in Section 3. Besides specifying wrappers and sources, this module should also provide some auxiliary added-value functions, as outlined below.

5.1 Schema Editing and Browsing

The administrator or user of the WGI should be able to access, browse and edit the schemas of the sources in addition to be able to just specify them. The editing functionality has to be provided anyway, as it is necessary for creating the schema for sources where the schema cannot be generated automatically.

Any XML schema editor and/or XML schema browser can be used to provide such a functionality. Listings of available tools can be found at `http://www.w3.org/XML/Schema` or `http://directory.google.com/Top/Computers/Data_Formats/Markup_Languages/XML/Tools/Editors/Schema_and_DTD/`.

5.2 Source Data Browsing

In addition, it will be useful for the user or administrator to browse the data in the sources. In essence, this refers to viewing the result of wrapper executions, and is therefore in between the design- and run-time system. In particular, if used during the design phase, it anticipates some run-time functionality (executing wrappers), and if used at run-time it provides direct access to the sources, which is considered to be a design-time privilege.

Such a feature has not been explicitly been included in the INFOMIX architecture, but even though it blurs the design- and run-time separation, providing it would imply a true benefit for designers, allowing for validation or debugging.

Any XML browsers can be employed to provide such functionalities, listings of available tools for XML browsing can be found at `http://directory.google.com/Top/Computers/Data_Formats/Markup_Languages/XML/Tools/Browsers/` or `http://www.wdvl.com/Software/XML/browsers.html`.

6 Implementation Guidelines

As for types of wrappers and supported raw data formats, not every possible combination needs to be implemented. In particular, we *foresee* the possibility of wrapping OODBMSs but only Code wrappers as described above might be supported in this case and *no concrete OODBMS will actually be wrapped.*

6.1 Data Transfer

While wrappers are specified to produce ISDF data, the Internal Data Store (IDS) will expect IIDF data. So, in general this implies that a data transformation from ISDF to IIDF is necessary. Indeed, in Section 4.3 we have described methods for achieving this.

A first, naïve approach will therefore always invoke this transformation. However, the IIDF is essentially the relational data format, and also query wrappers are usually defined on relational data. In this particular setting, it is evident that the transformation from relational format to ISDF and then back to relational format produces redundant ISDF data. If the wrappers allows for doing so, the production of ISDF data should be skipped, yielding directly relational IIDF data. However, it is still necessary to produce ISDF data if needed and the ISDF schema definition must always be available, as this is the user's view of the source data. We will therefore refer to this technique as *Bypassing* or *Lazy ISDF Generation*.

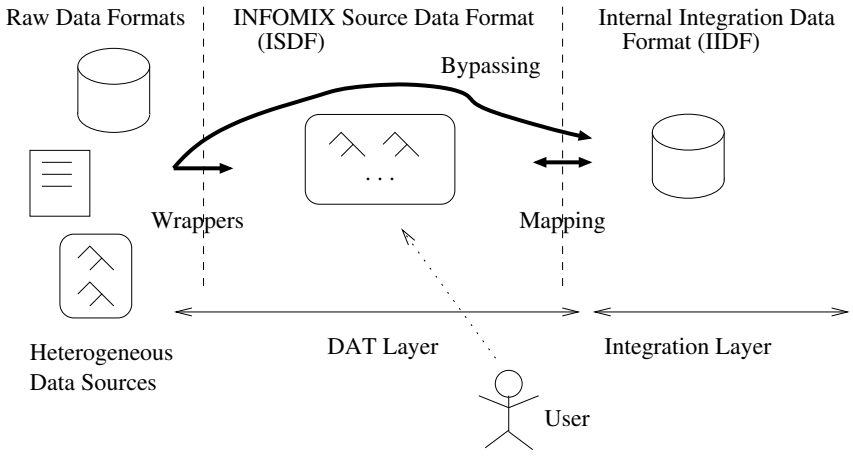

Figure 5: Bypassing and Lazy ISDF Generation.

Note that not only query wrappers are candidates for bypassing techniques. If the wrappers permit this, raw data could be mapped to IIDF directly, rather than producing ISDF as an intermediate step. In essence this would mean to push the data conversion from between the Wrapper Executor and IDS into the wrappers (cf. Figure 3). However, also in this case the corresponding ISDF data must be producable upon request. In the setting of Figure 4, bypassing can be visualized as in Figure 5. Here, it becomes evident that a bidirectional mapping between ISDF and IIDF guarantees that ISDF data can always be produced from IIDF data upon request, rather than having to invoke the wrapper in order to produce ISDF data.

