Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte
FZKA 5669

Integrated Data Management for RODOS

**K. Abramowicz, A. Koschel, M. Rafat,
R. Wendelgass**

**Institut für Neutronenphysik und Reaktortechnik
Projekt Nukleare Sicherheitsforschung**

Dezember 1995

Forschungszentrum Karlsruhe
Technik und Umwelt
Wissenschaftliche Berichte
FZKA 5669

Integrated Data Management for RODOS

Karol Abramowicz*, Arne Koschel*, Mamad Rafat**, Ralf Wendelgass*

Institut für Neutronenphysik und Reaktortechnik
Projekt Nukleare Sicherheitsforschung

*Forschungszentrum Informatik (FZI), Karlsruhe

**Fa. D.T.I. Dr. Trippe Ingenieurgesellschaft m.b.H., Karlsruhe

Forschungszentrum Karlsruhe GmbH, Karlsruhe
1995

This work has been performed with support of the European Commission,

DGXI-A-1, under contract No. 94-PR-006

Als Manuskript gedruckt
Für diesen Bericht behalten wir uns alle Rechte vor

Forschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

ISSN 0947-8620

Abstract:

The report presents the results of a feasibility study on an integrated data organisation and management in RODOS, the real-time on-line decision support system for off-site nuclear emergency management. The conceptual design of the functional components of the integrated data management are described taking account of the software components and the operation environment of the RODOS system.

In particular, the scheme architecture of a database integration manager for accessing and updating a multi-database system is discussed in detail under a variety of database management aspects. Furthermore, the structural design of both a simple knowledge database and a real-time database are described. Finally, some short comments on the benefits and disadvantages of the proposed concept of data integration in RODOS are given.

Integrierte Datenverwaltung in RODOS

Kurzfassung:

Die vorliegende Studie dokumentiert die Ergebnisse eine Machbarkeitsanalyse zur integrierten Datenorganisation und Datenverwaltung in RODOS (real-time on-line decision support), dem Entscheidungshilfesystem für den Notfallschutz nach kerntechnischen Unfällen. Ausgehend von der Architektur der Betriebskomponenten und der Betriebsumgebung des RODOS Systems wird der Entwurf eines integrierten Datenbankkonzepts beschrieben.

Insbesondere werden die Schema - Architektur eines integrierten Datenbankmanagers zum Zugriff auf ein Multi-Datenbanksystem ausführlich diskutiert und die unterschiedlichen Aspekte der Datenverwaltungsmodellierung hinsichtlich der RODOS Zielsetzung im Detail erläutert. Weiterhin werden einige Ansätze zur Aufbaustruktur einer einfachen Wissensdatenbank und einer Echtzeitdatenbank vorgestellt. Abschließend werden die Vor- und Nachteile des vorgeschlagenen Konzepts zur globalen Datenintegration in RODOS kurz aufgeführt.

Figures

Chapter 1 Introduction

1.1 RODOS - objective

The RODOS project is carried out by FZK (Karlsruhe Research Centre) in cooperation with the D.T.I. engineering bureau as the subcontractor. The objective of the project is to design and implement a distributed information system, which supports decision-making authorities after the release of radioactivity due to an accident. In the future, RODOS shall be applied in Western and Eastern European countries, fulfilling three main tasks:

- supply of data on the current and future radiological situation to decision-making authorities;
- support in the decision on protective measures and countermeasures within the range of competence;
- area-overlapping coordination of locally or regionally initiated measures.

The RODOS system will be installed at a number of places in Europe. They will be interlinked by an efficient and reliable network. When implementing this system, commercial relational database management systems will be employed for both persistent storage and efficient supply of data. Some major components of the RODOS system have already been made available, i.e. heterogeneous databases and several application programmes based on various technologies. These components will have to be integrated in the future system.

The present study, which has been prepared in cooperation with the D.T.I. Dr. Trippe engineering bureau and the Database Systems Division of the Research Centre for Computer Science is aimed at demonstrating the feasibility of integrated data management. This study represents the starting-point for the implementation of RODOS integrated data management, which shall be completed by the end of 1995.

1.2 The FZI

The Database Systems Division of FZI (Research Centre for Computer Science) is specialized in the development of database technologies for engineering applications. Main activities focus on the implementation and support of relational and object-oriented database systems as well as on the making of profits on the basis of existing database systems. Up to date, three database systems have been developed by the above Division. Their latest product (OBST) has met with worldwide attention.

1.3 Structure of the study

The present study is structured as follows:

- The RODOS project is defined in chapter 2. To use the system throughout Europe, an appropriate hardware architecture is required. This architecture represents the basis of most design decisions. In RODOS operation, two system modes are distinguished: normal operation (no time-critical response) and emergency operation (time-critical response). In the latter mode, partial functionality is offered only in order to reduce the response time.
- Data management plays the key role in the RODOS system. Mechanisms supporting the integration of heterogeneous data (heterogeneous computers, heterogeneous database management systems, distributed data) have to be made available. When accessing the data, the users and application programmes will not need any information as to where the data desired are stored physically and by which database system they are managed. The architecture and the functionality of the integrated data management component are described in the following chapter.
- The knowledge applied for making a decision and describing the dependences of simple facts and conclusion patterns also has to be managed in RODOS. In chapter 4, several knowledge description formalisms and derivation methods are discussed. Particular attention is paid to the suitability of relational database systems for knowledge management.
- Real-time data represent another special group of data to be managed in RODOS. They describe the current environmental situation and are used as a basis of decisions on the taking of protective measures. After insertion, the measured data are modified in exceptional cases only. As measured data are collected at a large number of measuring points, a large data volume is expected. Management of measured data is discussed in chapter 5.
- Communication of the users with data management takes place via a graphic interface. A survey of the methods for the generation of graphic user interfaces is given in chapter 6.
- One of the major results of the present study is the feasibility analysis of integrated data management in RODOS. This analysis, which is presented in chapter 7, describes the software tools that can be applied in software development. Database integration tools, expert system shells as well as generators of user interfaces allow to reduce the prototype software implementation expenditure and contribute to facilitating the maintenance of RODOS.
- The study is concluded with a summary of the results achieved and the references.

Chapter 2

RODOS - project definition

2.1 Objective of RODOS, FZI contribution

The RODOS system shall serve as an integrated information system in the area affected by an accident-induced release of radioactivity. As the consequences of such accidents can hardly be localized, supra-regional application of the system is planned. The information on RODOS is taken from [DTI94] and talks with the project partners FZI and DTI.

In addition to the purely technical requirements, RODOS will have to meet a number of complex coordination and organization tasks. Some of the organizational measures to be specified, such as the existence of a central person or group of persons to contact for RODOS data management, will also influence the FZI work and decisions.

In general, the tasks of RODOS can be divided into two phases. The first phase represents _normal operation_, which means that no time-critical accident has to be dealt with. In this phase, environmental data of all kind are collected, analyzed and evaluated regularly. Hence, the system is mainly used as a pure information system. The second phase represents _emergency operation_. A time-critical (acute) accident has to be handled. In this phase, RODOS will be used as a so-called emergency system. Current situation data (facts) are acquired automatically by measuring devices and manually by e.g. monitoring teams, the police and the fire brigade. Then, the data shall be represented. In addition, the system shall be applied for decision support on the basis of the current facts. Knowledge evaluation tools shall provide the authorities responsible for emergency response with recommendations as to how to proceed in the respective situation. The facts for instance may include data on the spreading of a radioactive cloud, data on the courses of roads and their current usage, etc. On the basis of rules, e.g. emergency plans shall then be proposed for groups of persons. All these functions shall be carried out by the system in the second phase with update intervals of ten minutes. Such time requirements are also referred to as soft real-time requirements. Furthermore, the system has to meet reliability and fail-safeness requirements. These aspects shall be explained in further detail below. Having the tasks described above, the system can be allocated to the class of so-called accident management systems or emergency systems [BrToRo94, MetA194, DSS93].

As many more or less standardized data management tools already exist (DBMS products), the most important task now consists in developing an integration environment *for* the RODOS system. In the future, any tools, e.g. tools for knowledge evaluation, and any data sources in the form of relational databases shall be integrated in this environment. At the moment, however, no precise statements can be made with regard to the type and scope of tools and data.

To increase fail-safeness and access efficiency of the entire system, it is reasonable to establish more than one system centre in Europe. Sub-systems in the form of local workstations or local workstation clusters shall be linked with these centres. Constant data harmonization of the systems shall take place in order to ensure that all authorities involved are provided with the same data and data structure information (schema information). Failures of parts of the entire system and, hence, possibly incomplete information, are put up with. The entire system shall enable the responsible authorities to make the _best possible_ decisions irrespective of their location and the site of the possible accident. After an accident in the area of Karlsruhe, for example, it shall be possible to coordinate the measures to be taken from Bonn.

Development of RODOS up to its final use is intended to be accomplished in a series of prototypes. However, partial results shall be made use of. In spite of the prototype character of the system, robustness and suitability for re-use are of crucial importance. Within the framework of the RODOS tasks outlined above, the activities of the FZI Division shall focus on some partial components. Above all, fundamental architecture and feasibility results shall be supplied. The following tasks shall be covered by FZI:

- RODOS database integration manager (RO-DIM): design and implementation of a component, by means of which a programming interface (API) is made available to RODOS applications for the use of heterogeneous relational database management systems (RDBMS). _In the upward direction_, this component is to provide the applications with a system-independent view of the data in the form of the API mentioned above. _In the downward direction_, it shall allow the integration of any relational database systems with partially redundant data. Consequently, the rather dynamic use of the data by applications and the rather static definition of integration of all database systems involved have to be distinguished.
- User interface manager (UIM): development of a user interface for integrated data management. This component is to allow comfortable development of simple database applications. It shall contain graphic elements for data definition, data manipulation in the form of updates and (graphic) data evaluation.
- Database design for knowledge management: RODOS allows to manage not only facts, but also relations among facts. For this purpose, a description formalism mapped to a database schema is selected. It is demonstrated, how new knowledge can be inserted using the data manipulation language of the database system and how this knowledge can be accessed in the conclusion.
- The decision process as such takes place within a conclusion component (ESY). This component is implemented as a tool taking the knowledge required from the database. Actually, several conclusion components with various functions (e.g. for tests or the emergency) can be implemented. Implementation of the conclusion component does not belong to the tasks of FZI.
- Design of a database for standardized management of real-time data and in particular time-related measured results: measured data collected around the clock locally, regionally and europeanwide represent an important basis of decisions in the RODOS system. These data,

such as e.g. current radioactivity values, have to be made available to all RODOS participants. In the extreme case, high access and update rates in a short period of time have to be expected.

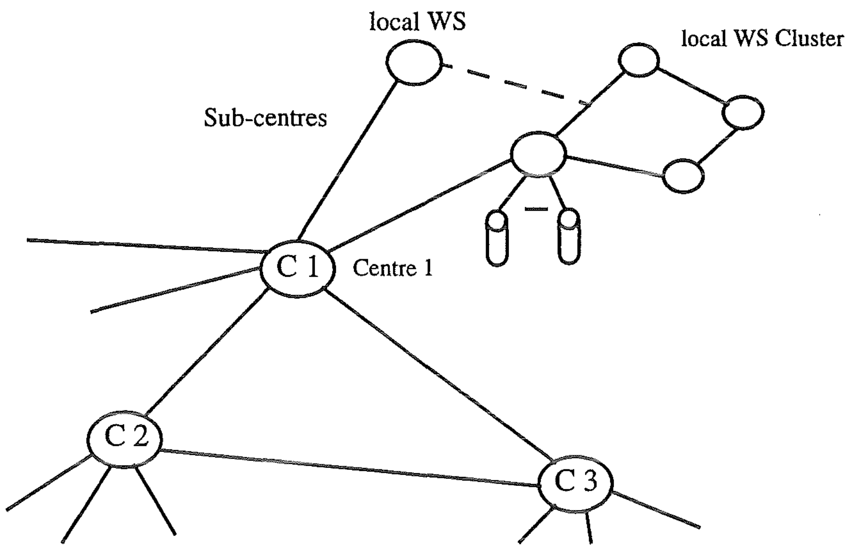
These five tasks have been subject of a number of talks of DTI (Dr. Rafat) and the DBS group (Dr. Abramowicz, Dipl.-Inf. Koschel). The technical and organizational requirements specified by DTI as a result of these discussions served as a basis of decisions in the present study. The partners involved were well aware of the fact that some of the procedures specified would be far from optimum from the technical point of view and represent compromises instead. Within the framework of these specifications, a first rough architecture and feasibility analysis will be performed by FZI for the tasks mentioned above.

2.2 System architecture

2.2.1 Supra-regional system architecture

As outlined above, RODOS shall be conceived as a distributed information system for emergency response in Europe. It is planned to establish three main centres (coordination centres) with several sub-centres (local servers) each. The hardware of the coordination centres is to consist of high-performance computers. In the final RODOS version, all sub-systems involved shall be linked via dedicated lines in order to achieve high data transfer rates. The tasks of the main centres will be the integration and management of the data as well as the description of all connected data sources.

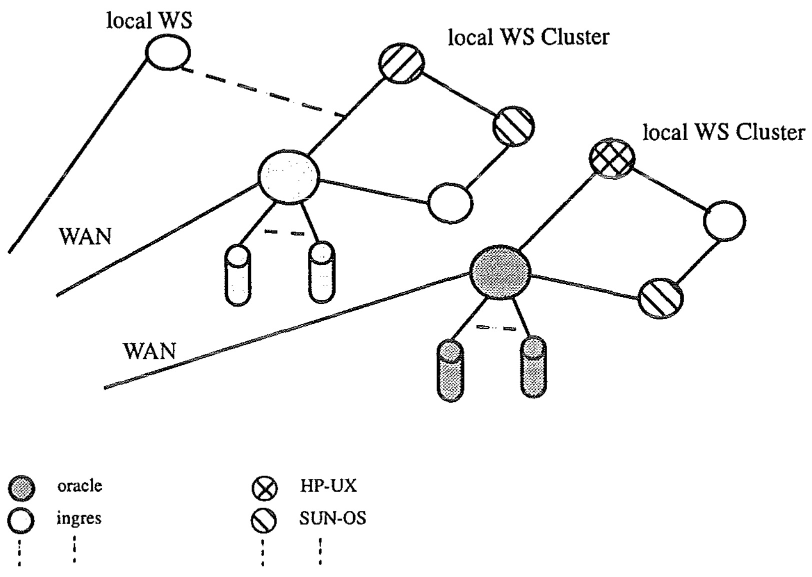
FIGURE 1 upra-regional system architecture



2.2.2 Local system architecture

The sub-centres mentioned above consist of local workstations or clusters of workstations. Various Unix derivates are applied as operating systems. Accordingly, the hardware used is rather heterogeneous. Each sub-centre is equipped with a relational database management system (e.g. Oracle, Ingres) at least. It will be the main task of the locally responsible system managers to describe access to _their_ data correctly and to harmonize updates with the main centres.

FIGURE 2 Local system architecture: heterogeneous platforms, heterogeneous DBMS



2.3 Functional components of RODOS

The entire RODOS system shall be developed as an open information system. It is planned to create an integration environment, in which existing systems can be integrated via defined communication protocols. The major components of the system will be the graphic interface (X-Windows, OSF-Motif), a distributed data maintenance system (relational databases), a communication interface and a so-called integrating operation processing system. The latter is a kind of special mini operating system for RODOS. Communication among the RODOS components shall be accomplished by means of the so-called RODOS message server. This server is connected with clients in the form of external programmes and other RODOS-internal components, such as the integrated data management. In an external application, data management services are not used directly, but indirectly via the general RODOS message server. Furthermore, these applications are completely autonomous, i.e. there is no information known *about* them. They cannot be allocated to a process or cost model and the current state of the system is represented by the data in the databases only. Allocation of the tools to a framework, which can manage such information, is not practicable at all! (Such a model may be used e.g. in a scheduling component for improving the processing sequence of queries or generating alternative access plans. Suggestions above all refer to the fields of query processing in parallel and distributed DB management systems and machine availability planning [BeGr92, Kram92]).

As outlined above, the FZI activities will focus among others on components for distributed data maintenance. The interfaces of distributed data maintenance will communicate with the above message server in the upward direction and serve for the integration of all databases involved in the downward direction.

FIGURE 3 Overall architecture of RODOS

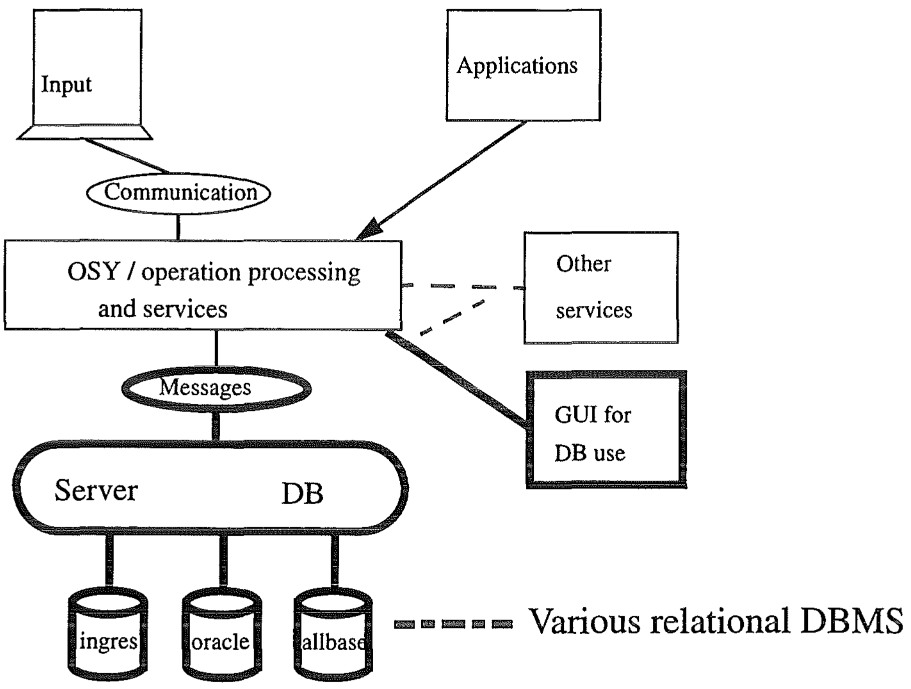
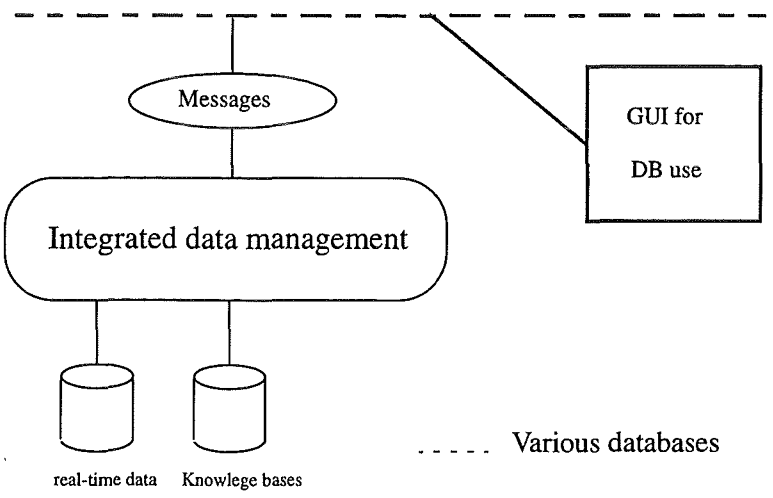


FIGURE 4 FZI contributions to RODOS



2.4 Application scenario

As stated above, the work with RODOS can be divided into two phases, normal operation and emergency operation. Both phases differ considerably which is reflected by the overall architecture of the system. In a report on the experience gained with the FRIEND accident management system in [BrToRo94], some of the differences are listed. They shall be explained in detail in the following chapters.

