An Approach to the Integration of File Based Systems into Database Federations*

Michael Höding
Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: hoeding@iti.cs.uni-magdeburg.de
Tel.: ++49-391-67-12220 Fax: ++49-391-67-12020

Abstract

In this paper we sketch an approach to the integration of file based systems in database federations. For that the main problem is the lack of meta-information for local file systems. Therefore, we have to derive an appropriate local conceptual schema. Our approach is based on a classification of local file systems. In the first step a local file system has to be assigned to one or more classes accordingly to the classification. In the next step, class-specific methods and tools support the derivation of a structure description and, furthermore, of a conceptual schema.

Keywords: federated database design, file integration, legal systems, reengineering.

1 Introduction

Considering the situation in many organizations and enterprises we discover a number of different information systems in use. However, in each part of the organizations (or even worse for each specific task) a specialized information system is used. Most of these information systems have to store their important data in databases. For this purpose often database management systems are in use. On the other hand, many other information systems are based on specific files. In consequence, there are now different kinds of heterogeneous database systems with different kinds of interfaces and many independently developed application programs.

The integration of the independent databases is a promising task which can help the organization to be competitive while the market is changing faster and faster. The possibility of a uniform access to data from different databases allows a comfortable and uniform representation of the information. There are different approaches to integrate the different heterogeneous local systems. We believe, the most suitable approach is the use of a federated database system (FDBS) [HM85, SL90]. An FDBS provides a uniform and transparent access to federated data by means of a global interface. For that a special software layer, called the federated database management system, is added to the database systems. Moreover the autonomy of the local systems is preserved. Therefore, local applications can be used with no changes. Furthermore, an FDBS provides semantical integrity between related data in the independent heterogeneous local databases [CHJ+96]. In the FDBS design process heterogeneity on data model and schema level has to be overcome [BLN86]. Thus we can distinguish the schema transformation and the schema integration step. The former design step deals with the transformation of a local schema into a schema based on a common data model. In the latter, the transformed schemata are integrated into one schema. The possible derivation of external schemata provides user specific views on the integrated data.

In this paper we will focus on the integration of file systems into an FDBS. **Here, the words *file system* refer to as a set of files which are used by one or more application systems for persistent data storing.** In fact, many applications for special purposes store their data in files using specific file formats [ACM93, HTJ+95]. However, the data could also be useful for global applications. Moreover, the global integrity of federated data should cover file systems, too. In case of file system integration into an FDBS we have to deal with some additional problems. Commonly, in contrast to database systems, neither a

*This research was partially supported by the German Country Sachsen-Anhalt under FKZ: 1987A/0025 (Federating Heterogeneous Database Systems and Local Data Management Components for Global Integrity Maintenance)

conceptual schema in a defined data model nor a documented interface to the file system is available. The schema, i.e. the *possible* structure and meaning of the files, is often hidden in application programs. The task of reconstructing a suitable description is quite difficult and costly. Such a description is necessary for the design of an interface, which supports access, similar to the interface of a database system.

In the following, we will discuss an approach to improve the process of deriving conceptual schemata for file systems. First, we briefly introduce the foundations of FDBS. After this, we present an example which illustrates the federation of heterogeneous information systems. Then, our approach to the derivation of local conceptual schemata is sketched. The approach is based on a classification of file systems. For every class, a specific set of design methods is reasonable. In the following sections, we discuss the derivation steps for a local file system from the example in more detail. In the conclusion, we give an outlook to future work.

2 Basics of FDBS

In this section we briefly present the essential foundations of federated database systems (FDBS). A federated database system is a distributed system consisting of different heterogeneous database systems or local file systems [HM85]. These cooperating but *autonomous* local systems are coupled by the federated database management system (FDBMS). The FDBS provides a uniform and transparent interface to the heterogeneous and distributed data sources of independently developed local systems. Therefore, new global applications have the possibility to read and write the federated data. The FDBS communicates with the local systems using their common user interfaces. Therefore, the local database systems and application programs can remain unchanged. For federated database systems with global applications the 5-level-schema-architecture of [SL90] is generally accepted (cf. Figure 1):