6.2 Materialization and Caching

With respect to the data explicitly managed by a data integration system, it is in general possible to follow two different approaches, called *materialized* and *virtual*. In the materialized approach, the system computes the extensions of the structures in the global

schema by replicating the data at the sources. In the virtual approach, data residing at the sources are accessed during query processing, but they are not replicated in the integration system.

The materialized approach to data integration is the most closely related to *Data Warehousing* [16, 9, 28, 6, 17]. In this context, data integration activities are relevant for the initial loading and for the refreshing of the Warehouse, but not directly for query processing. Obviously, maintenance of replicated data against updates to the sources is a central aspect in this context, and the effectiveness of maintenance affects timeliness and availability of data. A naïve way to deal with this problem is to recompute materialized data entirely from scratch in the presence of changes at the sources. This is expensive and makes frequent refreshing impractical. The study of the materialized data management is an active research topic that is concerned with both the problem of choosing the data to be materialized into the global schema, and reducing the overhead in the re-computation. However, an in-depth analysis of this problem is outside the scope of this report (see [31, 14, 4]).

INFOMIX follows the virtual approach to data integration, in which sources are accessed on the fly, i.e., each time a user query is posed to the system, since it provides the user only with a virtual global schema, whose extension is not stored in a Data Warehouse or similar. Still, it is possible and, in fact, even likely that at run time the same sources will be accessed in equal or similar ways frequently when processing user queries. In the setting described so far, this would entail wrapper invocations each time a source is accessed, possibly repeating computations often.

It would therefore be natural to provide means in order to assimilate the virtual to the materialized approach, but without having to deal with problems arising from it in the integration framework proper. Such means would be caching techniques, in which an intermediate storage (the cache) holds results of previous wrapper executions and takes data from this intermediate storage when the same or similar source access has to be processed again.

Caching techniques work transparently, i.e. the conceptual system behavior does not change by their addition, only performance is affected. In this way, the integration formalism is relieved from having to deal with materialization issues, but its performance can still benefit from transparent materialization. We point out that such methods are therefore to be regarded as *optimizations* and are therefore not of immediate importance for the prototype. However, the system design should incorporate the possibility of plugging in such caching modules.

Concerning the question where exactly such a caching module should be located, we note that while ISDF is more a 'logical' data format used for specifying schemata, the important format for actually providing data in stored form to other system layers via the IDS is the IIDF, especially since the ISDF can be "bypassed" by applying "lazy ISDF generation" as outlined in Section 6.1. Hence, the caching module should handle data in IIDF and should be located in the Wrapper Executor at its interface with the IDS.

A cache can be plugged into the DAT layer as follows: Whenever the Wrapper Executor (cf. Figure 3) receives a query, it checks whether a materialized view of exactly the same query is stored in the cache. If so, it uses the materialized data and does not invoke any wrapper. The Wrapper Executor can also check whether a more general query is stored

in the cache and retrieve the data by executing a trimming query over the stored view. If nothing matches in the cache, the standard wrapper invocation is initiated. The retrieved data can be stored in the cache – if the cache capacity is exceeded, one of several standard replacement strategies (like *first-in-first-out* or *least-frequently-used* etc.) can be adopted.

We give the following implementation guidelines with respect to caching:

- Caching must be transparent.
- Use simple, proved, and readily available techniques.
- Some of the tools which will be incorporated as wrappers offer their own caching possibilities. These should be exploited whenever possible.

6.3 OOPortal for implementing the DAT layer

OOPortal is a platform for information management application development. Thus, at the system level, INFOMIX will be realized as an application built on top of the OOPortal platform. As such, an oo-analysis will be carried out, as usual for OOPortal applications.

The result of the analysis is a schema for the full application domain in terms of OOPortal classes. Hence, although OOPortal objects consist of content plus meta-data, it need not be the case that every 'logical source' is reflected in a single OOPortal object with its schema, kind of raw data format, etc. as meta-data and its data as content. It may well be that, in the implementation, a 'logical source' is internally represented by several OOPortal objects.