2.4.1 Phase 1 (normal operation)

The general requirements made on the RODOS system have very much in common with those of traditional distributed information systems. Examples are the World-Wide-Web system, special versions, such as environmental data directories [Sch_u93, HeSc93], geo-information systems, etc. In these systems, data are acquired at various points (in databases) and evaluated using various programmes (in our case, RODOS applications). The respective research activities in the field of databases are classified under the terms of multidatabase systems and federate database systems, the origin of which lies in distributed database systems. These terms shall be explained in detail in chapter 3.

In such systems, data sources may represent e.g. programmes using measuring devices with automatic acquisition or direct user inputs. Data of interest to RODOS are geographical information, such as maps, environmental information on the extent of pollution, meteorological data, etc. In particular, RODOS shall allow long-term access to data of past accidents (e.g. Chernobyl data). If necessary, cooperation mechanisms will have to be provided for this purpose. They will ask the local database administrator to start a tape with old data again.

Consequently, processing of very complex data files may be required in this phase. At that moment, however, the response times of the system are uncritical. Furthermore, the reading operations are expected to exceed by far the writing operations in the data stored. By using automatic devices, minimum requirements may be imposed as far as the response times of the data maintenance component are concerned. Programmes, which supply data of the measuring devices to the data maintenance component, may not be able to buffer them for any period desired.

In this phase, the system will be used above all by users, who want to evaluate certain situations on the basis of the data stored. Other users are persons in charge of emergency response. If required, the data may also be processed and made available for a future general survey.

2.4.2 Phase 2 (emergency operation)

In the second phase of system operation, the tasks to be performed by the system vary considerably. In this phase, an _accident_ shall be managed. A radius of about 50 km around the accident site is covered. (As a consequence, appropriate place and time data have to be supplied.) In intervals of ten minutes, the emergency response staff taking the decisions shall be provided with current information on the entire situation, with the reservation that this information may be incomplete due to system failures or _slow_ communication links. A possible loss of information is accepted! It is assumed that the tools working with this information supply _reasonable_ results even in case of incomplete answering of the queries.

During the second phase, a large set of update information (real-time data) is supplied by the connected measuring stations and has to be processed. At the same time, complex queries to geographical data (for example, for emergency plans), but also to knowledge-based data for decision support have to be expected. This phase represents the highly critical part of RODOS use, as only vague statements exist with regard to the actual data transfer and transaction volume at that time.

High fail-safeness of the system is of crucial importance. Aspects of data integrity have to be taken into account, as the system will have to be made available to authorized users only. During that time, the system will be used above all by the emergency response staff taking the decisions and the staff operation centres. Offices for the information of citizens or staff outside the area of competence will not be authorized for on-line use. These aspects shall be dealt with in further detail below.

2.5 Implementation requirements

As many partners with various systems will participate in the europeanwide use of RODOS, their needs have to be taken into consideration. In this connection, particular attention will have to be paid to system-technological and financial requirements.

2.5.1 General requirements

2.5.2 Budget

The budget of RODOS is limited. This means that a reasonable amount of software tools shall be purchased only. The concrete budget depends on the type and scope of tools. As the system is to be distributed throughout Europe, it is of crucial importance that no royalties will have to be paid for the running of commerical tools. Moreover, the tools will have to be independent of the Unix platform applied.

2.5.3 Hardware

2.5.3.1 Local systems

Well-equipped computers that may be compared with the _large_ HP9000 or Sun-SparcStation-20 systems at least will be used as local workstations. Each computer will have at least 80 MByte RAM and 2 GByte disk storage capacity. It is planned to link the systems via dedicated lines and, thus, ensure a high data transfer rate. Installation of parallel ISDN lines is considered. They allow a data transfer rate of 4 Mbit/sec. at least.

2.5.3.2 Central systems

High-performance systems shall be applied. Minimum hardware requirements have not yet been specified.

2.5.4 Software

2.5.4.1 Operating systems and graphic interface

As basic operating systems of RODOS, any Unix derivates shall be applied, if possible. Consequently, particular attention has to be paid to the portability when developing the components or purchasing the software tools. At least, HP-UX, Sun-OS/Solaris and IBM/AIX are planned to be used. X-Windows and OSF-Motif will serve as graphic interfaces of the system.

2.6 Network, communication

For basic communication within the entire RODOS system, Berkley sockets on the basis of TCP/IP will be available. The addresses of all systems participating in RODOS will be known. This means that links among them will be established from point to point. As availability of SUN-NFS cannot be ensured for all Unix derivates, it shall not be applied. Higher services, such as DCE or CORBA, are considered to be too slow or immature.

2.7 Database management systems

At the moment, it is planned to use relational database management systems for distributed data maintenance in RODOS. Later extension to object-relational systems is desired. Relational DBMS will be _large_ relational systems exclusively. They are largely heterogeneous. Examples are AllBase, Ingres, Oracle and Sybase, but no dBase-compatible or similar systems.

2.8 Programming languages (compilers)

Any Unix-C or C++ as well as Fortran 77 compilers shall be applied for system development.

2.9 Development requirements

Software development requirements have not yet been specified. This shall be done by DTI and DBS after the complete selection of the development tools. The following aspects will have to be considered:

- type and scope of documentation (system use and development documentation)
- interface descriptions.

2.10 Validation scenario

The first FZI prototype shall be tested in a system environment consisting of FZK and FZI computers. In order to make statements with regard to the performance of the prototype, validation requirements have to be specified. These will have to include factors, such as the general system environment, expected amount of data, RODOS applications to be used, etc.

2.10.1 Hardware

Computer hardware of the validation environment shall consist of a HP-9000/8xx workstation at FZK and a Sun-10/40 workstation at FZI. A direct Internet link via sockets shall be used for communication.

2.10.2 System software environment

HP-UX and Sun-OS or Solaris shall be applied as operating systems. The graphic interface is X-Windows and OSF-Motif. The system versions, DB management systems applied, tools, etc. still have to be specified.

2.10.3 Requirements to be met by the prototype

Development of the first FZI prototype will have to meet certain minimum requirements with regard to the technical and organizational environment. These requirements will sometimes represent limitations compared to the boundary conditions described above.

2.11 All phases

It is assumed that the hardware and software system technology applied for the prototype is functioning. This especially applies to the failure of network links, computers or periphery, operating system software, etc. Nevertheless, it shall be possible to re-start a computer system after termination. This must not take place automatically, but can be done manually by a system or database administrator. It is assumed that a competent person will always be available, also in an emergency.

2.11.1 Normal operation

It is necessary to make detailed specifications!

2.11.2 Emergency operation

It is necessary to make detailed specifications!

Chapter 3 Data management

One of the FZI tasks defined in the previous chapter, namely, the component for integrated data management (RODOS database integration manager (RO-DIM)), shall be dealt with in the present chapter. It shall allow integration of all supra-regionally distributed database systems (DB systems) involved in the entire RODOS system. Distribution of these systems is given by organization and therefore has to be maintained. Furthermore, the use of distributed systems generally promises an increased availability of the autonomous parts of the system at least. The individual partners participating in the entire RODOS system shall *keep* their own data and be able to *use* the data of all other partners. Thus, the complexity of the entire RODOS system can possibly be managed.

As far as the applications existing in RODOS are concerned, RO-DIM in principle shall enable any number of them to access simultaneously distributed data in distributed heterogeneous DB systems with local transparency being ensured. For this purpose, a uniform view of all distributed data or parts of them shall be offered to the applications _in the upward direction_. This view shall be an integrating total view, i.e. independent of the physical data distribution. Such integrated data management and the resulting tasks are dealt with in literature under the topic of multidatabase systems. These systems represent an integration level for the database systems involved. Consequently, they do not contain any user data, such as e.g. measured values, but access information *about* the systems involved. _In the downward direction_, RO-DIM shall allow the integration of all distributed databases. In chapter 3.1, the term of multidatabase systems shall be explained in further detail and architectural approaches shall be presented. An intelligible introduction to the fundamentals of such systems is given in [TaVa91, BeGr92], where distributed database systems are described in general. They can be integrated by means of multidatabase systems. Consequently, a distributed multidatabase system shall be investigated here. Another aspect is the heterogeneity of the database management systems applied (Ingres, Oracle, ...). Harmonization of the different data structures and data accesses is required. In our case, the data structures are identical, purely relational systems. However, the access forms vary due to the varying dialects of the access language (SQL).

In comparison with traditional, non-distributed database systems, distributed multidatabase systems bring about some additional problems, which may be attributed above all to the following factors:

- When integrating databases, various forms of redundancies may occur. Two of them are data redundancy and schema redundancy. (The database schema is understood to be the description of the logical data structure of this database. On the lowest level, this may be a description of an individual relation (table) within a database.)
- Let us assume that geographical data on Karlsruhe are stored both in a database in Karlsruhe and a database in Paris. In case of data redundancy, the same data are stored several times in both databases, e.g. two identical street names. This may be useful for increasing fail-safeness and access efficiency of the entire system (intended redundancy) or simply result from the organization of already existing systems. Here, the tasks consist in the treatment of data redundancy during queries and updates in the databases.
- Furthermore, possibly varying schemas, i.e. schema redundancy, may exist in both databases for the description of data structure. Generally, such redundancy is not desired. Both descriptions therefore have to be integrated.
- By europeanwide distribution of the database systems in particular, transfer delays or complete (partial) failures of systems may occur. This has to be taken into account when queries and updates are made.
- Due to the distribution of the database systems, it is impossible to inform all systems at once about updates in other systems. This aspect shall be covered in detail in the chapter dealing with distributed transactions.

These major problems shall be discussed in the following chapters. Furthermore, it has to be taken into consideration that the two soft real-time requirements mentioned in chapter 2 have to be met, as the system is to serve as an emergency system.

3.1 Schema architecture

3.1.1 Multidatabase systems: a survey of architectures

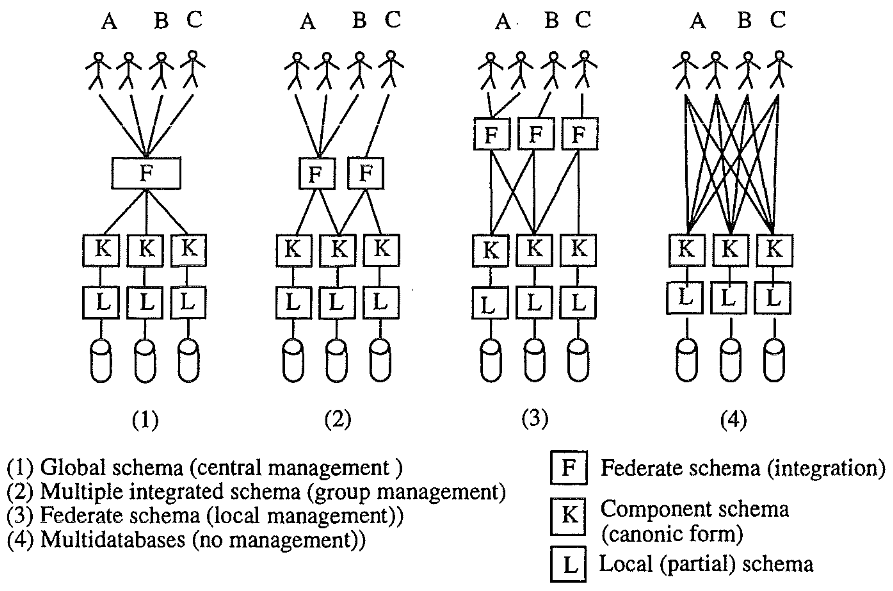
According to [ACM90, TaVa91], the term multidatabase systems applies to systems allowing common access and updates in several database systems. The DB systems involved may be distributed within local or wide networks and based on heterogeneous DB management systems.

However, heterogeneity is not limited to the systems as such, but may also occur in the form of schema redundancy as is indicated in the introduction. In our case, the data structure is represented in various ways, i.e. variable schema information is used for data of the same semantics. (In general, a situation can always be modelled in various ways [Kent89].) Furthermore, redundant data may be encountered in the distributed DB systems. Another aspect is the local autonomy of the individual _data suppliers_. Under certain circumstances, the local database administrators may want to keep control of _their_ data.

The motivation for the use of multidatabase systems results from the fact that all database systems involved shall be commonly used somehow. This means that they have to interoperate. Interoperability is ensured by several architectural approaches for the integration of the participating partial schemas. These architectures shall now be characterized briefly [Fahl94]. None of them is optimum for all cases. Hence, the best suited approach has to be selected in each individual case. All approaches use a standardized (canonic) data model for data representation _in the upward direction_.

From the architectural approaches represented in the following figure, the integration of the schemas of all database systems involved, their representation in a standardized data model with a standardized access language, their integration in one or several federate schemas and the application's view of the integrated data are evident _from the bottom to the top_.

FIGURE 5 Various schema architectures



A B C
A B C
A B C
A B C
F
F F
F F F
K K K
L L L
(1)
(2)
(3)
(4)
(1) Global schema (central management)
(2) Multiple integrated schema (group management)
(3) Federate schema (local management))
(4) Multidatabases (no management))
F Federate schema (integration)
K Component schema (canonic form)
L Local (partial) schema

In the first approach, a *single* so-called *global schema* is used for the integration of all partial schemas involved. This global schema is made available to all system users in an identical form and managed centrally by a _global_ database administration. Access to the partial schemas always takes place via the global schema.

Above all, two aspects of this approach are criticized. Firstly, local autonomy of administration of the databases involved is given up, as their integration is managed centrally at least. The stronger the data of the systems involved are coupled, the smaller is their autonomy. When using global integrity conditions referring to several partial systems, runtime problems may arise. Secondly, *all* partial systems or all their used parts at least have to be integrated by this approach. The resulting total schema may quickly become unhandy. This approach can only be applied, if most of the partial systems have to be integrated once only and few subsequent schema updates are expected.

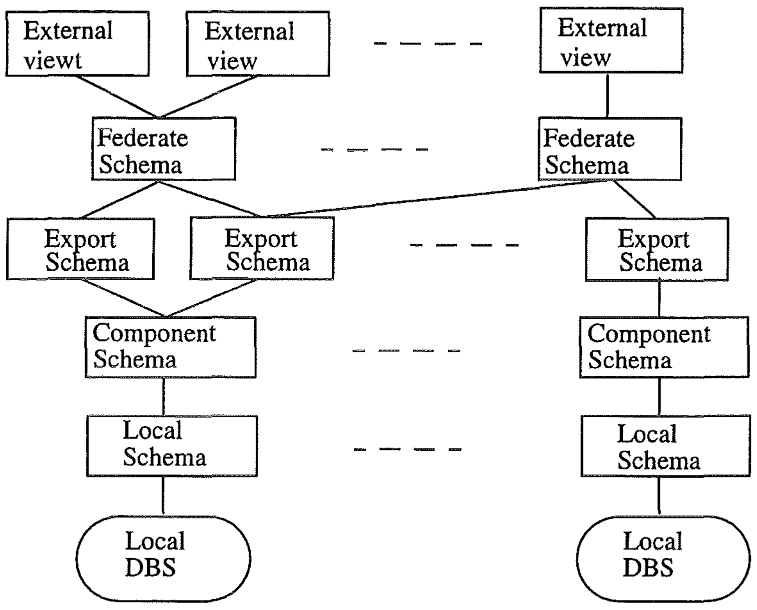
The advantages are as follows: once the integration has been performed, a common data view - a _large_ multiDBS so to speak - is made available to all participants. Integration is managed centrally. Thus, management overhead is reduced. Global integrity conditions are feasible with this approach exclusively. Here, runtime considerations have to decide, whether their use is practicable.

In the second approach, *several integrated schemas* are studied. They are also managed centrally. Individual user groups share one or several integrated schemas. Consequently, the basic advantage consists in the fact that those partial schemas, which are used jointly (in the respective group), have to be integrated in these schemas only. As a result, not all partial schemas of the entire system have to be integrated. However, strong integration is given up both in this and in the following approaches. In particular, it is no longer possible to use global integrity conditions, as variable integrated schemas allow variable views of the same data.

In the third approach, the so-called *federate schemas* (federate DBMS or shortly FDBMS) are investigated. This term was first mentioned by McLeod in 1985. Its difference from the previous approach is of organizational rather than of technical nature. In this approach, responsibility for the management of the integrating, i.e. federate, schemas is left to the individual user groups. Data for common use are made available by each partial system in an (exported) partial schema. Several partial schemas are integrated by a single federate schema. This explains the advantages and drawbacks of the second approach. An advantage is the large local autonomy of the groups involved.

The last approach studied is based on the so-called multidatabase system languages. In this case, there are no integrating schemas. Instead, each system user _knows_ that he is working with several partial database systems (no integration level). The language used for data access contains components, which explicitly allow access to a certain DB system. Possible common data use always has to be indicated explicitly. In these systems, integrity conditions can only exist or be managed locally, as there is no global management. Of course, autonomy of the partial systems is the largest of all architectures described. Since there is no integration level, the probably best runtime behaviour of all approaches has to be expected. However, expenditure for the user of the entire system, in this case the DB application developer, is by far the highest.

FIGURE 6 5--level schema architecture



To sum up, all approaches can be represented by the so-called 5-level schema architecture. From the bottom to the top, this architecture consists of the partial database system with a local and a partial schema in canonic form. The partial schemas are made available as export schemas and integrated by means of federate schemas. For the latter, external views can be defined and accessed by the applications.

3.1.2 Selection of an architecture

As outlined in the introduction to this chapter, a multidatabase system that is able to integrate heterogeneous relational database systems in wide networks is required for the use of RODOS. Major requirements made on the functionality of this distributed heterogeneous multidatabase system include a uniform transparent data view for all users and system functionality even under time-critical conditions. In addition, the principle option of managing global integrity conditions is desired.

Within the framework of RODOS planning, several organizational boundary conditions have been specified. Here, they shall only be covered briefly. They will be summed up in chapter 3.2.1. The major boundary condition is the existence of a central coordination system in the form of a global database administration (gDBA). Furthermore, the partial DB systems participating in

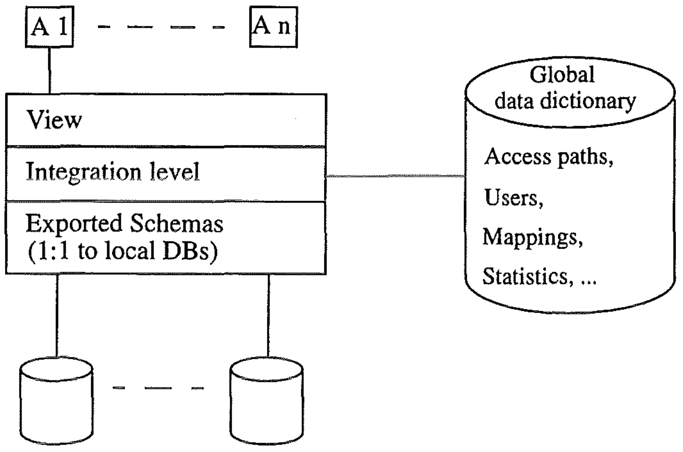
RODOS were specified to belong to the entire system either completely or partially. Relatively few schema updates are expected, once a partial DB system has been integrated.