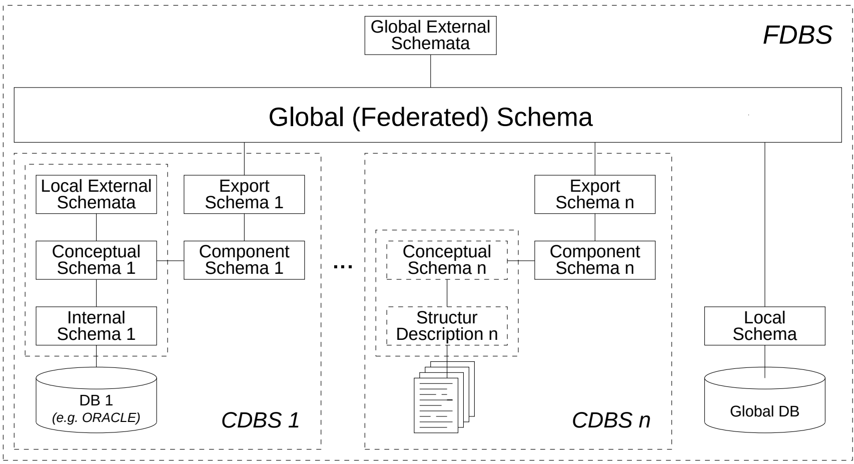

Figure 1: Schema levels of a federated database system

- *Local Schemata*: The local schema is the conceptual schema of the component database system (CDBS). It is expressed in the data-model of the component system. On this level, commonly, schemata are based on different data-models. However, a conceptual schema for file systems is not available. Therefore, the derivation of a local conceptual schema is necessary.
- *Component Schemata*: To overcome data-model heterogeneity, the local schemata have to be transformed from the native data-model to a common data-model.
- *Export Schemata*: Export schemata filter a subset of schema information. In other words, only exported schema elements and data are visible to the federation layer.
- *Federated Schema*: The federated schema is an integrated schema, which is the result of the integration of the export schemata. On this level schema heterogeneity is overcome. The federated schema is the schema of the global interface.

- *External Schemata*: External schemata serve specific views for different applications. This can include the the change of the data-model.

In conclusion, we can distinguish two main tasks for the design process. These are the *schema transformation* to overcome the data-model heterogeneity and the *schema integration* to overcome the schema heterogeneity.

For the design of federated database systems different integration methodologies are available. Many methodologies use a semantical rich data-model, e.g. an object-oriented data-model [CSGS94, RBB+95, TS93]. Other integration methodologies are based on semantical poor data-models. For example, the *Generic Integration Model* GIM [SS96], which is developed especially for the FDBS design, can be characterized as a semantical poor data model. The choice of the integration data model influences the derivation of conceptual schemata for file systems. Using the common data model or a similar data model for modeling an conceptual schema makes the derivation of the component schema unnecessary or easier.

3 An Example Scenario

In this section we describe an example of a heterogeneous information system. This example shows the problems of file system integration in more detail. Furthermore we will use the example for the illustration of our approach. We consider a scenario of an enterprise producing and selling transport machines. In the enterprise three information systems are in use. First, we find the system in the production department which manages information about roller conveyors, the only type of transport machines, which are produced by the enterprise itself. The system is based on a object-oriented database. Other types of transport machines, e.g. belt conveyors or fork lift trucks, come from other manufacturers. The information system of the public relation department presents a product catalogue in form of structured WWW pages in the Internet. These pages show the self-produced roller conveyors *and* the other transport machines. We have also some additional attributes in the WWW "database", e.g. `price` and `picture file`. The third information system is used by the dispatch department. This system manages information about deliveries and customers and uses specific files for data storing.

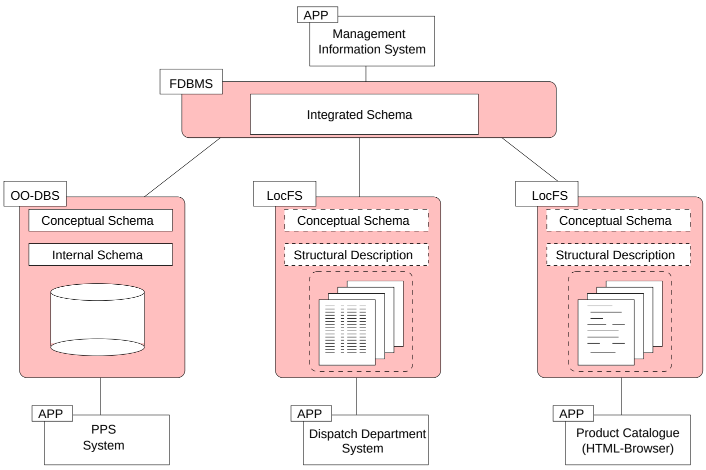

Figure 2: Example of a heterogeneous information system

One goal of the integration of these local information systems is to maintain global integrity. For instance, all products in the production database should be automatically presented in the product catalogue or products which are contained only in the catalogue could be shipped by the dispatch department. Establishing a global management information system is another goal of the integration.