In Section 6.4 a preliminary UML model of the DAT layer is given, which serves as a first recommendation for implementation within OOPortal. This model is subject to further reviews, discussions and improvements within the INFOMIX consortium, and RODAN in particular. With respect to this oo-analysis we note that:

- Currently, inheritance is not supported in OOPortal. The way inheritance is implemented at the moment uses a particular type field and depending on its value a pointer is interpreted as a pointer to a particular data structure.
- Care needs to be taken: The same external source may be accessed by different wrappers.
- The INFOMIX meta-data repository will be a 'logical' entity in the sense, that the meta-data may be 'physically' distributed over several objects stored in the OOPortal.

6.4 Object Model of the DAT Layer

Figure 6 shows a UML class association diagram of the DAT Layer from the logical perspective, based and elaborated on the specifications in this report. It is important to bear in mind that each of the boxes represents a class of instances rather than an individual object. This diagram can serve as a starting point for the design of the IDP, and can be refined in consecutive steps of the application development.

The diagram consists of three separate spheres:

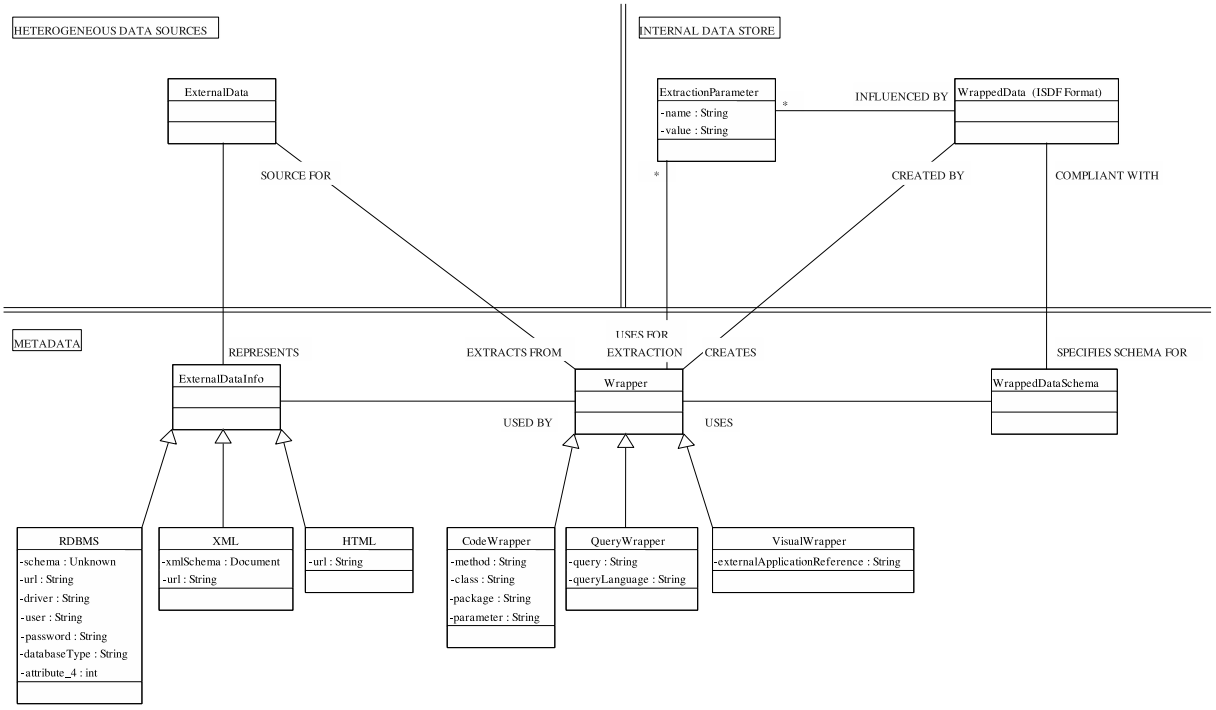

Figure 6: UML Object Model of the DAT Layer

- Heterogeneous Data Sources (outside DAT, presented only for overall picture)
- Metadata
- Internal Data Store

The **Heterogeneous Data Sources** sphere contains an **ExternalData** entity, which represents any external and heterogeneous data sources accessible from the IDP system.