It is obvious from the approaches of schema architecture above that the functionality required is ensured by the _global schema_ and _several integrating schemas_ approaches under the given boundary conditions. When insisting on the completely standardized, locally transparent view for the applications as well as on the principle option of managing global integrity conditions, the approach of the global schema only is left. In spite of the criticism stated, its use seems to be practicable, as there will be a centre for the integration of partial DB systems and no local autonomy of the systems is required. As few schema updates are expected, they can be managed centrally with an acceptable expenditure and transmitted to the partners with a certain delay. (In the meantime, the partners have to perform their schema updates on local copies of their databases, if applicable.)

To reduce the complexity of the global schema for the *users* of the entire system at least, the global schema is extended by the external views indicated in the 5-level schema architecture. (Thus, it is attempted to combine the advantages of a global schema and several integrating schemas and reduce their drawbacks.) The external views define that part of the global schema that is of interest to the respective user group. They are also managed centrally, as a result of which schema updates in particular can be carried out globally and simply transmitted to the partial DB systems by copying the relevant parts. Smaller updates can also be accomplished on the integration level of the global schema without the views of the user groups having to change. Complexity of global schema management for the gDB administration is maintained.

The question as to whether the functionality mentioned can be combined with the time-critical conditions cannot be settled within the framework of the present study. The respective main storage and (soft) real-time DB systems in distributed and, hence, possibly parallel environments are still subject of research. It may be doubted, whether sufficient scaling of RO-DIM can be achieved within the entire RODOS system. These and other questions will be answered by the RO-DIM prototype development. Other aspects in this connection shall be dealt with in chapter 3.8.

FIGURE 7 Schema architecture of RO-DIM



3.2 Schema integration and modification

In this chapter, the technical and organizational possibilities of implementing the RO-DIM schema architecture selected in the previous chapter shall be described. The organizational boundary conditions obtained from the meetings of the project partners shall be presented at first. Then, integration functions, the global data dictionary and schema updates shall be covered.

3.2.1 Organizational tasks and requirements

RODOS system architecture in the coordination centres and on a local level results in a multitude of tasks above all of organizational nature. In the previous chapter, organizational requirements made in the course of the discussions of the present study and influencing the selection of the RO-DIM schema architecture were pointed out. Now, they shall be summarized briefly:

- A global description shall be made available for the standardization of all data sources and sinks involved, i.e. all local partial schemas have to be integrated in the above-mentioned global schema.
- A very small group of persons, maybe one person only, will be responsible for global schema management as the global database administration (gDBA). Management of the global schema shall be performed centrally by this gDBA.
- The global schema is generated initially once. After this, updates are expected in large intervals (weeks or longer) only. They shall be carried out centrally by the gDBA.
- Integration of the local partial schemas in the global schema will be accomplished using general mechanisms for the description of mappings. Via these mappings, the global schema data

are accessed by the users. From there, access to the local schemas takes place. This means that the mappings have to be unambiguous in all directions. A number of these integration mappings shall be predefined. Not yet existing mappings will have to be extended by the gDBA, if applicable.

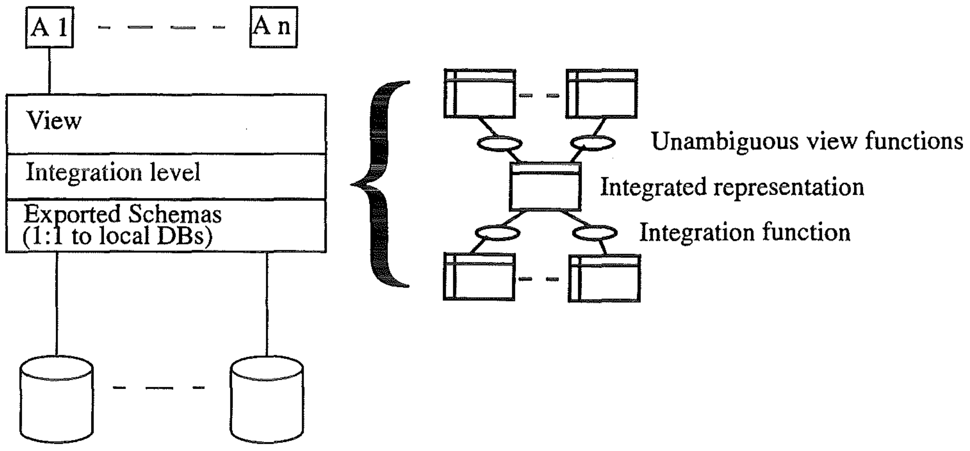
- To reduce the organizational expenditure (less persons involved), such mappings are allowed between local partial schemas and the global schema only. It may therefore be concluded that local RODOS databases are made available either completely or not at all. The description of local data structures shall no longer include additional (complicated) view mechanisms to keep the local integration expenditure as small as possible. It is deliberately accepted that this may lead to multiple local data maintenance, as the data involved in RODOS are no longer managed autonomously. Multiple data maintenance can be accomplished locally, for example by using functions of automatic data copying. A mechanism used for this purpose is the so-called database trigger.
- Data access in a global schema by applications is defined by external views. Data are to be accessed via these views only! The corresponding integration expenditure is accepted. Mappings from the global schema to the external view are possible.
- External views of the applications are managed as constituents of the global schema. In particular, these views are defined by the gDBA. This shall enable global user management and prevent _uncontrolled growth of the schema_. In the first step, user management shall be accomplished by a single access control to the system.
- Local schema updates are transmitted via the gDBA to the other systems involved with a certain delay. In order to be able to make use of local updates immediately, e.g. during programme development, the respective data have to be maintained twice. Thus, correct integration in the global schema is ensured in spite of updates of the local schema. At least access to these data is not impaired in an emergency.

3.2.2 Definition of mapping and integration components

For the integration of multidatabase systems, integration functions have to be made available. Using these functions, mapping between the exported local schemas referred to in chapter 3.1 and the entire global schema is carried out. Furthermore, these functions shall be defined in mappings between the global schema and the external views.

In general, these functions are mappings, by means of which any number n of source relations and their attributes are mapped to any number of target relations. Without limiting the functionality, a single target relation may be indicated only. In principle, any number of mappings may exist. (The latter version is probably easier to implement.) In RO-DIM, these functions are needed for the integration of partial schemas in the global schema and mapping from the global schema to external views.

FIGURE 8 Functions for integration



To enable the use of these functions by RO-DIM both in reading, i.e. _from the bottom to the top_, and writing, i.e. _from the top to the bottom_, they have to be unambiguous in both directions. Examples of such functions are given below. Unambiguity in both directions is also referred to as bijectivity. Instead of a bijective function, pairs of functions representing a bijective mapping may be used (one _reading_ and one _writing_ function each). The use of these functions shall now be explained by way of examples:

- In various partial DB systems, measured values may be represented in various units, e.g. a temperature in degree Celsius in Germany and degree Fahrenheit in England. By means of an integration function, this temperature could be represented uniformly in a global schema and, hence, be made comparable. In the general form, these functions can carry out conversions of value intervals.
- Integration of several attributes of various relations in a common target relation.
- Aggregation of values, e.g. in the form of summations. (Under certain circumstances, no unambiguity may be achieved in both directions such that the results of these functions can only be used in reading.)
- Renaming of a 1:1 mapping of a relation and its attributes in the global schema.
- Integration of redundant schema information, for instance in the form of several local relations describing identical situations. This is the schema redundancy referred to in the introduction.
- Filtration of data items, which exist several times, in reading accesses at least. This is an aspect of the data redundancy mentioned in the introduction.

There are various possibilities of defining and implementing these functions. Three of them shall now be presented briefly. The best founded approach would consist in the direct use of mathematical models [BLN86]. This approach has the advantage that statements with regard to the functions modelled can be proved mathematically (e.g. statements on the bijectivity of a function). However, mathematical models often are rather unhandy for practical use and therefore can be applied with certain limitations only. It also may be doubted, whether integration functions can be implemented efficiently on this basis.

Languages possessing a formal semantics and, hence, using e.g. calculi as a basis, go a step further. Examples are TROLL [HSJHK94] or F-Logic [FrLa93]. Statements on functions described by this language still can be proved (in most cases). They allow a more vivid description of facts, as they already possess accordingly powerful language constructs. The drawback of these approaches, however, consists in the fact that conversion into an efficient code sometimes requires a high expenditure and often has been achieved on the prototype scale only.

The third possibility is based on the direct use of _current_ programming languages, such as C/C++. Integration functions are coded directly as C/C++ programmes. By means of this solution, the probably most efficient results are obtained in terms of runtime, however, (practicable) proof of mathematical correctness of these functions is dropped.

A combination of the second and third approach, i.e. mapping of languages with formal semantics to efficient implementations e.g. in C/C++ is of particular interest. In the first prototype, mechanisms can be provided for C/C++ integration functions. They can be stored and made use of efficiently in a translated form. In a second step, mappings of a language with formal semantics to these functions could be made available. In the first prototype, the possibility of implementing such an approach could be investigated. As DTI and FZI shall design and make available a number of standardized integration functions, a language with formal semantics certainly would be an adequate means of description. Mapping to e.g. C/C++ can be carried out _manually_ in the first prototype. If both the data and the integration functions operating on them are considered, they can be regarded as objects. An object-oriented database management system, such as the OBST system developed by FZI [FZI93a], would be well suited for their management.

To support the designer of such integration functions, a toolkit, which is based on the integration language, may be considered. An integration editor may be developed, by means of which a survey of the relations existing in the system is given and the relations can be linked by _clicking_. Standard integration functions may be installed automatically. Other proposals in this connection shall be sketched in chapter 3.3.3 on redundancy.

3.2.3 Global data dictionary

For the management of global schema information or the RO-DIM structure and process information, a suitable management system shall be applied. It shall have the form of a global data dictionary (gDD), the contents of which are made available to all participants. The information to be managed results from the RO-DIM components, such as schema integration, distributed transaction management, access optimization, etc. Examples of such information are given below:

- *Access paths*: the _paths_ to the individual computers in the local and wide networks as well as to the databases of the entire RODOS system. Mechanisms are required for conflict-free naming tests. (Up to now, the name of a database has been specified to be unambiguous throughout the system. This allows the database to change to another computer.) Relation names have to be locally unambiguous and integrated unambiguously in the global schema. For this purpose, integration functions are provided. Path descriptions with alternative communication paths (deliberately redundant data maintenance) are planned for database servers.

- *Schemas*: here, the structures of global schema declarations and describing additional information, the local schemas of the individual partial databases and external views of the global schema are contained.
- *Integration functions*: it may be reasonable to combine the description of the integration functions with the description of the schemas and file it in the data dictionary as well. Thus, the schema structures and their integration functions can be combined to objects and, hence, represent object-oriented views of relational data. An example of this view is the use of an extended entity relationship data model in [HoK_o93].
- *Cost parameters*: parameters of scheduling, optimization and efficiency of computers and communication paths.

As the information mentioned probably will seldom be updated, but often accessed by reading, high-efficiency management is useful. For this purpose, use of an own DBMS, such as the OBST system [FZI93a] mentioned above, of the local DBMS existing in each local cluster or of an efficient file structure may be taken into consideration. Application of an X.500 directory for this task is described in [BCLM94]. CORBA mechanisms [OMG93], i.e. objects stored in a distributed manner, may also be considered. It may be doubted, however, whether sufficient gDD access efficiency can be achieved by CORBA. First statements made by the manufacturers with regard to the runtime [Iona93] are positive.

3.2.4 Schema modification

A problem of multidatabase system management consists in the (automatic) transmission of updates of the global schema or partial schemas of the databases involved, i.e. so-called schema evolution. The most flexible solution would be the automatic transmission of local schema updates to the global schema and all partners involved. This solution has the advantage of the partners being largely autonomous. However, implementation expenditure is very high. An example of such a system is Mariposa in [SADLO94]. The drawback of this solution consists in the fact that it is not certain, whether a schema information in the global schema actually describes a valid access to a local partial schema. Moreover, this solution takes much time. It is necessary to check constantly, whether schema updates are carried out or not. Furthermore, such updates may be contradictory and, hence, impossible to implement.

As a central administration of the global schema is planned for RODOS according to chapter 3.2.1, such a flexible solution will not be needed. It is planned to make all schema updates via the gDBA, i.e. the respective local DB administration informs the gDBA of its wishes for updates, e.g. by e-mail. There, they are integrated in the global schema. At times, which still have to be specified, new versions of the global schema will be distributed to the local systems. Possible schema conflicts will be settled by agreement. As far as runtime is concerned, simple tests are required only. The entire process is practicable organizationally. Once it has been integrated, relatively few updates of a schema are expected, as outlined in chapter 3.1. If local schema updates are required, copies of the data affected have to be used until updating of the global schema will be completed.

It is a problem of schema updates that it is not necessarily known, which applications are affected. The respective information may be managed in the global data dictionary. Thus, messages on these updates at least can be sent to the partners in charge. It should be checked, however, whether the expenditure is worth-while judging from the few updates expected. As the external views of the global schema shall be managed centrally, it is easier to adapt these views and inform the user group affected by it only. Many of these adaptations can be achieved by updates of the respective integration functions without changing the external views.

3.2.5 Redundancy

The problem of data and schema redundancy was explained in the introduction to this chapter. Schema redundancy refers to a variable description of the same information. It plays a role when integrating the partial schemas in the global schema. These redundancies are handled by the general RO-DIM schema architecture with the integration functions described in chapter 3.3. Data redundancy is understood to be the fact of the same data being stored various times at various places. This difficulty is encountered above all in queries and updates in distributed data files. Redundancies can only be handled after they have been recognized.

For the description of data redundancies, the so-called dependence graphs can be applied. Here, information is stored as to which databases or relations contain data, which also (i.e. redundantly) exist in other systems. When processing operations on the basis of these data, it is checked which other data are also affected and proceeded accordingly. Automatic use of dependence graphs represents the most complex solution for the handling of redundancies. However, an enormous implementation expenditure is required and significant losses in runtime may occur. An analog problem with comparable solution approaches is met in parallel database systems. The PANDA parallel DB management system developed for this purpose by FZI is presented in [Kram92]. As complete treatment of all aspects of data redundancy is very complex and difficult to implement, some partial solutions may be considered for the first prototype.

Filtering functions are taken over by the integration functions of chapter 3.3.

Partial results may be obtained for queries. Thus, the two most important cases, namely, transfer delays and complete system failures, can be handled satisfactorily. By the selection of certain databases, acceptable limitation of the set of results to the data desired is possible. See also description of the RO-DIM language in chapter 3.5.

- The mutual data dependences are described by a dependence graph. By means of a tool, manual management of this graph is ensured, i.e. it serves as an editor of data dependence descriptions. In a second step (maybe not in the first prototype already), a check functionality can be developed on the basis of the descriptions. It is used to indicate conflicts, which may then be corrected. Consequently, inconsistent data, which may be handled by the RODOS applications, are accepted sometimes (see also chapter 3.5).

The following amendments are considered, but probably not for the first prototype already:

two tools for the recognition of data and schema redundancies on the basis of data values and schema descriptions (names of databases, relations and attributes). For this purpose, storage of additional information in the global data dictionary, i.e. a type of common thesaurus may be required. The prerequisite is uniform naming based on a *single* (natural) language at least. This may give rise to proposals regarding the handling of redundancies e.g. by appropriate integration functions.

- Use of the dependence graph in query processing. See e.g. [RaSc94]. Suitable are local triggers signalling global conflicts. Correction may then take place semi-automatically. (Automatically, as long as this can be done in a conflict-free manner.)

3.3 DBMS integration

Actual access to the heterogeneous relational database systems represents the lowest level of RODIM architecture. On this level, the concrete DB management systems (Oracle, Ingres, ...) are mapped to a canonic data model (s. chapter 3.1.1) or a standard interface is made available in the form of access functions. Even if the special properties of concrete DBMS mostly are not made use of by such a standardization level, as a result of which performance losses may occur, the use of such a level is reasonable. Without it, special mappings would have to exist for each DBMS on the integration level. This solution is difficult to implement and difficult to use by the gDBA. As a consequence, this last approach should not be applied in our case.

This leaves us with the first approach based on a driver for each DBMS. Here, it seems to be useful to apply already existing solutions, e.g. class or function libraries, which support the access to various DBMS and possibly have already been equipped with communication components for the access of the local and wide networks planned for RODOS. As it is not clear with which DB management systems RODOS will work and how already selected systems will further develop, commercial libraries should be applied. Here, it can be assumed that the adaptation to new DBMS versions will be performed by their manufacturer.

If the driver component has to be developed, a set of access functions required has to be defined in a first step, such as Dynamic-SQL [MeSi93], ODBC by Microsoft [Miso93] or others. These access functions have to orient themselves according to the language to be developed for RODIM (cf. chapter 3.5), i.e. at least contain functions for data definition and manipulation, lockings and transaction management. Furthermore, active components (e.g. triggers) and mechanisms for the termination of transactions are desired. The latter will be used for asynchronous termination of transactions. Performance can only be described in further detail after having reached an agreement on the total scope of the RO-DIM language or the RO-DIM prototype.

Implementation of the driver components can be defined e.g. as C-function library, a partial set of Dynamic-SQL or by object-oriented views of relations. Approaches are e.g. C++ class libraries describing relations as classes with access methods (see [Heue92, HoK_o93]). Another approach would consist in the generation of shell scripts containing SQL instructions. Most commercial relational DBMS offer a text-based interface, which may be used for this purpose. Files or semaphors for synchronization can be applied to determine the status of an instruction in execution.

3.4 Database language (DDL/DML)

For application development, RO-DIM shall make available an interface in the form of an own data definition and data manipulation language (DDL and DML). Data manipulation is understood to be the access to data, i.e. mostly query, update and delete operations. In addition, instructions for the control of such operations are included. The data definition language above all contains instructions for the generation of data descriptions. These are instructions for the generation of relations.

The language to be designed shall support the functionality of a partial scope of ANSI-SQL 92, Level 1-2, also called SQL-2, which has not yet been specified. For the description of SQL-2, see also [MeSi93]. Furthermore, the syntax of the language shall be in line with SQL. This, however, is no requirement. An object-oriented representation would also be acceptable.

In the following chapters, a survey of the functionality to be covered by the language shall be given as it is known up to now. Here, description will follow SQL syntax. As concrete requirements have not yet been made, it is often rather vague. Especially the use of software tools (s. chapter 7) can and will influence the amount of language means or underlying functionality that can be made available with a reasonable expenditure.

3.4.1 Interface

As indicated in the previous chapter, data exchange between RO-DIM and RODOS applications shall take place via the general RODOS message server, which is working asynchronously. To allow an adequate integration of RO-DIM in the entire RODOS concept, DTI arranged for the data exchange between RO-DIM and the message server being asynchronous as well. Thus, it is ensured that each component of the RODOS system can work as independently as possible. Embedding of the language into the applications e.g. by pre-compiler mechanisms is not desired at all. A set of DB operations, which is limited by system capacity exclusively, shall be processed in parallel (in transactions, see chapter 3.5.6). It is planned to use a simple communication protocol that may be amended by further protocols, if necessary.

In principle, a message to RO-DIM will consist of a DB operation, e.g. a chain of characters containing an SQL order, and a reference to a communication and a data buffer. The communication protocol is based on polling, i.e. a DB operation is started by the application. In the communication buffer the status of _in execution_ is set by RO-DIM. The operation is executed and after its completion the status _completed successfully or not successfully_ is signalled. Consequently, the application has to check (polling), whether the operation has already been completed.