Whereas we find a defined schema for the production information system, a schema of the other information systems does not exist. Therefore, we have to model equivalent schemata for the other two systems. Due to the space restrictions, we will discuss the derivation of a conceptual schema only for the file system of the dispatch department. Therefore, we list a small extract from the example files, which gives a first impression(cf. Figure 3).

```
-->plane1996.tra<------------------------------------------------------------
...
26-06-96;4501009;fork lift truck XX27  ;Macrosoft Corp., Seattle;  ;BA-1033
26-06-96;8111031;short RC DIN 42/4711  ;Town Library, Bonn     ;OK;LH-706
26-06-96;8612343;long DRC DIN 42/5573  ;Fosters Brewery, Sidney;OK;MA-17
...
27-06-96;4501009;fork lift truck XX27  ;Magnigum Ltd.,Moscow   ;OK;AF-380
--------------------------------------------------------------------------
-->train1996.tra<------------------------------------------------------------
...
14-05-96;4501009;fork lift truck XX27  ;Fosters Brewery, Sidney;  ;ICE-706
14-05-96;8612343;long DRC DIN 42/5573  ;Pivovar Brewery, Plsen ;OK;ICE-706
...
29-06-96;8612343;long DRC DIN 42/5573  ;Becks Brewery, Bremen  ;  ;ICE-345
...
--------------------------------------------------------------------------
```

Figure 3: Example files of the dispatch departments database

4 Towards an Efficient File Schema Derivation

In this section we will focus on the derivation of conceptual schemata from local file systems. First, we briefly discuss the problems and define the goals of the derivation step. After this, we propose a classification for similar file systems. Based on this classification we sketch a systematic and thus efficient way for the derivation. Last we discuss the different information sources for the derivation.

As mentioned before, we have the problem of missing schema information for local file systems. The meta information is hidden in the application programs or it is merged with data in the files. Hence, *it is necessary to derive a schema which describes the structure of the file system.* This means not only the structure of the current state of the file system, e.g. represented by example files, but also of every possible state. However, such a structure description, e.g. a grammar, is not very understandable for the designer. Therefore, a second representation in a more understandable notation and a mapping to the structural description would be useful. To prevent the necessity of additional transformation steps the common data model of the FDBS should be used for this purpose. These two schema levels can be compared with the internal schema and the conceptual schema according to the three-level-architecture of ANSI/SPARC [DAF86].

Another problem in file systems is the lack of documented user interfaces. An access to the files is only possible via the common file interface of the operating system. This interface allows a number of simple file access operations which can be considered as a primitive data manipulation language. Combining the previously derived schemata with a generic access module, we have to build *an interface which allows the access to the files similar to database system.*

In conclusion, the goal of the derivation process is the derivation and modeling of a structure description and, furthermore, the design of an access module. Since an individual and manual derivation is very costly a systematical approach, combining different techniques and experiences, should help to make the process easier.

4.1 Classification of File Systems

Nowadays a large number of specific file formats are in use [Bor95]. The manual derivation of a suitable schema is very costly and has to be done again and again. However, most of these specific formats could be grouped according to their similarities. This classification can help to make the process of schema derivation easier, because file formats based on similar concepts need similar derivation methods and tools. For a systematical classification we can use different aspects, e.g. *structure concept of the file system*, *granularity of read/write operations*, *available sources for meta-information*, and *access rights of the FDBMS.*

A more empirical classification is based on practical experiences, especially as a result of the investigation of PC systems for factory planning [HTJ+95]. In the following, we introduce some interesting classes of file systems. These classes, in our opinion, cover a wide range of file systems in use.

Record oriented Files
Simple record oriented files containing no meta-data are often used for specific applications, which are

based on programming languages supporting record types. These systems often use random access single record read/write operations because immediate storing of data in the permanent memory facilitates more safety. As a result of the high granularity, they are good candidates for participation in a database federation. On the other hand, it could be quite difficult to derive an equivalent schema, because meta-information is completely hidden in the application programs.

xBase Files
The xBase file format is used by many PC applications [Bor95]. It is well-known and well-defined. A single file stores information like a relation according to the relational data model. Beside this, every file contains meta-information describing the structure of the records, which could be stored in the file. Moreover, in many xBase-like file systems a specific file exists which manages meta-information about existing files in the database. Additionally, some file systems contain index files. Due to the high availability of meta-information, the integration of xBase file systems into a database federation is quite easy. Thus, the derivation of a local schema can be done automatically.