The **Metadata** sphere contains the following entities:

ExternalDataInfo represents specific information for a given external data type. The information is necessary to identify, locate and connect to these data sources. For now, there are three external data source types supported. These types are represented by:

RDBMS entity contains attributes specific for RDBMS support like database type, URL, driver, schema (optional - if needed, could be retrieved from the original external data source) and authorization data (user, password)

XML entity contains an XML schema (optional attribute) and a URL of the XML code

HTML entity contains a URL of the HTML code

Wrapper represents information necessary for data extraction. For now, there are three wrapper types supported. These types are represented by:

CodeWrapper entity contains information on a specific code to be executed upon extraction

QueryWrapper entity contains information on the query language and a possibly parameterized query

VisualWrapper entity contains the reference to any external extraction application (e.g. OfficeObjects DataExtractor, LiXto; it is important to notice that each of these applications is responsible for definition, storage and exploitation of its own specific queries, programs etc. - these are hence not modeled in the diagram)

WrappedDataSchema represents internal unified data format (ISDF), specifies schema of wrapped data, defined during wrapper design time

The **Internal Data Store** sphere contains the following entities:

WrappedData represents ISDF formatted data retrieved by any of available wrappers

ExtractionParameter represents the actual parameter set used during a particular extraction, e.g. name of a person whose monthly salaries are to be extracted. A number of pairs $\langle$parameter name, parameter value$\rangle$ can be passed to the wrapper during its execution. Such defined parameters allow either for substitution of hard coded values or for restriction of extraction set (e.g. $\langle$first_name, "Jan"$\rangle$).

6.5 Functionality of first implementation prototype

The INFOMIX prototype will be realized in two stages. The first prototype should be available, after a first rapid implementation phase, by the end of month 23 of the project. It will not offer the full functionality of the system, though.

As for the first prototype, the INFOMIX DAT layer shall exhibit the following features and limitations:

- The main focus is the servicing of data sources in the relational format, with wrapper generating support at the level of code wrappers and query wrappers.
- Automatic wrapper generation will be supported for relational data sources, including the generation of ISDF schema information. However, while the ISDF schema is visible, support of materializing the XML document (for viewing purposes) need not be realized. According to lazy ISDF generation, relational source data might be straight queried using SQL for simple mappings to global relations.
- Concerning access of other types of data sources (whose support is not mandatory for the first prototype), just graphical support by LiXto Visual Wrapper will exist, where a proposal for the ISDF schema is either created automatically by Visual Wrapper, or the ISDF schema is first created manually by the administrator using the INFOMIX interface and then uploaded to Visual Wrapper).

- No type-checking of the source data with respect to the ISDF schema will be performed. That is, it is assumed that all data are of proper types.
- The mapping of ISDF to IIDF can be dispensed for the first prototype, since the latter is intended to support only mappings for a GAV style data integration, where the data integration algorithms do not access source data explicitly.

7 Conclusion

This report is the second of two reports aiming at the definition and description of a framework for engineering the "low level" data access to data of different data sources, which will be used for the design and implementation of the data acquisition and transformation layer of the INFOMIX system. In the second document, we have taken up the basis for such a framework as being developed by

(*i*) specifying the raw data formats for the heterogeneous sources supported by the INFOMIX system and by

(*ii*) specifying a homogenized data format at the source level, the INFOMIX source data format, to which the heterogeneous source data formats are converted.

Building on this, we defined methods for data acquisition and transformation, including methods for wrapper generation, methods for mapping the internal source data format to the internal integration data format, and suitable caching techniques.

Acronyms

1. DAT = Data Acquisition and Transformation.
2. DBMS = Database Management System.
3. DTD = Data Type Definition.
4. HTML = Hypertext Markup Language.
5. IDP = INFOMIX Demonstration Platform.
6. IDS = Integration Data Storage.
7. IIDF = Internal Integration Data Format.
8. IGDF = INFOMIX Global Data Format.
9. ISDF = INFOMIX Source Data Format.
10. JDBC = Java Database Connectivity.
11. ODBC = Open Database Connectivity.

12. ODBMS = Object DBMS.
13. ODMG = Object Data Management Group.
14. OQL = Object Query Language.
15. RDBMS = Relational DBMS.
16. SGML = Standard Generalized Markup Language.
17. SQL = Structured Query Language.
18. W3C = World Wide Web Consortium.
19. WGI = Wrapper Generation Interface
20. WP = Workpackage.
21. WWW = World Wide Web.
22. XML = Extensible Markup Language.

References

[1] Vinayak R. Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatically extracting structure from free text addresses. *IEEE Data Engineering Bulletin*, 23(4):27–32, 2000.

[2] Ron Bourret, Christof Bornhövd, and Alejandro P. Buchmann. A generic load/extract utility for data transfer between xml documents and relational databases. In *Proceedings of the Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, pages 134–143, June 2000.