In addition to this simple process, an application may also possess functions, which may directly respond to messages of the RO-DIM using adequate operating system mechanisms. Further communication protocols may be added, if necessary. Within the framework of the RODOS message server, similar functions shall be implemented at the moment. It is reasonable to transfer them to RO-DIM later on.

3.4.2 Additional parameters for database operations

When starting a DB operation O_{DB}, a set of options can be defined. These options influence both the effect of and the amount of data processed by the DB operation. Up to now, the following fields have been specified:

- *Selection of the data sources*: it is indicated, whether O_{DB} shall be carried out on the basis of a precisely specified set of databases. If no such set is given, all databases of the system are taken into account. In case of queries, for example, the answer is searched for in the entire system. For queries, this difference is of particular importance, as the sets of (partial) results may vary accordingly.
- *Schema use*: it shall be specified, whether processing shall take place directly on the basis of local database schemas in the system. This option mainly aims at increasing the performance. If it is not used, the global schema is applied. Responsibility for the use of this option will have to be borne by the respective developer, as e.g. (global) integrity conditions (cf. chapter 3.5.7) may be by-passed. It should be checked, whether such an option is actually needed or exclusive accesses via the global schema are sufficient instead. For the first prototype at least, accesses via the global schema exclusively were specified (see also chapter 3.2.1).
- *Termination criteria*: when processing DB operations, delays may occur for various reasons. As a first criterion, maximum processing time shall be indicated. In addition, proceeding after this time shall be stated. The O_{DB} is either terminated immediately or a message is written into the communication buffer and the O_{DB} is continued. This message can be evaluated by the application and the O_{DB} can be terminated explicitly, if applicable. The latter is accomplished by terminating the respective transaction. In case of a termination of queries for reasons of time, it can be indicated, whether the supply of partial results is desired. In this case, the data that have been collected in the communication buffer up to termination are supplied as the result and the respective status is signalled to the application. (This termination behaviour

is supported by many relational DBMS. If this is not the case, it has to be simulated by global termination of the O_{DB}, even if it is further processed by the local system.) This case is in analogy with the treatment of partial results in the *selection of data sources*. Other termination criteria or termination actions are conceivable, in particular the priorities mentioned below.

- *Priorities*: in order to rapidly execute extremely _important_ O_{DB}, they shall be given priorities in a first step. These and other measures shall be evaluated by a scheduling component (see chapter 3.8). Other measures, such as the type of tool, the type of user, etc., are subject of discussion.

Specification of the parameters mentioned may take place in various forms. Amendment of the RO-DIM language is suitable. A set of keyword/parameter pairs may also be used. They may be transmitted in the communication buffer. Another possibility is a special area, e.g. a communication relation, in global RO-DIM data management.

3.4.3 Data definition

RO-DIM shall contain the basic options for the definition of data structures in the underlying databases. This applies to the creation and deletion of new relations or their attributes. The attributes are based on conventional SQL data types, such as Char, Number, etc. There may be limitations in the national set characters and the date/time data types. This corresponds to the following SQL commands:

```
CREATE TABLE relation

COLUMNS (attr. 1 data type 1, ..., attr. n data type n)

as well as DROP TABLE and ALTER TABLE or ALTER COLUMN.
```

These operations shall work on local databases only, in the first prototype at least. Therefore, exactly one local database has to be indicated. The schemas created there will then be taken over completely by the global schema. If necessary, they have to be provided manually with other integration functions.

Database creation as such takes place locally. They will be available upon integration in the global schema only. In a local database, relations have to bear unambiguous names.

Operations aimed at increasing the performance, such as the creation of indexes, clusters, etc., do not have to be part of the RO-DIM language. Such operations can be carried out on the respective local systems for the time being. In another prototype, global operations of this type could be provided, for instance for the generation of global index structures.

3.4.4 Queries

The possibilities of database queries still have to be specified in detail. In a first step, it is proposed to orient according to a restricted form of the SQL-SELECT instruction. This restricted form has two major advantages. On the one hand, it can be implemented with an acceptable expenditure for the first prototype at least. On the other hand, processing expenditure of such a query can be estimated more easily than that of advanced forms. This estimation can be evaluated later on in the RO-DIM scheduling component for performance increase and query priorization (see chapter 3.8). A drawback of this simple form of queries, however, consists in the fact that complex queries have to be made in several steps under certain circumstances. It is planned to orient according to the general form of SELECT:

```
SELECT attribute 1, ..., attribute n

FROM   relation 1, ..., relation n

WHERE  search condition.
```

In this case, the search condition shall only contain the usual comparison predicates, i.e. =, <>,_, etc. as well as the logical operations AND, OR, NOT. LIKE, IS NULL, ORDER BY and function references are added. The latter shall be used for standard operations, such as the formation of partial character chains, as long as they are based on individual attributes only. In contrast to this, set comparisons, such as >=ANY, ALL, EXISTS, IN and sub-queries shall not be applied, nor shall GROUP BY and HAVING. The operation DISTINCT (for duplicate elimination) will be available, but only for the elimination of identical result tuples (lines of the result).

Queries referring to several relations shall be possible in the form of so-called simple joins. This form expresses explicitly that several relations shall be worked with. In addition, the implicit form is generated by summarizing relations of various local database systems in a global schema using integration functions (see chapter 3.3). A query referring to a relation of the global schema may consequently affect implicitly several relations of the local databases.

3.4.5 Inserting, updating, deleting

These operations directly orient themselves according to the respective SQL instructions in [MeSi93]. Hence,

```
INSERT INTO relation

VALUES (value 1, ..., value n)
```

inserts a tuple into a relation.

```
UPDATErelation

SETattr. 1 = expression 1, ..., attr. n = expression n
```

```
WHERE<search condition in analogy to chapter 3.5.4>
```

updates attributes in tuples of a relation, to which the WHERE condition applies.

```
DELETE

FROMrelation

WHERE<search condition in analogy to chapter 3.5.4>
```

deletes all tuples of this relation, to which the WHERE condition applies.

It must be noted that the search condition (at first) may operate explicitly on one relation exclusively. A search condition may be carried out implicitly over several relations using the integration functions for the global schema. For this purpose, the integration functions have to work _satisfactorily_.

3.4.6 Transactions

To ensure that simultaneous, concurrent database accesses by several users (or processes in general) do not affect each other, the transaction concept has been developed. A transaction consists of a series of DB operations that belong together and classically comply with the so-called ACID properties. ACID is explained as follows:

- *Atomicity:* either all DB operations or none of them are carried out within the transaction. The transaction acts like an atomic DB operation.
- *Consistency:* by the transaction, the database is transferred from a consistent state to a consistent state.
- *Isolation:* concurrent transactions, i.e. transactions operating on the same data, do not affect and, hence, are isolated from each other.
- *Durability:* all data updates by a (successful) transaction are permanent (durable), even in case of hardware or software failures.

At the moment, two principle transaction modes are planned for RODOS. In the first form, the transactions, as presented above, correspond to series of DB operations that belong together. Such transactions are started explicitly by a BEGIN TRANSACTION command and terminated explicitly by COMMIT or ROLLBACK TRANSACTION. In a transaction, no other transaction may be started - at the moment. Using COMMIT, all data updates of the transaction become permanent, provided that no failures occur. By ROLLBACK the transaction is terminated and the databases are transferred back to the state prior to the start of the transaction. In a transaction, instructions for data definition may not be mixed with instructions for data manipulation. As an alternative, each DB operation may implicitly represent a transaction. Thus, old existing programme packages in particular are enabled to make a simple DB access. In such cases, the application itself is responsible for maintaining data consistency. As the old applications at least have been developed without any transaction mechanisms, this will be practicable in many cases.

Performance may be further increased by the transactions partially dispensing with the *isolation* property. Here, several cases are distinguished. Only one is considered for RO-DIM for the time being. It is known as the dirty-read phenomenon [Date90]. It is assumed that a transaction T1 performs a data update and a transaction T2 uses the updated data before these values have been declared valid by T1 using COMMIT. Thus, T2 can be carried out more rapidly. For a certain reason, however, T1 stops by a ROLLBACK such that T2 works with invalid data. This may not be desired in practice, but is of interest to RODOS above all in the emergency phase. In this phase, many data insertion operations will occur especially in the field of measured values. For the handling of these real-time transactions, approaches have been developed e.g. in [AbGa92]. The tools that are supposed to decide on the basis of these data, however, must be able to work with partially incomplete or incorrect data. In this case, permits for dirty reads are reasonable and may lead to a considerable increase in system performance. In SQL-92, two other cases are distinguished. They are dealt with e.g. in [MeSi93].

3.4.7 Integrity conditions

Within RO-DIM it shall be possible to specify so-called integrity conditions. In general, they serve to ensure certain data properties particularly in case of data dependences. In principle this can be done in two ways. It can either be ensured by an application that the data used by it fulfil certain properties or this may be done directly by the DBMS. As the data, which are to be managed by RO-DIM, can generally be used by various applications, the system itself shall be able to manage (global) integrity conditions and their observation. A detailed discussion of this subject would exceed the scope of this chapter such that a few examples only shall be given.

In SQL-92, three basic possibilities of indicating integrity conditions are available. The first is the CHECK instruction giving the conditions for attribute values in relations. Its syntax is simple:

```
CHECK (search condition)
```

In this case, the search condition shall correspond to a query condition according to chapter 3.5.4. Attribute values are checked for the observance of value ranges and the being contained in static value lists or value lists of other relations. While CHECK is planned for conditions in a relation, the ASSERTION instruction allows to specify conditions for attributes that are contained identically in several relations.

The third major form of integrity conditions concerns the so-called referential integrity. In this connection, the terms of primary key and foreign key are important. A primary key of a relation is a combination of attributes, whose value combination in the relation is unambiguous at any time. This means that there are never two tuples having the same primary key in a relation. If several attribute combinations, which have the properties of a primary key, exist, they are referred to as candidate keys. It shall be possible to specify one or several of these primary or candidate keys for a relation.

To indicate direct dependences (references) among relations, so-called foreign keys are employed. A foreign key of a relation R1 consists of values corresponding directly to values of a relation R2. This shall be illustrated by an example:

```
Referenced relation: CUSTOMERReferencing relation: ORDER

Primary keyForeign key customer name

Customer name: K1K1, article 1


K1, article 2
```

It remains to be stated what is to be done, if the integrity conditions are violated. The easiest way is to reject a DB operation leading to a violation or to terminate the entire transaction. In the first RO-DIM prototype at least, this should be sufficient.

In principle, integrity conditions are very useful, but they have two major drawbacks. Firstly, a high expenditure is required for the implementation of their check. Secondly, their use may lead to a significant reduction of system performance, as they have to be checked after each DB operation in the case of doubt. (Attempts have been made to improve this behaviour.)

Therefore, it should be considered to switch off the integrity conditions in the emergency phase at least. Generally, it has to be verified, whether and to which extent integrity conditions are to be implemented - in the first prototype at least. At the moment, they are made available within the local database systems only. Hence, global integrity conditions do not exist. However, mechanisms, which possibly exist in the software tools to be applied, may also be used.

3.4.8 Events, triggers

To make available the integrity conditions mentioned in chapter 3.5.7, another way of indicating a general form of events is possible. In accordance with the future versions of SQL, the use of the so-called triggers is considered. Their syntax and semantics have not yet been standardized, however, comprehensive proposals have already been made. A trigger refers to a data manipulation, which is carried out in a relation of the form:

```
TRIGGERtrigger name  time  event

ONrelation

WHENsearch condition [actions]
```

Each time of an event of the form INSERT, UPDATE or DELETE occurring in a relation, the trigger _fires_. The time is used to indicate, whether the trigger is to be called prior to or after the event. By the search condition (see chapter 3.5.4), the trigger can additionally be limited to tuples fulfilling this condition.

The actions to be performed during the _firing_ of the trigger are restricted to the instructions DELETE and UPDATE. For RODOS, an extension in the form of messages to applications or references to procedures stored in the DBMS may be considered.

In analogy to the integrity conditions, the use of triggers may lead to considerable runtime losses. Consequently, limited use only is reasonable. Switching off of the triggers in the emergency phase at least should be considered. As far as the implementation expenditure is concerned, it also has to be assessed, whether and to which extent triggers are to be realized - in the first prototype at least. At the moment, use of the possibilities provided within the local database systems only is planned.

A general problem of global integrity conditions and triggers is the autonomy of the local database systems involved. It is always possible to perform local updates that are not noticed globally. In our case, this problem is covered by the organizational requirements (see chapter 3.2.1).

3.5 Transactions

3.5.1 Transaction concept

In database management systems supporting multi-user operation, accesses of the users have to be synchronized such that the data are always in a consistent state and the users do not disturb each other. For this purpose, the transaction concept is applied.

To observe the consistency of the common data and prevent the quality of the decisions taken with RODOS from being endangered, it is absolutely necessary to permit database operations within the framework of transactions exclusively. In RODOS, however, old application programmes are used. They are prepared for manipulations of files only and do not support any transactions. These programmes can still be used in the future. In RODOS, a new transaction will be started automatically for each database operation. Integrated data management of RODOS has to meet the requirement of nesting transactions.

3.5.2 Distributed transactions

As already mentioned, RODOS belongs to the class of distributed information systems. The individual databases are managed at several places using heterogeneous data management systems. In certain situations, if for example a rescue operation is initiated simultaneously at several places, distributed transactions are required, by means of which several databases are partially accessed in writing. During these transactions, in which heterogeneous data management systems may be involved, RODOS has to ensure that the transactions will be completed properly in all participating databases.

When implementing distributed transactions, the standard _two-phase commit_ (2PC) procedure is applied. This procedure was described in a number of database manuals, e.g. in [Date91]. It can be implemented, if one of the data management systems provides an adequate support (or to be precise, the prepare to commit operation). Most commercial relational data management systems already contain this operation. As the 2PC procedure is employed widely on the one hand and can be implemented easily on the other, this procedure is selected for the realization of distributed transactions in RODOS.

3.5.3 Treatment of deadlocks

A deadlock is generated, if two (or several) transactions block each other by locking the same tuples. In a complex distributed architecture like RODOS, the recognition of deadlocks represents a non-trivial and time-consuming problem. For the first prototype we would like to suggest the time limit procedure. A time limit is set for each transaction or operation. If this operation (transaction) is not completed within the time limit specified, occurrence of a deadlock may be suspected. In such a situation, the operation is terminated (rollback in transactions) and an error message is sent.

This method also allows the processing of soft real-time queries in RODOS.

3.5.4 Management of locks

By setting locks, concurrent transactions are prevented from influencing each other. Locks can either be set implicitly by the database management system or RO-DIM or explicitly by the application programme itself.

In addition to the traditional locks used in database systems, it is useful to introduce new types of locks into RO-DIM:

- _Dirty read_ lock - it allows the reading of data that may be modified simultaneously by other programmes. No guarantee is assumed for data consistency.
- Insertion lock - it is only used for the insertion of new data (no modifications of existing data). All reading accesses are allowed.
- _Administration lock_ - this lock may always be requested by an administration programme; it causes the termination of all transactions, which originally locked the tuples affected.

3.6 Events

As already mentioned, a notification mechanism is required in RODOS. In case one of the predefined events occurs, the respective action is initiated.

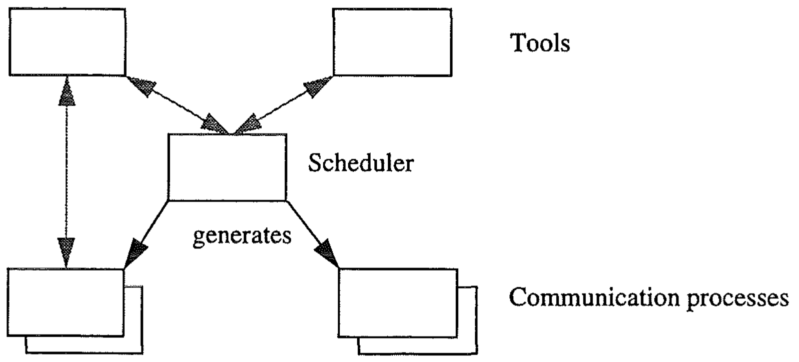
Possible actions include among others the starting of a certain programme on any computer in the network by means of the remote-procedure-call mechanism, notification and/or reactivation of a programme by a signal as well as the sending of an e-mail to the users.

The RODOS notification mechanism has two advantages, by means of which system performance is increased. Firstly, the data consumers are notified immediately after the occurrence of an event without losing time. Moreover, the data consumers can wait in a suspended[1] manner without making queries to the database system in the meantime.

3.7 Parallelization and distribution

The requirements of high system performance and observance of time limits can only be met by a multiple process architecture. In Fig. 9, the process architecture implemented on each workstation is represented.

FIGURE 9 RO-DIM process architecture



As before, the application programmes (tools) are started as separate processes. To start communication with RO-DIM, they send a message to the scheduler. The scheduler generates a new communication process. The task of the communication process consists in the processing of the database operations called by the tool. Each communication process is responsible for communication with one application programme and one database system precisely. If a query has to be processed by several database systems, a clone is created by the communication process (using the UNIX command fork). This architecture allows to use the waiting times for communication with other computers or I/O communication for other processes. Communication processes are terminated upon the completion of a transaction.

[1]without using the CPU

The number of active communication processes is controlled by the scheduler. If this number is too high, performance may be decreased. Thus, a tool may have to wait until a communication process is allotted to it. Scaling of this architecture is possible. If a computer possesses several processors, the number of processes that are active at the same time is accordingly high.

The priorities of the communication processes are also controlled by the scheduler. The priorities are given to transactions (including sub-transactions) or DB operations for tools using no transactions. The algorithm for the giving of priorities may be specified by the DBA. It is controlled by the following parameters:

- User recognition - deciding authorities are privileged, the priorities may be allotted both to individual users and user groups;
- Name of the application programme - statistics programmes can work with lower priorities;
- Priority desired by the programme;
- Other parameters are feasible.

The strategy can distinguish between normal and emergency operation. In an emergency, some programmes may be excluded.

To increase the reliability of this architecture, an additional daemon process is planned. This process is activated cyclically and checks, whether the scheduler still exists. If this is not the case, a new scheduler is generated.

3.8 Other DBMS-typical functions

3.8.1 Data protection and data integrity

In particular the distribution of the systems involved in RODOS results in a general problem of data integrity in data networks. It has to be specified to what an extent data protection is required, which methods have to be used for e.g. data coding and which RODOS component is responsible for which field. Coding could be achieved e.g. by a database management system, the communication component of the operating system or by hardware measures. Except for a simple user management, no special protection measures are planned for the first RO-DIM prototype.

3.8.2 User management

Due to the requirements of the decision process in RODOS, several user categories have to be introduced in RO-DIM:

- Deciding authorities
  In an emergency, deciding authorities shall be served especially quickly. They mainly perform reading operations.
- Global database administrator
  The global database administrator has all rights - in particular the global schema can be changed by him.
- Local database administrator
  The local database administrator can perform all administration processes on the local computer. When logging in on a foreign computer, the local database administrator is treated like a normal user.
- _Normal_ users
  This group of users is entitled to perform all allowed operations in the normal case. In an emergency, these users are served with a lower priority. High-expenditure operations are not allowed in the latter case.

For each database in RODOS, a list of users (user groups) with their rights (reading, inserting, modifying, deleting, schema modifications) is available.

3.8.3 Fail-safeness

In the first prototype at least it is assumed that the hardware and system software components work reliably. Time-out mechanisms only are planned for failures of partial systems due to network overload. Furthermore, explicit re-start of a local node shall be possible. All transactions are set back in these cases.