Completely Structured Text with Meta-Information
This class of files is often used to support (human) readable exchange formats. Here, meta-information and information are merged. We find hierarchies of meta-information which determines (a set of) data. (The hierarchies, sometimes, support complex attributes.) At first glance, the manual derivation of an equivalent local schema from example files seems easy, because the meta-information associates information in a quite intuitive manner. Since we have to deal with a large number of complex example files, this task has to be supported by tools collecting information about similarities and differences in the available example files.

Partially Structured Text with Meta-Information
Since, in some cases, not all local data could or should be federated we look at a file system as text with some structured components. An example for such a structured component could be the phone number of an employee, written in the letter of a company, which should be confirmed by the database system of the management. For that, an integrating FDBMS has to find the name of the employee and the phone number. The employee could be represented by his name, i.e. his name might determine that it is an employee (meta-data) sending the letter and also that he is *the* employee (data). The telephone number could be determined by a leading meta-information, e.g. `Phone:`, `Telephone:` or `Phone-No:`, example phone numbers from the database or by patterns which are typical for phone numbers. However, this information is rather fuzzy because the employee could also be the receiver of the letter or the phone number references to another person mentioned in the text. Therefore, write operations by the FDBMS could be very dangerous. Nevertheless, the integration of such semi-structured file systems could help to maintain more integrity in the enterprise.

Hypertext
Hypertext file systems are partially covered by the patially-structured text files. One of the most popular examples is the HTML format, which defines the depiction of a document available by the WWW. Beside the graphical meaning some HTML expressions, e.g. the table construct, give hints for a schema derivation. Moreover, the powerful construct of hyper-links could be used for information structuring, e.g. to present the hierarchical structure of a product catalogue (cf. section 3). Due to the fact, that hypertext documentations become more and more popular, the generation and maintenance of up-to-date hypertext files by the means of a FDBS is very interesting.

For these classes specific characteristics have to be formalized. Furthermore, specific methods for the derivation of local schemata for every class have to be established. Another research task is the improvement of this quite empirical, and therefore incomplete and less orthogonal classification.

4.2 A Sketch of our Approach

In the following we sketch the steps of our derivation approach (cf. Figure 4). Beforehand, the possible information sources[1] are listed:

- example files: $\mathcal{D}_{ExampleFiles}$
- informal background knowledge: $\mathcal{K}_{Background}$
- file access information/logfiles: $\mathcal{M}_{AccessInfo}$

[1] $\mathcal{D}$ means Data; $\mathcal{M}$ means (formal) Meta-data; $\mathcal{K}$ means (informal) Knowledge

- source code of application programs: $\mathcal{M}_{SourceCode}$
- common dictionary - formal common background knowledge: $\mathcal{M}_{CommonDict.}$

These information sources will be discussed in section 4.3 and section 5.1 in more detail. Depending on

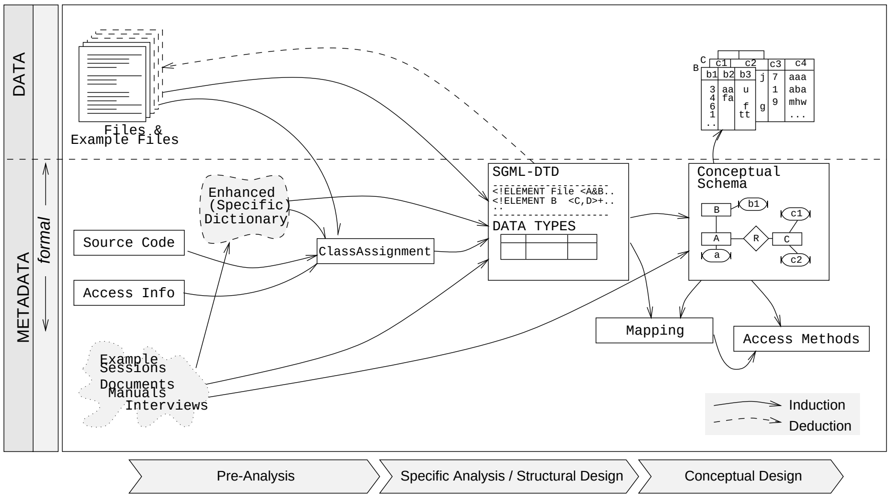

Figure 4: Steps for schema derivation

the information sources we propose the following steps for the derivation process:

1. **Pre-Analysis**: common methods for all classes
   (a) **Background-Knowledge Formalization**: extend common dictionary by specific data
   $\mathcal{M}_{CommonDict.}, \mathcal{K}_{Background} \rightarrow \mathcal{M}_{EnhancedDict.}$
   (b) **Class Assignment**: assign the file system to a class according to the classification
   $\mathcal{M}_{EnhancedDict.}, \mathcal{M}_{AccessInfo}, \mathcal{D}_{ExampleFiles}, \mathcal{M}_{SourceCode} \rightarrow \mathcal{M}_{ClassAssignment}$
2. **Specific Analysis/Structural Design**: use specific methods and tools according to the assigned class to derive information which supports the (manual) generation of a structure description (e.g. expressed in SGML enriched with datatype definitions)
   $\mathcal{M}_{ClassAssignment}, \mathcal{M}_{EnhancedDict.}, \mathcal{D}_{ExampleFiles}, \mathcal{K}_{Background} \rightarrow \mathcal{M}_{StructuralDescription}$
3. **Conceptual Design**: design of a conceptual schema according to the structural description
   $\mathcal{M}_{StructuralDescription}, \mathcal{K}_{Background} \rightarrow \mathcal{M}_{ConceptualSchema}$

In the following we will explain these proposed steps in more detail by means of our example.

4.3 Specific Information Sources

Now, we have to discuss the mentioned specific information sources for the schema derivation in more detail, because the efficient analysis of the basic information is really important for the following analysis steps.

- *Background Knowledge*
  Often, in information systems the meaning of data is hidden in a documentation or it is knowledge of the user. Therefore, the first step is the analysis and, if possible, the formal description of important information. We can distinguish between common knowledge, e.g. “phone” has the same meaning as “telephone”, and application system specific knowledge, e.g. “fork lift truck” as an example for a “transport machine”. Moreover, we can classify knowledge about data and meta-data. These aspects will be discussed in section 5.1 in greater detail.

- *Example Files*
  Commonly, the first intuitive approach is an examination of example files. The analysis has to cover not only the structure of files themself but also file and pathname, which can also hide meta-information. Please note that example files represent only one valid state of the file system. The examination of a large number of example files will lead to better results. However, such an examination of many files has to be supported by examination tools mining for possible meta-information. Another more meaningful possibility is the analysis of files which are manipulated as a result of a special example session with the application system. Since this is a very costly process, this can only be done in some very critical or important cases.

- *Access Information*
  Quite useful is information about the access operations of application programs to the files. Especially when more than one application (or instances of an application) use a file system, these information give hints for the analysis. The most interesting point is the investigation of the granularity of access information. In many cases, file based systems use unsafe "load" and "save" operations for reading or writing complete files. Other applications read and write data in random access. Access information can be delivered by a modified file server.[2]

- *Source Code*
  In rare cases source code of application programs is available. Therefore the complete reconstruction of the schema, which includes all possible file extensions, is possible. However, the automated examination of source code is a very difficult and costly process. This means, an experienced system developer has to do this job manually for this purpose.

As mentioned before, the availability and the quality of this information is very important for the quality of the derived schema. As much as possible formalized and therefore automatically compressible information should be collected and classified.

5 Pre-Analysis

The goal of the pre-analysis step is the condensing of basic information. The first result of this step is an enhanced dictionary containing common and specific knowledge about the local system. Based on this enhanced dictionary and the other information sources, e.g. example files and access information, the file system is assigned to one or more classes of a predefined set of classes, such as presented in section 4.1.

5.1 Formalization of Background Knowledge

As mentioned above a method catching the common and specific background knowledge is necessary. Our idea is to use a dictionary that contains common information and which could be enhanced during the analysis of background information. Due to the the space restrictions, we only discuss the kind of information in this dictionary, shown in (cf. Figure 5). The goal is to find the meta-information according to the files. The meta-information is sometimes represented by an attribute or object name in the file. In other cases, typical information indicates meta-information. Therefore, we distinguish a part of the dictionary containing meta-data-expressions, which may refer to other meta-data-expressions, e.g. synonyms, too. The other part of the dictionary contains data-expressions that refer to meta-data-expressions. Moreover, in the common base dictionary we use two different kinds of data-expressions. The first are simple example data, e.g. the number `1996`, which implies the assumption that this data element refers to an object or attribute `year`. The other more powerful kind are patterns for very common value types, e.g. for date or flight numbers. During the examination of documentations and user programs the designer should collect his results in the dictionary. In that way, the dictionary can be enriched with specific data-expressions, e.g. examples for `product_no`, and with possible meta-data-expressions, e.g. the table or column names known from specific application programs. This enhanced dictionary can help to find hidden hints during the example file analysis. Obviously, the presented dictionary is a simplified solution of the problem but a useful approach to catch the specific informal information for the next steps. In conclusion, the enhanced dictionary contains:

- different patterns, e.g. for dates (common information)

[2] We made some successful experiments with a modified SAMBA file server server, which supervised the access of PC applications to the global file space of a UNIX server.