[3] Ronald Bourret. Xml-dbms. `http://www.rpbourret.com/xmldbms/`.

[4] Mokrane Bouzeghoub, Francoise Fabret, Helena Galhardas, Maja Matulovic-Broqué, Joao Pereira, and Eric Simon. Data warehouse refreshment. In Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors, *Fundamentals of Data Warehouses*, pages 47–86. Springer, 1999.

[5] Mokrane Bouzeghoub and Maurizio Lenzerini. Introduction to the special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 26(8):535–536, 2001.

[6] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Data integration in data warehousing. *International Journal of Cooperative Information Systems*, 10(3):237–271, 2001.

[7] Michael J. Carey, Daniela Florescu, Zachary G. Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene J. Shekita, and Subbu N. Subramanian. Xperanto: Publishing object-relational data as xml. In *WebDB (Informal Proceedings) 2000*, pages 105–110, 2000.

[8] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing semistructured data with STORED. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*, pages 431–442. ACM Press, June 1999.

[9] Barry Devlin. *Data Warehouse: From Architecture to Implementation.* Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[10] Mary F. Fernandez, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, and Wang Chiew Tan. Silkroute: A framework for publishing relational data in xml. *ACM Transactions on Database Systems*, 27(4):438–493, 2002.

[11] Daniela Florescu and Donald Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical report, Inria, Institut National de Recherche en Informatique et en Automatique, 1999.

[12] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. Delarative data cleaning: language, model and algorithms. Technical Report 4149, INRIA, 2001.

[13] Mauricio A. Hernàndez and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[14] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 481–492, 1996.

[15] IBM. Ibm db2 xml extender. `http://www.ibm.com/software/data/db2/extenders/xmlext/`.

[16] W. H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, second edition, 1996.

[17] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses.* Springer, 1999.

[18] Dongwon Lee and Wesley W. Chu. Constraints-preserving transformation from xml document type definition to relational schema. In Alberto H. F. Laender, Stephen W. Liddle, and Veda C. Storey, editors, *Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000)*, volume 1920 of *LNCS*, pages 323–338. Springer, October 2000.

[19] Dongwon Lee, Murali Mani, and Wesley W. Chu. Schema conversion methods between xml and relational models. In Borys Omelayenko and Michel C. A. Klein, editors, *Knowledge Transformation for the Semantic Web*, volume 95 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.

[20] Wai Lup Low, Mong Li Lee, and Tok Wang Ling. A knowledge-based approach for duplicate elimination in data cleaning. *Information Systems, Special Issue on Data Extraction, Cleaning and Reconciliation*, 26(8), December 2001.

[21] Alvaro E. Monge and Charles P. Elkan. The field matching problem: algorithms and applications. In *Int. Conf. on Practical Applications of Prolog (PAP'97)*, 1996.

[22] Alvaro E. Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of the ACM-SIGMOD workshop on research issues on knowledge discovery and data mining*, 1997.

[23] Alvaro E. Monge and Charles P. Elkan. Matching algorithms within a duplicate detection system. *IEEE Data Engineering Bulletin*, 23(4):14–20, 2000.

[24] Erhard Rahm and Hong Hai Do. Data cleaning: problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[25] Intelligent Systems Research. Odbc2xml. `http://www.intsysr.com/odbc2xml.htm`.

[26] Abhik Roychoudhury, I. V. Ramakrishnan, and Terrance Swift. A rule-based data standardizer for enterprise data bases. In *Int. Conf. on Practical Applications of Prolog (PAP'97)*, pages 271–289, 1997.

[27] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 302–314. Morgan Kaufmann, September 1999.

[28] Dimitri Theodoratos and Timos Sellis. Data warehouse design. In *Proceedings of the Twentythird International Conference on Very Large Data Bases (VLDB'97)*, pages 126–135, 1997.

[29] Volker Turau. Db2xml. `http://www.informatik.fh-wiesbaden.de/~turau/DB2XML/`.

[30] Volker Turau. Making legacy data accessible for xml applications, 1999. Available on `http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html`.

[31] Janet L. Wiener, Himanshu Gupta, Wilburt J. Labio, Yue Zhuge, Hector Garcia-Molina, and Jennifer Widom. A system prototype for warehouse view maintenance. Technical report, Stanford University, 1996. Available at `http://www-db-stanford.edu/warehousing/warehouse.html`.