3.9 Internal RO-DIM components

To sum up the previous chapters, a first survey of the planned RO-DIM structure shall be given. A rough sketch of the architecture is shown.

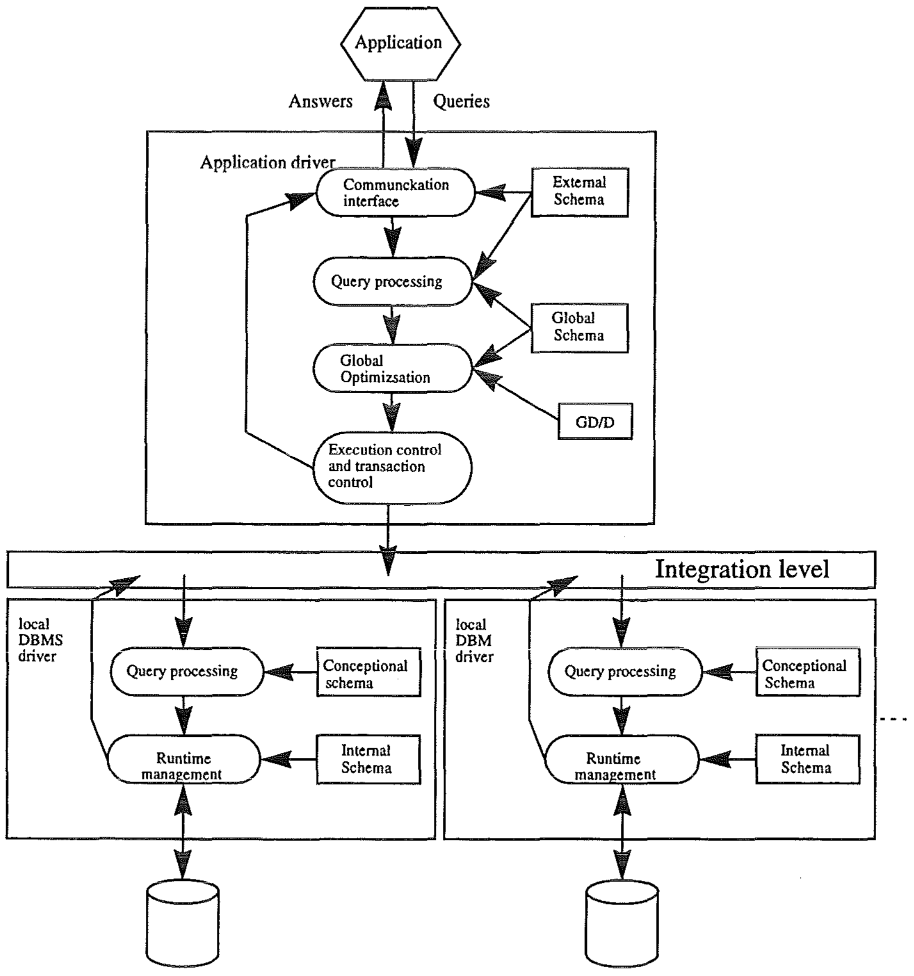
In the figure, the most important functions of RO-DIM are indicated according to the previous chapters. These are:

- Message-oriented interface for communication and data transfer from and to the applications.
- Interpretation of the RO-DIM language (DML/DDL).
- Query optimization on the basis of the performance parameters.
- Global runtime control with distributed transaction management, query scheduling, integrity control, scheduling, etc.
- Communication with other instances of the global schema.
- Specific or generic drivers for all local relational DBMS with query processing, data buffering and communication.

In addition, user management and control as well as general components are applied:

- Timer mechanisms, e.g. for the termination in case of deadlock recognition.
- Meta data and buffers for asynchronous message distribution.
- Data management for automatic re-start after system failures.
- Failure control of partial services (e.g. a daemon per workstation or user). Notifications to applications in case of failures and termination of the respective DB operations.
- Establishing of communication among the nodes via WAN mechanisms.
-

FIGURE 10 Rough RO-DIM system architecture



Chapter 4

Knowledge management

4.1 Introduction

In the present chapter, the structure and implementation of a knowledge base for RODOS shall be described. Distributed data are accessed by the system. Each authority has its own knowledge base such that it can make its decisions locally. The knowledge base may differ from authority to authority. It probably will be implemented in INGRES.

In Tab. 1, some examples of possible events and responses derived by means of the knowledge base are presented.

By analysis of this event-response relationship, the following findings are obtained:

TABLE 1 Possible events and responses

ozone level exceeds the threshhold	Prohibit traffic
Crocodile in lake	Prohibit access to lake
Waterline exceeds limit	Construct a dam
Fire in a factory	Warn neighbourhood
Fire in nuclear plant	Evacuate people
Explosion of a nuclear power plant in Russia	Warn the population

- This relationship is based on very simple _if, then_ rules. The rules do not contain any negations, i.e., they are neither _if, then not_ rules, nor _if not, then_ rules.
- No evidences occur in the rules. If such an event happens, it must not be thought about how dangerous it is, but all precautionary measures that can be taken have to be taken.
- The current values only count. This is a form of unmonotonous conclusion. In case new values occur, the old conclusions may be forgotten. There are, however, some cases where also pre-

vious values have to be considered:

- this, for instance, applies to the response in case of the water level of rivers being too high. Here, it is of interest, whether this level is increasing or decreasing. Having an increasing level, auxiliary dams have to be built. With a decreasing water level, this is not required. For this purpose, the water level of an hour ago must be known. Therefore, temporary aspects have to be introduced into the database. If this is not done, the water levels of upstream river sections could be evaluated instead. However, this would be very complicated. It is therefore assumed that the information required is supplied as a differential, e.g. in the form of _increasing water level_.
- If a measure taken in a similar case is known, it usually can also be applied in this case.

4.2 Structure of the knowledge base

The above observations suggest to implement knowledge management as a rule-based system in **Horn logic**[1]. An _IF - THEN_ structure is realized most rapidly and easily by Horn clauses. Using variables, rules can be selected such that they fit in similar cases.

4.2.1 General structure of a rule-based system

The general structure of a rule-based system consists of rules and facts. A fact has the form:

$p\ (x_1,\ x_2,\ \ldots,\ x_n).$

It describes a general context. In this context, it is stated that all x_i fulfil the predicate p. Instead of the general variables, also instances of them might have been applied, which would have softened the statement. The variables assume the values of a countable basic set. This is the so-called Herbrand universe, which represents the set of all basic instances. The Herbrand universe is generated from a finite set of symbols. This assumption has to be made, as (_+1) sets or infinite sets of symbols cannot be represented by the computer. The set of facts represents the _basic knowledge_ of a rule-based system. The task now consists in deriving from this set of facts additional facts that have not been listed explicitly in the system. This is done by means of rules having the following form:

$A_1,\ A_2,\ \ldots,\ A_n \text{ --> } B$

They represent a 'modus ponens': if A_1, A_2, ..., A_n are fulfilled, B can be derived. All variables existing in these formulas can be considered quantified by all-quantors.

[1] Rules of the Horn logic are a special type of general rules; they do not contain any negations, i.e. it is attempted to derive _positive_ knowledge only.

The set of all statements derived can then be extended by B. The set of all facts that can be derived can be referred to as the significance of the system. The significance of the Horn logic above all may be attributed to the fact that fulfillability can be decided upon for a set of Horn formulas in polynomial time. In contrast to this, fulfillability of logical formulas can be decided upon in exponential time only. This fact may be made use of to determine inconsistencies in the rule base.

For the mechanism of derivation, two procedures have been developed:

1. Starting from a given objective, the attempt is made to verify, whether it can be derived. If, for instance, B is given as the objective, it is tried to find out in the next step, whether A_i can be derived. This mechanism is referred to as **reverse chaining**. This procedure is used above all for plan preparation and construction when a given objective exists and the path of how to reach this objective is looked for.
2. Another procedure, which is referred to as **forward chaining**, uses the set of all given facts as the starting-point. Here, the following algorithm is applied:

```
F:= set of all given facts
REPEAT
F':= {B: B _ $A_1$, ..., $A_n$ is in the rule base
and $A_i$ is in F};
F':= F+F';
UNTIL F' _ F
```

It must be noted that all F generated in the course of the algorithm are finite and that all derivable facts are actually derived, as the set of all derivable facts and the basic set are countable. The algorithm, however, does not need to terminate.

Forward chaining is of particular interest, if the initial facts are known, while the objective is unknown and shall be determined.

4.2.2 Rule base of RODOS

Each problem has a special structure. The structure of the rule base has to be selected accordingly. Given the conceptions of the first chapter, forward chaining only may be applied in this case.

The forward chaining algorithm is given in 4.2.1. In this form, however, the algorithm cannot be used. This is due to the fact that there are numerous rules and given facts. Furthermore, most of these rules and facts have nothing to do with a given case. Verification, whether a rule can be applied to a given fact, takes time. In our case, this cost factor is enhanced by the facts existing in the knowledge base not in a local, but a regionally distributed manner. Access via the network takes extra time. Consequently, the algorithm is unsuitable from the time expenditure point of view already.

The major task therefore consists in structurizing the rule base. Rules, the so-called meta rules, have to be installed into the knowledge database. Thus, the number of rules accessed in the course of derivation is limited automatically. This limitation may be fixed or even further limited by the user in an interactive manner.

Moreover, the existing facts have to be structurized. There must be a given set of facts, indicating whether or what kind of catastrophe has happened (e.g., alarm(x)). Usually, the case of occurrence of a catastrophe is an exceptional case and it would not make sense to start the entire derivation process without any reason. In the normal state, the derivation process may either rest or restrict to this set of facts only.

The next important aspect is the acceleration of the derivation process. Non-determinism of this process implies the selection of correct rules and the determination of the facts to which they are applied. If the strategy presented in the basic algorithm is used (wide-range search), exponentially much time may be spent until the right measure is found. Therefore, a selection rule indicating which rules are to be applied has to exist. This rule has to ensure that the derivation process always terminates, even if certain rules are excluded from the derivation process. Moreover, the derivation process shall supply the response most rapidly in those cases where it is needed most rapidly.

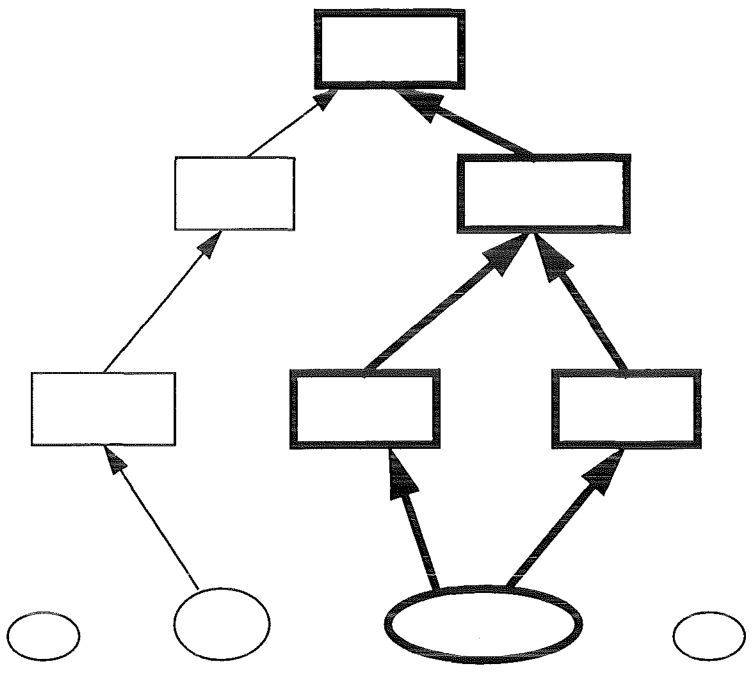
For this purpose, the strategy of the strongest pre-condition is suitable. According to this strategy, the rule with the strongest pre-condition shall always be selected first. For example, the rule B<-A, C, D has a stronger pre-condition than B<-A, D. Consequently, the first rule is selected. However, the premises of both rules may have different predicates with those of one rule being more known than those of the other. In any case, a list of priorities of rules has to be managed. This list may have been specified or vary depending on the type of partial problem. The same applies to the facts used. Here, a list of priorities may also be reasonable.

Measures of the highest priority shall be derived first. As such measures have to be output most rapidly by the system, the behaviour desired in the previous chapter is generated. Furthermore, this rule is helpful, as it always has a certain meaning, if a rule of high priority can be applied. In the decision process for the handling of catastrophes, a fact derived by such a rule hardly stays alone and will not be used any longer. In most cases, it will have a certain effect. Therefore, no time was lost. As decision processes of the authorities have a hierarchical structure, the rules can mostly be arranged such that a rule that cannot be applied will also be considered not applicable in the course of the further derivation process.

If no measures have been found, it shall be tried to derive measures from similar cases.

The set of all rules may consequently be arranged into a hierarchical schema. The following diagram is obtained:

FIGURE 11 Rule hierarchy



The ellipses represent disjunct sets of facts (observations). The bold ellipse is a set of facts currently needed for the actual problem. The rectangles represent synonymous disjunct sets (**clusters**) of rules with the bold rectangles denoting the rules needed. To apply rules of higher clusters, a certain number of rules from the clusters below must have fired. With increased priority it can then be tried to make rules of the top cluster fire.

The clusters are used to determine **meta rules** for the resolution of conflicts. They make use of **states** of the derivation process. A state can be defined as follows:

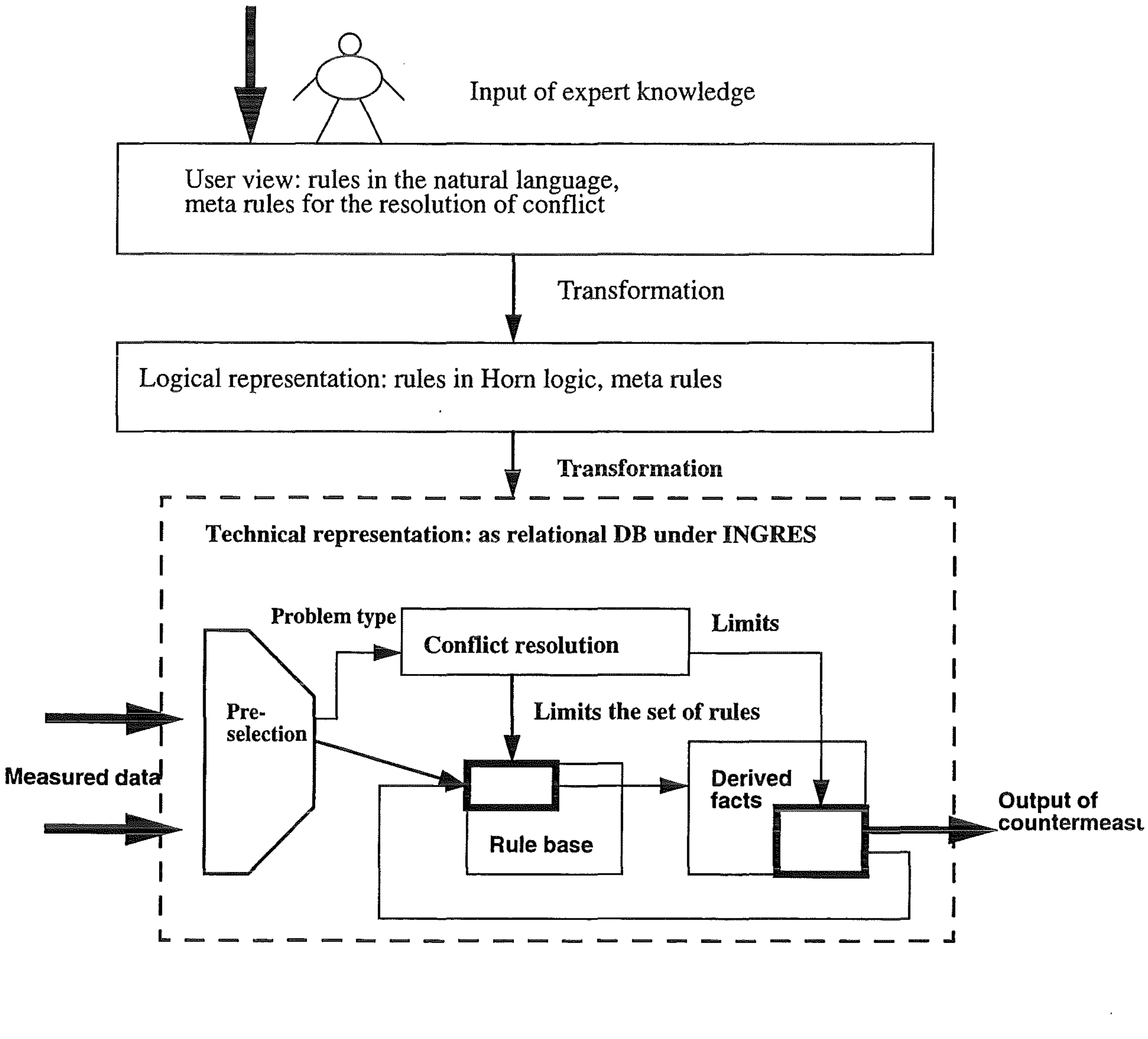
```
Definition: let K be the set of all clusters. Then, a state
             is the function
              f: K _ Nat.
```

A state may also be considered the number of rules that may be applied in the individual clusters. The meaning of the states shall be explained in chapter 4.3.

Finally, it must be noted that the set of all measures can be integrated well in a hierarchical schema. Certain measures automatically result in other measures. When evacuating the population for instance, a warning has to be given first. As the rule selection mechanism suggested is always aimed at deriving the strongest measure first, the derivation mechanism can be stopped at this point. Derivation of less strong measures, which may take a lot of time, may be done without. The hierarchy may be used directly to obtain the remaining implicit measures. There are exceptions, however, if e.g. several catastrophes occur simultaneously and various measures, which do not result in each other, have to be taken at the same time.

The resultant structure of the knowledge database is represented in Fig. 12.

FIGURE 12 Structure of the knowledge base


Input of expert knowledge
User view: rules in the natural language, meta rules for the resolution of conflict
Transformation
Logical representation: rules in Horn logic, meta rules
Transformation
Technical representation: as relational DB under INGRES
Problem type
Conflict resolution
Limits
Limits the set of rules
Pre-selection
Measured data
Derived facts
Rule base
Output of countermeasu

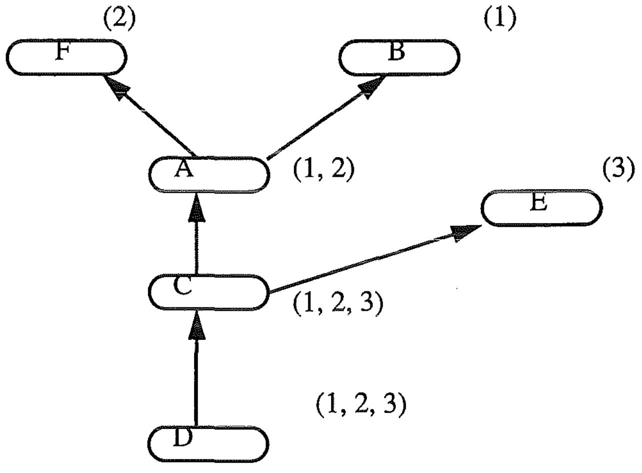
4.2.3 Other possibilities of improvement

Testing, whether a rule may be applied, takes a lot of time. However, it is often possible to exclude others together with a single rule. For this purpose, data are stored as to which rules depend on individual events. If it is known that a certain event does not apply, all rules depending on this event may be excluded. If a certain event applies, however, rules depending on this event become applicable. The following set of rules is given by means of example:

```
X _ D, C, A, B
Y _ D, C, F, A
Z _ D, C, E
```

This set of rules is based on the data structure shown in Fig. 13.