- possible object and attribute names (added meta-information)
- their synonyms (common meta-information)
- example data (added information)

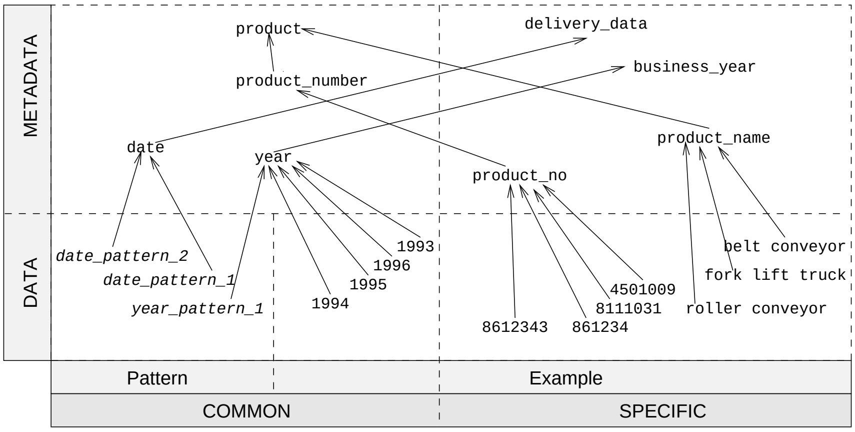

Figure 5: Common and specific meta-data and data in the enhanced dictionary

Accordingly, we start the derivation of the local schema for the dispatch information systems in our example with an analysis of the background knowledge. Thereby, we find useful information in the manuals and the application programs. We know that the information system stores for every delivery date, product number, product name, address of the customer, the confirmation state and the train number. For instance, we should add as many as possible valid example data to the dictionary which are used in the company, e.g. for product number or product name,. Moreover, the possible attribute names, e.g. the date, product, etc., should be enclosed in the enhanced dictionary. Furthermore, we recognize three installations of the system; each one for deliveries by train, ship, or plane. In the installations for ship or plain deliveries the train number attribute is used as a ship or flight number. Hence, the pattern for flight and train number from the common dictionary could bring some matches. In conclusion, the added attribute names associate other possible attribute names by synonym-links in the base dictionary. They represent meta-data, whereas the example product numbers represent data.

5.2 Class Assignment

The next step is the tool-based analysis of the example files. The first information source is the enhanced dictionary. A second information source is log file information about the access of the applications to the files. For our example, an important result is the observation, that the application programs read and write fix records with the size of 73 bytes. Beside this, we recognize that the different installations, e.g. for ship or plain shipping, access to different (own) files. For our example system no source code is available.

For the analysis, a pre-analysis tool searches the example files for matching words in the dictionary according to the available access information. We can conclude as the result of the pre-analysis step for the example system :

- The suspected meta information, e.g. `date`, cannot be found in the example files. We assume that the files do not contain meta-information, i.e. the schema is completely contained in the application program.
- We find matching information patterns according to the common dictionary, e.g. the pattern for date.
- We find matching example information according to the enhanced dictionary.
- Moreover we detect the delimiter **EOL**[3] on every 73rd position and the delimiter **;** in regular distance to the position modulo 73.

Based on these results and the source information we can classify the system as *record oriented file system.*

[3]end of line

6 Specific Analysis

The goal of the specific analysis step is the derivation of a structure description for a local file system. Due to the fact, that neither the example files nor the other information sources contain enough information about the semantics of the local file system in the main this process cannot be done automatically. Mostly this process has to succeed manually. We suggest the use of SGML[4] [ISO96] for the structure description of the local files. In our opinion, SGML is more suitable than a grammar because it is understandable *and* formal. Moreover, SGML is quite popular, hence the widespread HTML format, which is an instance of SGML or, in other words, a schema according to the data-model SGML. Such a SGML schema is called a document type definition (DTD). On the other hand, we miss datatype concepts in SGML. The lowest granularity is the expression `PCDATA`, which means "parsed character data". Therefore, we have to add information about data types to the structural description, e.g. in table form. Based on the SGML-DTD and a data type table a parser could be derived.