FIGURE 13 Data structure for managing the applicability of rules



This diagram has the following meaning: each node exactly corresponds to a literal of the rules. At each node, the depending rules are given in brackets. If rule 1 shall be applied and it is found that A does not exist, rule 2 can be excluded at the same time.

This data structure can be further extended to ensure the derivability of rules from newly derived facts. If it is known that fact A does not let the premise of the rule apply, while all other prerequisites remain applicable, it may be concluded that rule 1 is applicable, if A is derived in a certain step.

If a fact A only prevents the rule 1 from firing, the entry (1(A), ...) is made under node A. For each fact derived, the rules blocked can be found under the respective node. From A it can be noticed that rule 1 is blocked.

This procedure may be transferred analogously to predicate-logical formulas. As the rule base exists in the Skolem form, predicate-logical formulas may be replaced by statement-logical formulas, which result from the basic instances of the former, according to the theorem of G_odel, Herbrand and Skolem.

Another case is what happens, if a prerequisite is not fulfilled within a premise. This, however, would exceed the scope of the present chapter. The procedure is described in detail in the RETE algorithm [Brown85].

At this point, the benefit for the knowledge base only shall be of importance. As in catastrophe handling a test as to whether a rule can be applied may take a lot of time due to the distribution of the facts, the number of these tests must be limited. This may be achieved by means of the present algorithm.

Furthermore, the finding of the most general description operator may take an exponential period of time when testing the rules (example: $p(x_1, ..., x_n)=p(f(x_0, x_0), ..., f(x_{n-1}, x_{n-1}))$. As only literals will occur in the facts of the databases, however, finding of a description operator may be reduced to a simple pattern adaptation, which is easier to programme.

4.3 Communication with the user

In this chapter, the user view of the system shall be presented. Internal implementation by a relational model shall be hidden from the user. In contrast to this, the rule structure of RODOS knowledge management, which is internally imitated by INGRES, shall be directly visible to him. The user interface of an expert system generally consists of three components:

- dialog interface
- knowledge acquisition component
- explanation component.

The **dialog interface** is used to input facts. In the course of this process, the system may ask questions to test the applicability of certain rules. Hence, a solution is reached by the system interactively with the user. Such a case is encountered e.g. in medicine. RODOS is supposed to reach the solution alone, as it may be assumed that all facts required exist in the databases. It is evident from the previous chapters that the problem of RODOS is not to have suitable, but to find the correct facts from the vast amount available. Nevertheless, a dialog interface is useful, if catastrophes are to be simulated by the user (e.g., for an exercise). As it may be difficult for the user to input suitable facts, the system has to support this process by requests for further particulars.

By means of the **knowledge acquisition component**, new knowledge (rules, meta rules) is input or existing knowledge is read by the user. The knowledge acquisition component reproduces knowledge such that it is understood by the user and hides the actual form of representation in the expert system. Here, an easily understandable rule representation may be applied as the user view.

Using the **explanation component**, the conclusions made by the system are explained. The user has the opportunity to ask detailed and specific questions, _Why was this and not that done?_. To answer these questions, the rules applied can be used mostly. In any case, RODOS should be equipped with a sufficient explanation component such that the measures taken can be justified.

A representation, by means of which the existing rules are indicated to the user as simply as possible, is best suited for the knowledge acquisition component. Here, fixed language schemas should already be given in a graphic interface. The rules are then input into these schemas. The syntax of the schema is obvious from the Annex. An exemplary session of a rule input shall also be presented in the Annex. Existing knowledge may also be used for modification of the user interface or verification of the input rules. In addition, rules input in the user view can be simplified. During the input of rules, the following steps have to be carried out by the user:

- The user has the opportunity to specify a certain topic, for which the rules are to be input. Each rule contains the component _AREA_ indicating the field of application of this rule.
- It is input by the user when the rule is to be applied (its applicability is to be tested). Maybe this rule shall be tested always or in certain situations only. This information is given by the component _OCCASION_.
- It is input by the user on which level of the decision hierarchy the rule is to be located. There are several deciding authorities, who pass on their decisions upwards in the hierarchy. Finally, the concrete measure will be taken. This information is available in _CLUSTERS_.
- The user specifies a logical schema (logical expression) for the premise of a rule. RODOS may make available a selection of schemas for certain areas of knowledge.
- Now, the conditions to be checked are inserted into the rules. At first, the name of the condition is entered. This name is taken from a list of names known for this area and cluster. **No negation** may be entered.
- For each predicate name, certain language constructs resulting from the component _EXPLETIVES_ are made available. It is also checked, whether the input constants apply to this construct, e.g. is _Karlsruhe_ known as a place?
- For numerical variables requiring a comparison with the variables of another predicate, a direct language reference is made to this value. For example, _higher than value in_.
- Finally, the user shall be enabled to see his rule in the total concept as to when it is applied, which priority it has, which expenditure is required for the taking of measures, etc. Thus, the user shall be enabled to assess, whether this rule makes sense.

A similar procedure is applied for the input of meta rules, i.e. rules determining the selection of facts and rules. Here, the following steps are performed:

- After selecting an area, a **state** of the derivation process has to be determined. A state generally represents the set of derived facts available for the derivation process. As this would allow too many possibilities, however, let the state be defined according to chapter 4.2. It is input, how many rules can be applied to this or that group of rules.
- Then, the priorities of the individual rules can be entered.

Here, the dialog mode is mainly used for exercising purposes. Using RODOS, the user can train measures in simulated accidents. Questions and answers again are input by means of a simple schema, which is presented in the Annex together with an exemplary session. If a catastrophe is to be simulated and the inputs are not sufficent to output measures, the system has to ask questions in a skilful manner such that by answering them the user can _attain_ any case as rapidly as possible. For this purpose, the data structure presented in 4.2.3 may be applied. Then, the number of facts required for the rules to apply can be determined. By suitable questions, this set has to be divided such that the facts required are made available and useless facts are excluded. To simplify this mechanism, reference cases stored in the knowledge base can be accessed.

To specify the dialog mode, compulsory selection of the area has to take place at first. Then, the following steps are carried out:

- For the exercise mode, a set of exercise data has to be made available.
- Then, the state, reason and rule of a message text output are input.
- The message text is mostly used for the input of a new fact.

The statements supplied by the explanation component are as simple as possible. An exemplary explanation is given in the Annex. It should also be possible to output partial solutions or let the system evaluate solutions prepared by the human user. The system shall indicate which damage is generated by realizing a certain solution or by the taking of no measures. Reference cases may serve as examples. Of course, detailed arguments have to be given for a measure taken. They are obtained from the rules used.

4.4 Implementation of the logical schema

In this chapter, generation of the logical schema in Horn logic from the user inputs shall be explained.

The most important step consists in the transformation of the input rules into Horn clauses. For this purpose, the premise input is first converted into the **disjunctive normal form**[1]. Now, the rule of the form

$p \_ K_1$ OR K_2 OR ... OR K_n,

[1] A formula is of the disjunctive normal form, if it has the form OR (AND ... AND ... AND) OR (AND ...) OR ()...

where K_i denotes the conjunctions, is transformed into n rules of the Horn clause form

$p \_ K_1$

. .

$p \_ K_n$

Another problem is the determination of the priorities. As the user, who inputs the rule, hardly knows the priorities of all remaining rules for determining the priority of the rule directly, priorities of newly input rules have to be determined automatically. For this purpose, already existing meta rules can be applied. The exact order of the process shall be subject of further studies. Transmission of priorities of facts used in a new rule to this rule is conceivable. A rule with a conjunction of facts in its premise contains as priority the minimum of priorities of the facts. Priority of a fact may also equal the maximum of priorities of the rules, in the heads of which it exists.

4.5 Implementation using INGRES

The knowledge base is eventually managed using the relational database system INGRES under C. It shall therefore be studied, how the rule-based system presented under 4.2 can be mapped to a relational database concept like INGRES.

In general, logically based systems are an extension of the relational concept. The facts describe relations, whereas the possibility of defining rules represents the extension as such. Moreover, common relational database operations, such as Cartesian product, union, projection, selection, intersection and combination may be described by rules such that everything that may be done by means of relation algebra can also be expressed by logical rules.

Mapping of the derivation mechanism to INGRES is of crucial importance. The missing derivation mechanism of logic programmes has to be imitated as largely as possible. The main task will consist in deriving the facts by rules using adequate SQL constructs. Other processes, such as the selection of rules, have to be programmed manually under C. The constructs presented in the following chapters are therefore rather general and can be further modified depending on the concrete structure of the database.

It must be noted that the distribution of the database system does not affect the logical design of knowledge management, as this distribution of the system is hidden from applications and users. This means that each database operation could be run in the same form on a non-distributed system. Therefore, only the criteria influencing the performance of the derivation mechansim have to be taken into account. Implementations, which are exceptionally quick on a non-distributed system and below the average speed on a distributed system, have to be excluded.

4.5.1 Implementation of rules using INGRES

It is started from the fact that the rule base does not contain any recursions (in the latest version of INGRES, recursions shall also be admitted in relations such that this limitation may be dropped). At the bottom of the hierarchy relations are found, which describe given facts. The relations at the top describe the measures to be taken.

Starting-point is a group of rules for a new relation p:

```
p(x) _ p1(x1), ..., pn(xn)
.........................................
p(x) _ q1(y1), ..., qm(ym)
```

Let all other relations be base relations except for p. They are generated in INGRES using _CREATE TABLE_. As the creation of databases is not the task of knowledge management, it shall not be covered in further detail.

Access to these databases is made using _RETRIEVE_. As no new data are created, views can be used. Each view describes a new relation. The general schema of a view for the above rules is as follows:

```
CREATE VIEW p(x)
AS SELECT p1.x, ..., pn.x FROM p1, ..., pn
WHERE [specify identities]
UNION
..........
UNION
SELECT q1.x, ..., qm.x FROM q1, ..., qm
WHERE [specify identities]
```

Comparison of variables under (identities) is represented by the AND conditions in the rules. The rule

```
trans(x, z) _ p(x, y), q(y, u), o(u, z)
```

has the query:

```
CREATE VIEW trans(x, z)
AS
SELECT x, z FROM p, q, o
WHERE p.y=q.y
AND q.u=o.u
```

It is a drawback that the intermediate relations generated may become very large, as the Cartesian product is formed over the individual predicates. In this case, a join has to be made do with. The procedure is similar to that of a logical programme. All relation pairs, which fulfil the join conditions, are formed. Nevertheless, both versions have their advantages and drawbacks. A system like INGRES can put indices on the attributes of the join conditions in order to be able to form all possible pairs very quickly. In contrast to this, a logical programme has the following advantage, provided that it has been written skilfully enough: if a rule g(x)<-h(x, y), p(x) occurs and y has been instanced such that h(x, y) is false for all values of x, p(x) does not even have to be considered for the rule to be dropped. INGRES, however, will first try to form all relation pairs again under which the condition is fulfilled. If p would have been evaluated prior to h, the logic system would not have been quicker than INGRES.

As views can use views again, provided that no recursions occur, the rules can be extended by this schema. Finally, e.g. a view of the form _measures(x)_ can be defined such that a measure desired can be output using SELECT * FROM measures.

This would represent a reverse chaining, as the view _measures()_ addresses the next views below etc. Neither could the derivation process be controlled any longer, as INGRES takes over the addressing of the individual relations. A possibility to escape this dilemma would be the definition of sub-views, such as _measures during smog(), measures during floods_ etc., which could be addressed by the officials for the reduction of the number of rules and facts used. Inflexibility of reverse chaining is clearly noticeable. Combinations of various catastrophes cannot be taken into account.

4.5.2 Implementation of forward chaining

In the present chapter, the individual mechanisms for the implementation of forward chaining shall be explained.

4.5.2.1 Basic methods and data structures

To get away from the concept of reverse programming, the addressing of views by views must be prohibited. Views shall only address base relations. Views directly accessing the master data can remain unchanged. However, all data generated by a query of the form SELECT * FROM VIEW have to be stored in a new relation. Then, this relation is addressed by the views of the following higher level. This shall be illustrated by the following example:

```
ozone alarm(place) _ ozone value(place, x), ozone critical
            (place, crit), crit<x
measure(_driving ban_, place) _ ozone alarm(place),

heavy traffic(place)
measure(_production ban_, place) _ ozone alarm(place),
much industry(place)
```

The relations _ozone value_, _ozone critical_, _heavy traffic_ and _much industry_ are contained in the external databases. Furthermore, it should be noted that in accordance with the statements made with regard to the hierarchy of measures the rule

```
measure(_driving ban+production ban_, place) _
ozone alarm(place), heavy traffic(place)
much industry(place)
```

should have been introduced, as the derivation process shall be terminated after the first measure derived. For further details, see next chapter.

The following views and relations have to be created:

```
CREATE ozone alarm(place= CHAR(20))
creates a new relation ozone alarm.
CREATE VIEW ozone alarm(place)
AS SELECT place
FROM ozone value, ozone critical
WHERE ozone value.place=ozone critical.place
AND ozone critical.crit < ozone value.x
```

realizes the first rule.

```
RETRIEVE INTO ozone alarm(ozone alarm.place)
```

fills the relation ozone alarm with all places, where ozone alarm is given.

```
CREATE VIEW measure(what, place)
AS SELECT _driving ban_, place
FROM ozone alarm, heavy traffic
WHERE ozone alarm.place=heavy traffic.place
UNION
AS SELECT _production ban_, place
FROM ozone alarm, much industry
WHERE ozone alarm.place=much industry.place
```

realizes the last two rules with the newly created relation being applied.

Thus, forward chaining can be implemented. To render this mechanism efficient, the rules have to be grouped such that certain rules only are applied depending on the situation. Furthermore, the rules have to be sorted in accordance with the strength of their pre-condition. The following relations are needed:

```
1. CREATE TABLE basic values (priority: integer, relname:
            CHAR(20), data: CHAR(20), type: CHAR(10))
```

creates a relation _basic values_, where the priority, with which relations from external databases have to be addressed in order to ascertain the occurrence of a catastrophe and its approximate nature, is specified. For our example, an entry may be as follows:

```
(34, _ozone value_, _big city(x), y_, _ozone_) or (35, _ozone
            value_, _small city(x), y_, _ozone_)
```

This means that in 34th place the ozone value of all big cities shall be investigated using the rules of the class _ozone_. If a catastrophe has occurred, there should be a table of the facts to be checked in the analog form.

Another relation manages the rules:

```
2. CREATE TABLE rule (nr: integer, viewname: CHAR(20), area:
            CHAR(10), event: CHAR(10), cluster: integer)
```

An entry may be:

```
(1, _ozone alarm_, _air_, _derivable_, 3)
```

In addition, the priorities of the rules have to be stored. This may be done by a second relation, as the contents of this relation may vary contrary to the previous one.

```
3. CREATE TABLE priority (nr: integer, identification: integer,
            first prior: integer, sec prior: integer)
```

It shall also be stored in a table which rules depend on which rules.

```
4. CREATE TABLE rule dependences (viewname1, viewname2: CHAR(10))
```

For example, the rule A<-B, C depends on the rule A<-B, C, D. This rule pair is then stored in the table. For further details see next chapter.

For the hierarchy of measures, a table

```
CREATE TABLE measures (successor Id, predecessor Id: CHAR(10))
```

can be defined. The measures as such can be listed in another table, which is referred to by * Id. The table of the measures as such is object-oriented, i.e. the predecessor measures leave their procedures to the successor measures.

As catastrophe management is a non-monotonous rule mechanism, it must be possible to delete information, if facts are changing. For this purpose, the DELETE instruction is used. Example:

```
DELETE ozone alarm
```

```
WHERE
ozone alarm.place IN
SELECT place
FROM LOCATION
WHERE state=_Hessen_
```

4.5.2.2 Algorithmic implementation

Implementation as a concrete programme covers several fields. As the presentation of the complete programme text would exceed the scope of this report, some indications as to the general procedure shall be given.

Recognition of catastrophes: first of all, the catastrophe has to be recognized. How this is done, was indicated in the previous chapter. But even after a catastrophe has occurred and evaluation has been started, the distributed regional databases still have to be controlled, as basic boundary conditions may change. It is best to apply a separate process, by means of which the corresponding messages are sent to the remaining knowledge management system. If other catastrophes are recognized, it would be reasonable to manage a process for each individual catastrophe. It may also be helpful to allocate computing time to the individual processes according to their significance. The basic pattern of such a process may be as follows:

```
WHILE TRUE DO BEGIN
FOR priority:=max DOWNTO 1 do begin
address predicate at the respective position in the
            table;

IF catastrophe condition fulfilled THEN BEGIN
create process of appropriate priority for measure
            derivation;

create process for controlling the variations of the distributed
            data;
END

ELSE BEGIN
IF EXISTS processes for this catastrophe THEN BEGIN delete all
            these processes;
END;
END;
```

```
END;
END;
```

Of course, there are also other possibilities of handling the individual priorities. However, they shall not be explained in detail.

The derivation mechanism: the basic structure is forward chaining. However, it is extended by the features mentioned above. In total, the following schema is obtained:

```
initialize data structure A with applicable rules;

REPEAT
priority:=max;
IF rule R with priority applicable THEN BEGIN
apply rule.;
extend set of all derived facts by rule head of R;
enter the resultant newly applicable rules into A;
delete all rules depending on R from A;
END
ELSE BEGIN
delete the resultant inapplicable rules from A;
END;

IF rule head of R a measure THEN

output measure;
END;

priority:=next applicable rule of A;
UNTIL A empty;
WHILE TRUE DO;
```

Here, a number of explanations is required. In chapter 4.2, the mechanism of concluding the applicability or inapplicability of other rules from the applicability or inapplicability of one rule was presented. In our case, A represents such a data structure. After each test, it is modified accordingly. Furthermore, a table was presented, in which the dependences of rules are stored. Deletion of all rules depending on an already applied one is aimed at avoiding the derivation of resulting measures with dependent rules, once a certain measure has been derived. When using this algorithm, it is not necessary to introduce the rule _measure(_driving ban+production ban_, place)<-..._ mentioned in the example, as the rules relating to the driving and production ban shall not be dependent on each other. The dependences can be defined by the user in any way.

By _output measure_, the mechanism managing the measure hierarchy is initiated and the bunch of measures is output.

If rules are no longer applicable, derivation is terminated. However, the process shall not stop to exist, but turn to a waiting position. Later on, a signal may come from the processes managing the initial data of the distributed databases, indicating that modifications have occurred. In this case, modified measure proposals have to be output. Hence, it should be possible to access already prepared data sets again. The response to the signal indicating the modification is realized as an interrupt. The response procedure can have the following format:

- delete all corresponding facts;
- delete all rules that are no longer applicable;
- re-start the derivation process at the applicable rule of highest priority.

4.5.3 Problems of implementation of the logical schema in INGRES

From the logical schema, the corresponding relations can be formed directly and stored in INGRES.

The views are generated directly from the description of the Horn clauses. Several problems arise when they are implemented.

Rules having the same head often vary in priority. Up to now, all these rules were integrated in one view using the UNION operator. Therefore, the rule base has to be rewritten as follows:

A rule

$$p \_ R_1;\ p \_ R_2;\ \ldots;\ p \_ R_n$$

has to be rewritten as

$$p_1 \_ R_1;\ \ldots;\ p_n \_ R_n$$

where p_i denotes the new predicate symbols. For each of the n rules, an own view has to be introduced. In addition, a rule

$$p \_ p_1,\ \ldots,\ p_n$$

has to be implemented in order to be able to use the symbol p in other rules.

A major problem consists in the access of the priorities of the individual premises while testing a rule. The only solution is to convert a formula of the form

$p \_ p_1, \ldots, p_n$

into a set of formulas

$q_1 \_ p_1, p_2$

$q_2 \_ q_1, p_3$

$p \_ q_{n-1}, p_n.$

All these rules have to be provided with a mark indicating that they all originate from the same Horn formula. In this case, let us assume that the priorities of p_i are sorted in descending order. If the priorities are distributed differently, sorting must be changed accordingly. The priority of the initial rule is allocated to the rule priorities as the first value. The priorities of the individual premises are entered as sub-priority. It is only taken into account, if two rules have the same marking.

The drawback consists in the fact that once the priorities of the premises have changed, the entire mechanism has to be started again.

Technical generation of the views may also cause certain problems. Here, it is possible to generate SQL files, which are then translated and integrated dynamically.

4.6 Alternatives

As an alternative, RODOS may use an expert system shell instead of INGRES. The advantages, however, still have to be examined in detail.

When doing without the implementation of rules with views, the derivation will be programmed directly. Views have the drawback that the order of testing of premises cannot be controlled.

Application of a deductive database is very promising. As this field is relatively new, only limited experience has been gained so far and few products are available only. Hence, the number of implementation possibilities is rather limited.

4.7 Summary

The version proposed in the present chapter has a number of advantages, in particular as far as the representation of information required for the implementation of decisions of the authorities is concerned. Here, they shall be summed up briefly:

- User friendliness

As explained in the previous chapter, a rule-based system provides a number of possibilities for a user-friendly interface. The view of the user comprises rules, internal representation may be hidden. These rules can easily be extended and adapted by RODOS. As the rule base is divided into individual topics, relevant positions can be found rapidly. By the natural-language interface, implementation of legal provisions into logical rules is facilitated.

- Consistency

The rule base is directly oriented according to the hierarchical decision procedure of authorities. Consequently, the decision procedure is mapped 1:1. The user interface of RODOS also supports the user when preparing adequate inputs and prevents the input of nonsense. Newest data are accessed only. If modifications occur, already derived facts can be deleted.

- Efficiency

Rules and facts can be divided such that a derivation process can access a part of the rules and facts available only. Furthermore, dependences among rules can be defined and experience gained from the applicability tests of rules can be taken into account in order to exclude the applicability of certain rules right from the start. Using INGRES, indices can be put on the stored data. Thus, the finding of existing rules is accelerated. Knowledge management can be divided into several parts, which may be run in parallel on various computers. This again accelerates the process.

Chapter 5 Real-time data management

5.1 Introduction

Measured data represent one of the major bases of the decisions to be made in the RODOS system. Measured data, which are a special type of real-time data, are collected all around the clock. They describe the actual state of the environment using parameters, such as e.g. temperature, wind speed and direction or radioactivity. The measured data are expected to have several hundred Gbytes. They are modified in exceptional cases only, once they have been input by the measurement process control programmes. As all other data stored in RODOS, the measured data have to be made available rapidly to the user in an emergency. Contrary to the other data, measured data are also modified and input in an emergency.

The present chapter deals with the problem of real-time data management. The objective is to design a database and the auxiliary programmes required to make use of the special circumstances (particularly access patterns) and, hence, to ensure functionality and sufficient performance.

The real-time database is implemented on the basis of a relational database system (probably INGRES).

5.2 Data characteristics

Data characteristics is of preliminary nature, as stocks of measured data do not yet exist and few information only is available on the planned accesses to these data. In the course of the project, the characteristics will be further refined. This may result in new requirements to be complied with by the measured-data management.

The measured data consist of the following components:

- **`Time of measurement`**
  When entering the results measured, the data generator (= a programme controlling the measurement process) has to indicate, whether local time or CET applies. Thus, the database system

is enabled to answer queries in both formats. Several data generators sharing a common database may also be located in different time zones. Depending on the situation (in an emergency probably more frequently) and the type of parameter measured, the measured results are entered at variable intervals. For this reason, the database has to support various time resolutions.

Recommendation: it is recommended to store all measured results using the same time zone.

- **Place of measurement**
  There are several possibilities: the respective database may only collect local data - the place consequently is determined by the names of the database (unambiguously for data generators and consumers). Or the database may directly take up the place. Here, the place may be indicated by names (e.g. temperature in Karlsruhe) or by geographical coordinates. The latter require less storage capacity than the complete names and allow more rapid processing of area-related queries.
  Recommendation: it is useful to introduce a uniform notation for the indication of places.
- **Name of the parameter measured**
  If various parameters are to be stored in a database (a relation), the parameter name is needed. Alternatively, an extra relation can be created for each parameter.
- **Actual parameter value**
  For simplicity, all values shall be stored in the same units (either in degree Celsius or Fahrenheit).
- Measurement accuracy
- If necessary and if certain parameters are to be measured by various measuring devices, measurement accuracy shall also be entered.
- **Additional information**
  If required, additional information may also be entered in the database: source of the results measured, confidence factors of uncertain measurement techniques, etc.

When dividing the measured-data base into relations, it is useful to keep the relations as specific as possible. Example: the database with the air temperature in Germany may be divided into relations indicating the places of measurement: Karlsruhe, Mannheim, Freiburg, etc. Thus, many of the queries expected can be answered more rapidly. Moreover, less storage capacity is needed compared to a relation storing the place of measurement in each tuple. A special relation may also be splitted into time-dependent parts. Example: air temperature in Karlsruhe in July 1994. This type of division may be helpful when filing data (cf. chapter 5.7).

The measured-data management component has to take over the transformation of queries. As the applications cannot be modified, division of the relations must be hidden from them. This also allows dynamic modification of the division.

5.3 Treatment of limit values

It must be possible to specify limit values for the parameters. When exceeding these values, a predefined action shall be carried out by the database system. Examples of possible responses are:

- Starting of any programme both on the local workstation and on any other computer of the network.
- Notification - notification of a user, application by e-mail or UNIX signal mechanism.Termination of certain transactions (e.g., statistics applications). The transactions are selected on the basis of the locks kept.
- Measured-data management must comply with these requirements.

5.4 Typical queries

Optimum database design can only be achieved, if the queries are known in advance. Some queries that are very probable in the RODOS system from the FZI point of view shall be listed below.

- Find the latest values of parameter X and place Y;
- Find the value of parameter X for place Y at the time Z;
- Find the values of parameter X for all measuring points in Europe at the time Z;
- Find the latest values of parameter X for all measuring points in Europe;
- Find the places, where parameter X has the highest value at the moment;
- Find the places, where parameter X has exceeded the limit value;
- Development of parameter X at place Y in the time interval from A to B.

The real queries will only be available when testing the prototype. They allow the use of indices and other measures aimed at increasing the performance.

5.5 Locking

The users of the measured-data base can be divided into four groups:

3. Measuring programmes introducing the latest measured results and modifying no existing measured data.
4. Administration tools modifying the existing measured results. These programmes have precedence over all other programmes.
5. Decision programmes allowing _dirty read_. These programmes can work simultaneously with the programmes of group 1.
6. Decision programmes that cannot be active together with other writing programmes.
7. To implement this division, new lock logs shall be defined in the database system.

5.6 Data compression

Due to the high amount of data, they have to be compressed after a certain period (e.g. one month after entry). Compression can be carried out in several phases (after a day, week, month, etc.). After each phase, storage requirements shall be reduced. Compression of measured data can be divided into two groups: compression without or with deterioration of data quality.

- **Compression without deterioration of data quality**
  Parameter A has the value q from time x to time y. In the database, several data records exist for this time period. They only differ in the time of measurement. These records can be converted into a single compressed record.
- **Compression with deterioration of data quality**
  From time x to time y, the value of parameter A ranges from p to q with the critical value not being exceeded. The intervals between the results measured can be increased. Alternatively, a tolerance range can be specified for a measured value. In the database, the mean value as well as the time period, during which deviation from the mean value is in the tolerance range, can be stored.

A special problem is the modification of already compressed data. In some cases, the data have to be decompressed prior to modification.

5.7 Filing

Despite of compression, the volume of the measured data will exceed the capacity of the secondary memory after a certain time. On the other hand, most of the queries are addressed to the latest measured data, while old data are used less often. In this situation it is necessary and possible to file data, i.e. to transfer them to external tertiary storage (WORM, magnetic tapes, etc.). The system is supposed to file data on its own. Data transfer to external storage takes place when the secondary memory is filled to a certain degree or when the data reach a predefined age. Measured-data management also takes over the logical management of the library of storage media.

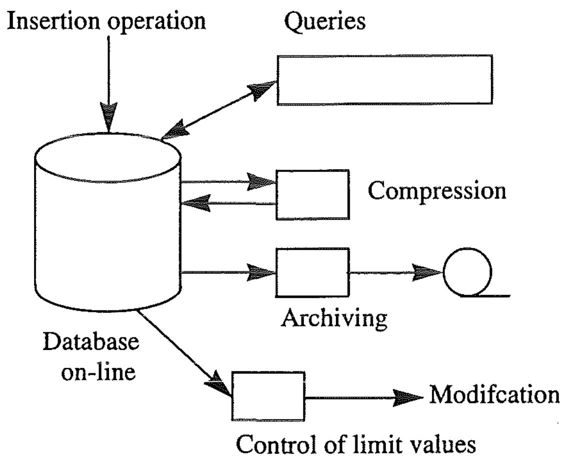
If required, the filed data have to be made available to the user. The accesses remain completely transparent. Due to the much longer access time only, the user may presume that filed data are accessed.

5.8 Aspects of implementation

The measured-value database shall be implemented on the basis of a commercial, relational database system. As already mentioned, the decision as to which product will be applied has not yet been taken. Entire architecture of measured-data management is illustrated in Fig. 14. Due to the schema transformations, all operations (both reading and writing operations) are converted into the actual physical schema. The system is also equipped with other components, which are implemented as separate programmes (processes):

- tool for the control of limit values - notification;
- tool for data compression;
- tool for data filing.

FIGURE 14 Rough architecture of measured-data management



5.9 Requirements made on the database system used

Measured-data management does not make great demands on the database system applied. Besides the usual functionality, it would be very helpful, if extendable locking and an active component were offered by the system. If these features are not available, they can be implemented as additional software components.

Chapter 6 GUI generation

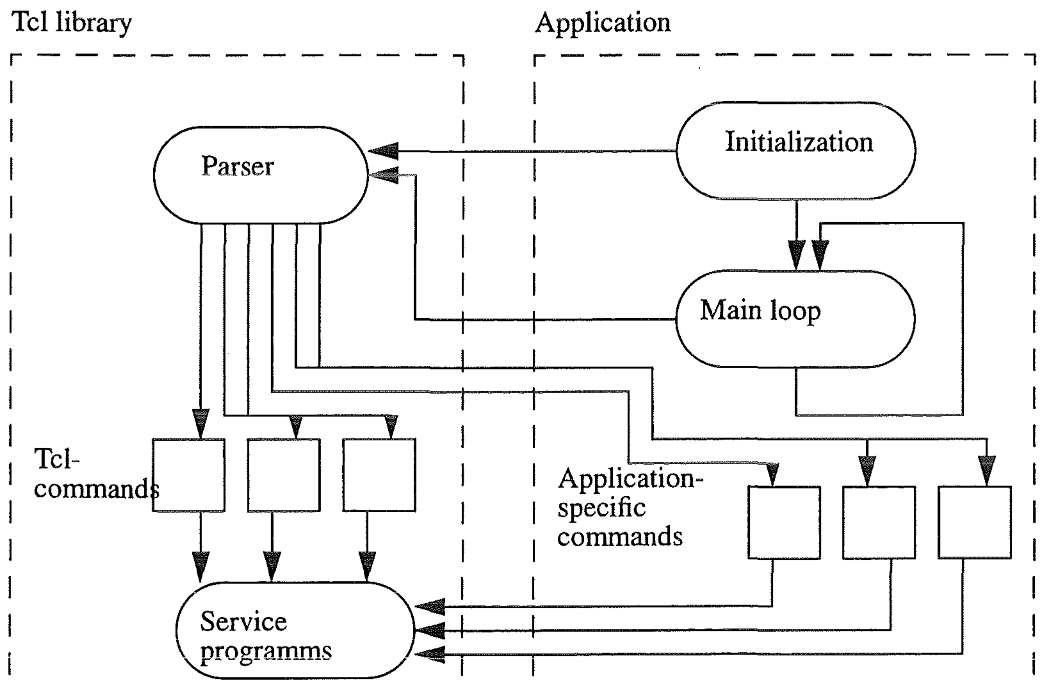
6.1 Introduction

In the present chapter, functionality of the tools for the generation of graphic user interfaces shall be described by means of the Tcl/Tk example. The decision as to which system will be used for the implementation of RODOS will only be taken, when the other tools (in particular for database integration) will have been selected.

Tcl/Tk is an efficient system for the rapid generation of user interfaces. It consists of a script language as well as of a toolkit system. Using the script system, textual description of a graphic user interface can be carried out, whereas the toolkit system is used to generate this user interface directly by hand. Tcl/Tk has the advantage of being available at FZI already. It has been tested and sufficient (positive) experience has been gained. Thus, development time of the user interfaces can be reduced considerably. In addition, Tcl/Tk is cost-free. However, it cannot be directly operated graphically. Hence, it requires the generation of some additional tools in order to be comfortable for the user.

In general, the application programme, database access and user interface are separated such that all components can be generated and modified separately. Each component uses the services provided by the other components via special interfaces. The application programme is available in the C code. In the main loop, user activities are waited for. Once an activity has been performed, queries are addressed to the user interface via a parser. The general structure is obvious from Fig. 15.

FIGURE 15 General structure of an application



6.2 Graphic I/O elements

A number of graphic elements used for the generation of comlex applications is made available by Tcl/Tk. The technical term of such a graphic element is **widget**. It is made up of the words _window_ and _object_ and means that a graphic element is a window located in a certain hierarchy and having a certain functionality. The elements of the hierarchy also are widgets with the top widget usually being the window containing the application. As a rule, a widget is a predecessor of another widget, provided that the latter is included graphically in the former. All widgets can communicate with each other: if the size of a widget is varied, messages are sent to its predecessor, as a result of which its size is also varied.

Widgets are distinguished in accordance with their location in the hierarchy and their functionality. The following main groups exist:

- Container widgets: they serve to take up other widgets. Their main purpose is the graphic and logical structurization of the application.
- Scroll widget: they serve to scroll the contents of the widgets allocated to them.
- Input/output widgets: they serve to input or output texts or graphics.
- Action widgets: they serve to trigger certain actions. The best-known examples are buttons or menus.

- Selection widgets: they serve to store settings.

After pressing a certain button, all widgets supply the values requested by the application or the X-window system, e.g. a text string input in a field or a function reference.

6.3 Control language

6.3.1 User interface

All widgets can be processed from the application by means of appropriate commands. Values can be called or entered into the user interface. Navigation through the user interface (opening of sub-menus, etc.), however, takes place autonomously. The application programme has nothing to do with the representation of the user interface.

Generally, the control language makes available both a script language and a toolkit system. The script can be carried out by an interpreter or compiled first and transferred to a DLL. A script has the general advantage that minor modifications can be performed more rapidly and precisely by a trained user. The toolkit system is suited above all for the untrained user. Another aspect consists in the fact that even compiled user interfaces do not have to be re-compiled in case of modifications. They can access **resource files**, where textually important parameters (colours, size, etc.) of the application are described. These values can be modified manually.

6.3.2 Event/trigger mechanisms

With each action executed by the user (e.g. select menu entry) a certain **event** is generated. Generation of the event is carried out by the **X-server**. The X-server is a programme under X-windows, which controls the activities of the user (keyboard, mouse click, etc.). When initializing a user interface, a link to the X-server is established such that the events are sent to the user interface. The event then has to be processed. For this purpose, it has to be checked, whether a user-specific response is planned for this event. A table of the form _event/response_ has to exist (invisibly) in the user interface. As the entry for a user-specific response, the name of a certain function or procedure to be executed is mainly used. The table is created by entering the name of the function to be executed for each menu entry in the script. Other forms are also feasible, e.g. separate storage in a file. They have the advantage of the access of functions being independent of the user interface. If a new function is to be addressed from a certain point of the menu, the user interface does not need to be re-compiled.

6.3.3 Database system link

Linking of a database system in Tcl takes place dynamically via a DLL (dynamically linked library). Thus, compilation of the database-specific function can be performed independently of the remaining application. While linking the database, the data structures stored in the database as well as the functions triggering the actions are made available. Similar to the user interfaces, a toolkit system and a script language are provided, by means of which the database can be generated.

6.4 User support for query generation and other tools

By means of the methods presented, tools for query generation and schema evaluation can be generated.

Using a query, data of the database are requested. The query may be very complex. A typical example: find all hospitals with more than 100 beds within a radius of 50 km of Karlsruhe. To create a user-friendly interface for the generation of the query, the user should be guided through the variety of options of preparing queries in a menu-controlled manner.

Starting from a descriptive, SQL-type query language, the following problems may arise for the user:

- A SQL query may refer to interlaced structures. This interlacing should be represented graphically such that the user keeps track. A suitable representation may be e.g. a tree, whose nodes indicate the type of link of the queries below. It should also be marked by colours which query is currently processed by the user. This may be achieved by a canvas widget, which supplies the necessary graphics options.
- It is of crucial importance that the user keeps track of the multitude of relations and attributes available. To select a certain relation, as it is done in the _SELECT FROM_ part of each query, first a topic, then a logically associated group of relations and then an individual relation shall be determined by the user. For this purpose, popup menus are used. They store a selection that has already been made such that e.g. the topic does not have to be input again and again. Pulldown menus can also be applied. In this case, sub-menus can be defined.
- Simultaneous explanation of the effect of the SQL instruction that has just been given (e.g. _group by_) should be as good as possible. As an example, the result of this instruction should be displayed in a special window. In our case, this would be the grouped data. A suitable means is the text widget.
- As shown above, the query can largely be generated in a menu-controlled manner, as also the commands can be input in this way. If nevertheless textual inputs of the user are needed, text input widgets can be used for entering the text. With the end of the input a certain action may be associated. It may be used for testing the input consistency.

- A clear structure of the user interface also is important. For this purpose, numerous container widgets can be used. They serve to specify the layout of the user interface.

The same tools may be employed for schema evaluation. Schema evaluation is described in

chapter 3.

Furthermore, a number of tools, which may also use a graphic user interface created with Tcl, have to be realized for DB administration.

One of the most important tools is the data browser. It gives a complete survey of the database: it shows all data used, their access paths, the performance parameters of the computers involved in the data network, etc. Realization of an appropriate user interface requires numerous elements: besides popup and pulldown menus for the execution of user commands, graphics for the representation of state information, scroll lists, from which the elements are selected, elements for setting of parameters, etc. are needed.