Due to the fact that the design process, especially in case of complex or large file systems, is very extensive, specific tools for the different classes should provide the designer with condensed information derived from the source information. For instance, specific parsers can be used for the analysis of a structured text. For record oriented file systems, a tool, which will find record borders in example files could deliver information about possible attribute positions. In case of the well-defined xBase format, a very specialized tool could derive the local schema almost automatically. (The only task of the designer is the improvement of the resulting schema.) We can conclude: An analysis tool should provide the designer with as much as possible condensed information.

We will sketch the analysis for our example of the dispatch department's information system. For the analysis we again use the enhanced dictionary. The tool reads all available example files according to the suspected record structure (record size 73 and regular ; according to position modulo 73). For every record element the algorithm searches in the enhanced dictionary for matching data or meta-data. The result of a successful search is the possible meta-data expression. Due to the fact that we have many files with a large number of records, the number of matches in comparison with the total number of tested records can determine the meaning of the attribute, e.g. as shown in Figure 6.

```
1st element: size 8 , delivery_date , matches 100 % date_pattern_1
2nd element: size 7 , product_number, matches 7 % examples
3rd element: size 22, product_name , matches 7 % examples
4th element: size 24, customer , matches 15 % examples
5th element: size 2 , state , matches 78 % examples
6th element: size 7 , train_no , matches 41 % train_no_pattern
6th element: size 7 , train_no , matches 7 % examples
6th element: size 7 , plane_no , matches 24 % flight_no_pattern
6th element: size 7 , plane_no , matches 5 % examples
6th element: size 7 , ship_no , matches 4 % examples
```

Figure 6: Output of the analysis tool

The first element matches completely with a pattern for date, and this is linked in the enhanced dictionary with the expected meta-data-expression `delivery_date`. The match rate for the second element is much lower because these matches are only based on some example for product numbers. This problem of a quite small number of examples data is also the reason for the low example matching rate for the third, forth and sixth element. However, we got many matches for the sixth element according to available patterns for train or flight numbers in the common dictionary. However, problems which occur because of homonyms and synonyms in the dictionary could only manually solved by the designer. For instance, if the dictionary delivers the meta-data-expression `product` for the second and the third element the designer has to solve the problem. We can formulate the structure descriptionBased on the examination of the output and on the background knowledge (cf. figure 7).
This structure description is a hypothetical schema for the file system of the dispatch department's information system. Based on this formal and quite understandable description the automated generation of a parser is possible. With the derived parser the structure description, which is a result of an inductive and deductive design process, the hypothetical schema could be proved.

[4] Standard Generalized Markup Language

```
---SGML-DTD:dispatch_department<----------------------------
<!ELEMENT disp_file - - (delivery_record)+                 >
<!ELEMENT deliv_rec - - (date,prod_no,prod_descr,cust,train)>
<!ELEMENT date      - - #PCDATA                            >
<!ELEMENT prod_no   - - #PCDATA                            >
<!ELEMENT prod_descr- - #PCDATA                            >
<!ELEMENT cust      - - #PCDATA                            >
<!ELEMENT state     - - #PCDATA                            >
<!ELEMENT train     - - #PCDATA                            >
------------------------------------------------------------
```

Data Type Definition Table: `dispatch_department`

Name	Size	Data Type
date	8	`char(8)`
prod_no	7	`char(7)`
prod_descr	22	`char(22)`
cust	24	`char(24)`
state	2	`char(2)`
train	7	`char(7)`

Figure 7: Structural description of the dispatch departments database

7 Conceptual Design

In this section we sketch the conceptual design step. The conceptual design is the last step of the derivation process. We assume that a correct structure description of the local file system is available. The goal of this step is to design a more understandable schema expressed in a well-known data-model. The use of the common data-model is a good choice because in this way we can save one transformation step.

The design process is related to the well-known conceptual design step of the database design process. However, in contrast to this methodology we could and should consider the information sources mentioned above, especially the example files which represent possible extension for the schema. Moreover, the analysis of the example files can help to find violations of the first normal form in the local file system. These problems have to be overcome in the conceptual design step.

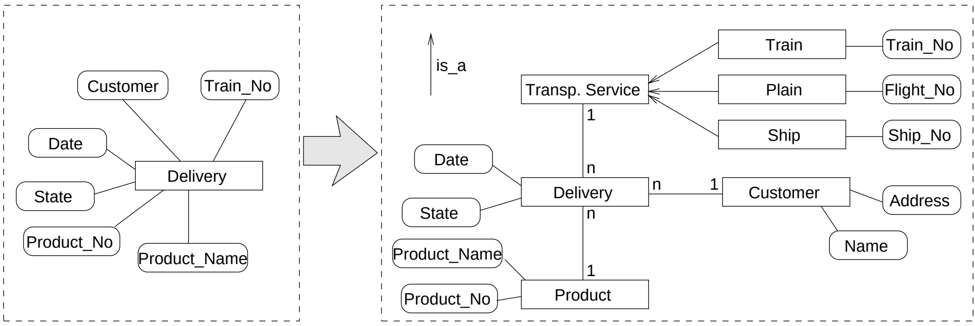

Figure 8: Improvement of the conceptual schema for the dispatch department's information system

For the record oriented file system of the dispatch department, the specific analysis tool or the derived parser could detect such violations of the first normal form. Therefore, we can improve the first approach of a single class schema (cf. Figure 8). For instance, identical pairs of `product_no` and `product_name` indicate the assumption that we can establish an extra class for this pair of attributes. Another possibility for an improvement is the discovery that the sixth element stores the identifier of the transport facility as well as that it indicates the kind of the transport service. This discovery is also expressed in the improved conceptual schema (cf. Figure 8).

Based on the derived conceptual schema, mapping to the structured files and a generic interface the local files system could be integrated in the FDBS according to common integration methodologies , e.g. [SS96].

8 Conclusion and Outlook

In this paper we suggested an approach for the integration of file based information systems in a database federation. Because of the problem of missing meta-information for local file systems, we focus on the derivation of an equivalent structure description and, furthermore, a conceptual schema. Our approach is based on an empirical classification and specific methods and tools for each class. In the first step of the derivation approach, a concrete file system is classified according to the classification. In the second step, specific methods and tools support the designer in the derivation of a structure description and, moreover, the generation and improvement of a conceptual schema. Due to the space restriction we discussed these steps only for one class by means of an example.

For further investigations a less empirical classification has to be derived. Furthermore, methods and tools for the specific tasks should be outlined. An enhanced formal description of the problem, the information sources and the sketched steps can make the process clearer. We are currently working on these problems. Additionally, we try to integrate the designed methods into our FDBS design framework called $\mathbf{SIGMA}_{Bench}$.

Thanks to Andreas Christiansen, Stefan Conrad, Andrea Diekmann, Eyk Hildebrandt, Andreas Köller, Gunter Saake, Ingo Schmitt, Kerstin Schwarz and Can Türker for many creative discussions.

References

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and Updating the File. In *Proc. of the 19th VLDB Conference*, pages 73–84, Dublin, Ireland, August 1993.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[Bor95] G. Born. *Fileformat Handbook*. International Thomson Publishing, London, 1995.

[CHJ+96] S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, April 1996.

[CSGS94] M. Castellanos, F. Saltor, and M. Garcia-Solaco. A Canonical Model for the Interoperability Among Object-Oriented and Relational Databases. In T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 309–314. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[DAF86] Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group DAFTG. Reference Model for DBMS Standardization. *ACM SIGMOD RECORD*, 15(1):19–58, 1986.

[HM85] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[HTJ+95] M. Höding, C. Türker, S. Janssen, K.-U. Sattler, S. Conrad, G. Saake, and I. Schmitt. Föderierte Datenbanksysteme — Grundlagen und Ziele des Projektes $\mathbf{SIGMA}_{FDB}$. Preprint Nr. 12, Fakultät für Informatik, Universität Magdeburg, December 1995.

[ISO96] ISO (International Organization for Standardization). *The Standard Generalized Markup Language.* (ISO 8879), 1996.

[RBB+95] E. Radeke, R. Böttger, B. Burkert, Y. Engel, G. Kachel, S. Kolmschlag, and D. Nolte. Efendi: Federated Database System of Cadlab. In M. J. Carey and D. A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, SIGMOD RECORD 24(2)*, page 481. ACM Press, June 1995.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SS96] I. Schmitt and G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In *Proc. of the 15th Int. Conf. on Conceptual Modelling (ER'96).* LNCS, Springer-Verlag, Berlin, October 1996. *To appear.*

[TS93] M. Tresch and M. Scholl. Schema Transformation Processors for Federated Objectbases. In S. C. Moon and H. Ikeda, editors, *Proc. of the 3rd Int. Conf. on Database Systems for Advanced Applications (DASFAA'93), Daejon, Korea*, pages 37–46. World Scientific Press, April 1993.