Chapter 7 Feasibility analysis

7.1 Doubts

At this point, FZI feels compelled to express some doubts concerning RODOS and integrated data management. These doubts can be divided into several categories. At first, a number of unknown factors exist, such as the scope of the data to be managed in RODOS, the access behaviour, etc. This information largely consists of quantities (that can be calculated). They have to be specified more precisely in order to be able to decide later on, whether requirements based on them are met or not. However, some of the corresponding questions will probably be answered upon the completion and acceptance of the first prototype only, e.g. in the form of performance measurements. Up to date, statements as to the functional scope planned or the runtime performance of RO-DIM have only been made within the framework of the present study. It is therefore nearly impossible to make statements regarding the actual use of RODOS. This especially applies to RO-DIM performance in an emergency. Other doubts refer to general technical and special database-technical aspects as well as to the organization.

7.1.1 Unknown factors

It became obvious during previous discussions that it is impossible to obtain precise information on all the services to be rendered by integrated data management of RODOS. An exact and complete specification of the requirements in the form of a system specification of course can hardly be made at the moment. As outlined in chapter 2, precise specifications have been made neither for the validation scenario of the first prototype nor for the emergency. Most important unknown factors are given below (unless otherwise provided, they shall apply to the first prototype and the entire project both for normal and emergency operation):

- Data structures and scope
  At the moment, precise statements as to the data to be managed in RODOS are unknown to FZI. They have only been divided into the categories mentioned in the previous chapters, namely, knowledge data, measured values (real-time data), etc. Further classification e.g. in the form of concrete schema descriptions are not known, nor is the scope of the data to be managed in RODOS at the various places.

- DB operations, data transfer and transaction volume
  It is not known which DB operations will have to be carried out at which times. Up to now, only basic limitations are planned. The scope of DB operations expected is unknown. Their number depends on the transaction volume expected per node. Neither is this known, nor the scope of the data that have to be transferred by DB operations.
- RODOS applications
  It is not known which and how many applications are to be applied in RODOS and which capacity (CPU performance, primary and secondary storage capacity, network transfer rate, etc.) will be required. Many of these data items shall be obtained from the emergency operation plans. They have to be determined in the course of time in order to be able to assess realistically the applicability of the entire system.
- Scheduler
  Few parameters only are known for the RO-DIM scheduling component. They shall originate from various fields. For the RODOS applications, e.g. their type and scope, their importance in an emergency, the dependences among them and on the data used by them, etc. should be known. Other factors, such as the type, scope and importance of DB operations, machine capacities and loads, network capacity and load, etc. may be added. In a second prototype at least, these parameters shall be covered in further detail.
- Secondary memory
  The actual secondary storage capacity of the applications and RO-DIM can hardly be estimated in advance. During local schema modifications at least (s. chapter 3), extremely high secondary storage capacity may be needed due to double data storage. Deliberate multiple data storage may also be necessary to ensure that the entire system is functioning as reliably as possible.

7.1.2 General technology

Given the complexity of the entire RODOS system and its extremely high functionality and performance, which are required in an emergency at least, it is not sure, whether the planned hardware and (system) software are sufficient. This aspect (unfortunately) cannot be clarified in advance, as the actual requirements to be met by the RODOS system are still very vague. As an ideal conception, some critical factors shall be listed below:

- Hardware performance
  The hardware applied may not be efficient enough. At least it is not sure, whether sufficient performance reserves will be available by scaling in the form of additional (autonomous) workstation systems or workstation clusters. This question may probably only be answered satisfactorily by means of simulations and more precise performance requirements.
- Network
  Capacity and behaviour of the network represent an unknown and sometimes critical quantity in particular in systems using widely distributed networks. The currently specified data transfer rate of 4 Mbit per second (s. chapter 2.5.2) may be too small for the large amounts of data to be transferred. The network may also be a weak point of the entire system, if it is not available in a sufficiently redundant manner.

- Framework
  The entire RODOS system should have a better control of the tools used such that they can be excluded according to their importance, if applicable. For this purpose, further descriptions of the tools, their users, their importance in the case of use, the data handled by them, etc. are needed. Emergency operation plans may serve as a first basis.

7.1.3 Database technology

In analogy to the previous chapter, major problems arising in the field of database technology shall be outlined:

- Integration and distribution
  A level for the integration of heterogeneous and distributed database systems requires an additional management expenditure in the form of computing power and storage capacity. Hence, it contradicts the requirement of a soft integrated real-time data or system management (real-time transactions). Both the integration of database systems as well as their real-time requirements are still subject of research such that a complete range of reliable results does not yet exist. To achieve acceptable solutions, compromises will have to be made between the functional and the performance requirements to be met by the RODOS integrated data management.
- Database management systems
  he local relational DB management systems planned to be used in RODOS are no real-time DB management systems. A system that integrates these systems in the form of a multiple database system depends on the basic components. Hence, it can meet _soft_ real-time requirements (at the most).
- Relational database systems
  In various studies it was pointed out that relational database systems do not exhibit an extremely good performance in geographical data processing, CAD/CAM, CASE, knowledge processing, etc., which shall be employed in RODOS in a similar form. In general, they are not ideally suited for the management of complex structured data. Relational systems have to reflect complex data structures in the form of various relations. Subsequent combination to initial data takes much time. Therefore, a system that can manage variably structured in addition to relational data would be very useful. Examples are the so-called object-relational database systems.
- Time behaviour
  As indicated above, the requirements made on the RO-DIM time behaviour, in particular with regard to the ten-minutes actualization intervals in an emergency, cannot be assessed without further information. This includes the above-mentioned knowledge on the data structure, the queries expected, the data and transaction volume, etc. (Even with this information, first simulations using the first prototype will be required). In analogy to the scaling of the entire RODOS system, it is not sure, whether scaling over e.g. three workstation clusters can be transferred to higher numbers.
- Organization
  Organizational aspects are not covered by FZI. Nevertheless, it must be pointed out that the

administration of the RODOS integrated data management will represent a very demanding organizational task already (see also chapter 3.2.1).

7.1.4 Conclusions

RODOS integrated data management is a technical challenge. In spite of the doubts expressed above, continuation of the project would be reasonable. However, the partners involved must be ready to make compromises. The major difficulty of this project consists in the unknown factors. As long as they are not quantified, no measures can be fixed for a service being fulfilled or not. They can only be given to the extent specified in the present study. Therefore, it must be objective of the first prototype to get to know the actual system requirements in more detail.

The general technical doubts seem to be more or less solvable by an increased resource expenditure, if all other methods fail. This, however, does not apply to the conception of the entire system as a framework. This aspect should be covered in more detail in the next prototypes at least. The dependence and priority descriptions may represent a first step in this direction. Information on short-comings may be supplied by the first prototype. Moreover, scaling of the entire system will have to be checked.

The doubts expressed in the database field cannot be dispelled completely. Provided that adequate tools are available, integration and access to the partial database systems seem to be solvable from our point of view. In the prototype, the respective basic mechanisms can be applied. This is an important argument speaking in favour of the construction of the first prototype. It is not sure, however, whether the integration requirements can be combined with the time performance requirements, as the latter are rather vague. There are many possibilities to increase system performance. With the first prototype being available, performance measurements and load simulations can be carried out, on the basis of which conclusions can be drawn with regard to performance limits and short-comings. Then, it can be decided on special measures to be taken for increasing the performance.

7.2 Use of tools

Development of RODOS integrated data management practically represents the new development of the basic components of a heterogeneous distributed multidatabase system. Even if mainly functional system requirements have been specified up to date, maximum performance and, hence, soft real-time requirements remain. Adequate software tools are required to find a satisfactory solution for all partners within a period of 1.5 person-years. It must be noted that the use of tools for the first prototype does not necessarily mean that they are also used in the entire project. In the first prototype, the actually required functionality of integrated data management in RODOS should be determined. If necessary, this functionality can be post-implemented for RODOS use with special attention being paid to performance aspects. Type and scope of the tools used will influence the performance of integrated data management considerably.

In this chapter, the tools shall be presented briefly in accordance with the descriptions and specifications of the manufacturers. When the boundary conditions (in particular financial) for the type and scope of the use of tools will have been specified in further detail, their performance properties will have to be investigated for selection.

As already mentioned in previous chapters, the use of software tools is reasonable for the realization of the following tasks: integration of heterogeneous database management systems (RO-DIM), development of graphic user interfaces and knowledge management.

7.2.1 Functionality of the tools

Generally, two groups of software tools can be distinguished. The first group includes pure development tools, such as compilers, development environments, etc. They have already been presented briefly in chapter 2 such that the present chapter shall be restricted to the tools of the second group. This group is made up of tools providing a certain additional functionality e.g. in the form of function libraries. Their functionality needed may be divided into various fields according to the previous chapters. These are:

- General services and communication
  This includes above all function libraries offering standard interfaces for basic operating system services, such as file and storage management, timer functions, etc. Mechanisms for communication in networks e.g. in the form of libraries for the use of socket links and user authorization are added. Commercial and freely available libraries exist for the services mentioned.
- DBS link
  This includes services allowing a standardized access to heterogeneous relational DBMS (drivers). Examples of an access interface are partial sets of Dynamic SQL or in general a common SQL dialect, class libraries offering an object-oriented view of relational data, etc. (see chapter 3). Some tools in the form of class and function libraries have already been made commercially available. A standardized partial set of Dynamic SQL may also be applied.
- DBS integration
  This function comprises the central RO-DIM level. It makes available transactions via distributed database systems, two-phase commit, global schema management and integration. In this field, known products hardly exist. Two commercial products cover some parts of the total functionality required.
- GUI development and DBS link
  These are tools for the development of the simple graphic RODOS applications mentioned in the previous chapter. For the development of graphic interfaces, a multitude of commercial products exist, ranging from simple interface generators to complex development environments. A compromise solution in the form of a freely available package, such as the Tcl/Tk presented in the previous chapter, would be suitable. Another tool is the DBS link of a generated interface. It is the link between the graphic interface and integrated data management

below (RO-DIM) and the prerequisite for the development of RODOS applications. This aspect is covered by some of the commercial products only. It is not contained in Tcl/Tk.

- Expert system shell
  For the development of knowledge-based RODOS applications, an expert system shell shall be employed (chapter 5). In such a shell, the rule sets required for RODOS can be managed such that data in the form of facts only have to occur in the actual RODOS databases. Commercial and freely available tools exist for this purpose.

7.2.2 Short description of some tools

In this chapter, some commercial tools that seem to be suitable for use in this project shall be described briefly. As a basis, talks with the manufacturers and leaflets have been used. Therefore, further evaluation is required prior to the decision on their use. The tools presented here are only a part of the products studied by FZI. As a detailed market analysis is not the objective of this study, some systems of particular interest only are listed.

While the results achieved using tools of the above-mentioned first group are mostly free of deployment royalties, this often is not the case for the tools of the second group. However, package licences can be obtained.

7.2.2.1 JAM^6

The JAM^6 tool of the Jyacc company is a development environment for client/server database applications. A major constituent is the user-friendly and intuitively operable interface generation and its link with the database interface. Via drag-and-drop, the latter allows to integrate database objects from the system's visual object repository in its own applications. Integration of distributed databases cannot be performed directly. The interface to the database link is defined in the form of a SQL generator. A standard SQL dialect is made available by a C library for the supporting DBMS.

Evaluation:
DBS link acceptable, DBS integration not, GUI environment good

Prices:
it is an advantage of JAM^6 that applications created with it are free of deployment royalties. At the moment, only the prices for commercial use are known. They amount to about TDM 15.- for a minimum of five users. In this case, user means developer only, as no fees have to be paid for the ready applications. Discounts for non-commercial use in the scientific field may be possible.

Systems:
the system is portable with regard to both the existing database interfaces (more than 20) and the supporting operating systems (many Unix derivates and others).

7.2.2.2 OpenInterface

OpenInterface of Nexeus/Neuron Data can be compared with Jam[6]. It supports the developer in the form of extensive libraries with interfaces to databases and integrates user interfaces developed by the user. The entire system is conceived in a very open manner to allow interference.

Evaluation:
DBS link above average, DBS integration not, GUI environment good

Prices:
under certain circumstances, no deployment royalties or lump sums are charged for non-commercial, scientific applications. The commercial prices of one-person development licences amount to about TDM 30.-. Good terms are offered for non-commercial, scientific applications, in particular discounts of 40 - 50% for the delivery of a higher number of systems.

Systems:
he major operating systems (SunOS/Solaris, HP-Ux, IBM-Aix, SGI-Iris and PC systems) as well as large DBMS are supported.

7.2.2.3 Uniface-Six

Uniface-Six is a product of the Uniface company in the Netherlands. It represents an integrated environment for the realization of technology-independent database applications. Functionality of this product is based on three major concepts: model-based development, ANSI/SPARC 3-level schema architecture and extended client/server architecture. Thus, new applications accessing heterogeneous data can be implemented rapidly and efficiently. The tools support the generation of graphic user interfaces. The client/server architecture is based on the 2 PC log.

Evaluation:
DBS link good, DBS integration partly, GUI environment above average

Prices:
Uniface-Six is licenced. Besides development licences, deployment licences are needed. The first offer submitted amounts to TDM 138.- for 100 computers including all drivers. The prices can be negotiated.

Systems:
the system offers database drivers for 17 common database systems on several operating systems.

7.2.2.4 UniSQL

The UniSQL/M product of the UniSQL company represents a modern high-performance system for accessing distributed database systems. It was generated in 1992 by transferring research results to industrial practice. The system already realizes parts of a multidatabase system according to the 5-level schema architecture presented in chapter 3. Integration of partial database systems is achieved by an object-oriented view. In addition, the system can manage its own object structures. It therefore belongs to the class of promising object-relational systems. On its basis, the most complete version - of RO-DIM at least - may be implemented. Products e.g. in the form of a graphic interface generation, which can be connected easily with data management, are added.

`Evaluation:`
DBS link good, DBS integration far above average, GUI environment above average

`Prices:`
for non-commercial use at university and in research, prices in the order of $ 15,000.- to 20,000.- have been fixed for 100 users in Europe. The prices can be negotiated.

`Systems:`
all common Unix systems, such as SunOS/Solaris, HP-Ux, IBM-Aix, SGI-Iris, OSF/1 and PC systems as Windows-client and NT server. The relational DBMS Oracle, Sybase, Ingres and Rdb, the object-oriented Versant systems as well as UniSQL/X contained in the scope of services are supported directly. A generic driver generator for relational DBMS is added.

7.2.2.5 Other products

Orbix of IONA represents an efficient CORBA implementation and, hence, supports communication and distributed objects. It may be employed for the implementation of the global data dictionary. Similar libraries of other manufacturers also exist.

Db.h++ of Rogue Wave is a C++ class library, by means of which heterogeneous relational DBMS with an object-oriented view can be integrated. Approaches are offered for distributed transactions, provided that these are supported by the basic systems. Of all class libraries of this type, this product seems to be very complex.

Finally it must be pointed out that none of the software tools analyzed meets all requirements made in the RODOS project. A large part of the functionality required has to be implemented separately. Strongly integrating systems, such as UniSQL and Uniface, are expected to provide the most complex solutions, but hardly offer complex opportunities of interference e.g. for increasing the performance. In contrast to this, far smaller solutions are expected from RO-DIM _assembly_ of the first prototype using components that have partly been purchased and partly been developed on our own. Of course, further interference in all levels of the entire system may be possible. As the first prototype shall allow above all the recognition of the actual functionality required and short-comings, use of the complex integration tools is recommended.

7.2.3 Economic aspects

When implementing a complex information system, two alternatives are available: use of common software tools or own implementation of the complete functionality.

Due to the special requirements of the RODOS project, it is nearly impossible to find tools, which provide the total functionality required. In any case, part of the functionality and the adaptation to the particular situation have to be supplied by FZI. The advantage of this solution is the costs. The purchasing price for all tools required by FZI amounts to about TDM 150.- europeanwide. The corresponding offers have already been submitted to FZI. This price includes both the respective development licences as well as deployment licences for 100 computers in Europe. Software maintenance costs (including hot line and software updates) amount to about 15% of the purchase price. A maintenance contract is of crucial importance for an information system with various hardware and software platforms and frequent software updates (among others new versions of operating and database management systems) being expected. The total costs of this solution are far smaller compared to the implementation and maintenance by FZI (not possible) or a software house over the entire service life.

The alternative consists in the own implementation of the entire system. The advantage of this solution is that the users are independent of (one or several) institutions offering software and that the programmes can be modified in any way with modifications in the source code also being possible. The drawback consists in the high maintenance and support costs (and the above-mentioned limitations).

Consequently, the use of software tools to the largest possible extent is recommended by FZI.

7.3 Ideas for increasing the performance

Performance of the RODOS system as a distributed emergency system can be increased at many points. Some ideas shall be outlined below:

- Introduction of the process concept
  Specification of the decision process, i.e. the sequence in which application programmes and in particular data generators and consumers are started, allows to reduce the number of database accesses (no more _empty accesses_), accelerate decision-finding (_just in time_ queries) and increase the decision quality (always the latest data).
- Global query optimization
  One of the most efficient methods of increasing the performance is global query optimization. For this, statistical data on the data stocks (number of tuples per relation, probable number of scores per query) as well as the actual technical parameters of the hardware (actual transfer

rate in the network, size of the shared memory, etc.) are needed. For global query optimization algorithms known from parallel database systems can be applied.

- Last balancing in workstation clusters
  Automatic last balancing in workstation clusters prevents short-comings taking into account the actual load by application programmes.
- Storage of queries as stored procedures
  Additional increases in performance can be achieved by the storage of frequent queries as stored procedures in the database systems, provided that this concept is supported by the database systems.

This list does not claim to be complete. It will be extended as soon as first experience will have been gained from the prototype.

7.4 Future activities

FZI suggests to divide the development of the first prototype into several working steps. According to the working programme, 2 members of the FZI scientific staff will be involved in the project. Adherence to the schedule can only be ensured by the extensive use of software tools.

7.4.1 FZI working programme

- 1 October 1994 **start of project**
  selection (evaluation and test installation)
  of the tools for database integration,
  GUI generation and of an expert system shell;
  consideration of additional requirements of
  system specification in cooperation with D.T.I.
  detailed planning of project - detailed estimation of expenditure;
- 15 January 1995 **1st milestone**
- 30 April 1995 **2nd milestone**
  demonstration of the knowledge management component,
  the measured data management component and the GUI
- 31 August 1995 **3rd milestone**
  demonstration of the database integration component
- 31 August 1995 **start of test phase**
  evaluation of the prototype
- 30 September 1995 **end of project**
- after
  30 September 1995 **planning of the second prototype**

•

7.4.2 working programme (proposal)

- 30 November 1994 preparation of system specification
  (Which and how many computers with which
  capacities are expected?
  Which DBMS shall be applied?
  Contents, type, place of the databases used
  (database schemas with descriptions,
  amount of data, test data);
  RODOS applications involved;
  type and scope of DB operations and DB options,
  if necessary, transfer and transaction volumes;
  time specifications;
  detailed descriptions and specification ofmeasured data and knowledge management)
- 30 March 1995 technical aspects, such as access paths, computer names,
  network specifications, etc.;
  integration functions;
  development of the evaluation scenario
- until 30 September 1995 project management, consulting

Chapter 8 Summary

8.1 Project definition

The RODOS project is aimed at designing and implementing an integrated information system for use following the release of radioactivity. This system is applied supra-regionally for the coordination of protective measures in a network of heterogeneous computers (heterogeneous computers, software tools and database management systems).

The present study includes a feasibility analysis of integrated data management, which may be implemented in spite of the extremely high complexity of the entire system.

8.2 Special features

- Due to its planned field of application, RODOS will have to meet a number of novel requirements:
- Integration of heterogeneous data
  The RODOS integrated information system manages a number of data stocks. As most of these data already exist, the data schemas can no longer be modified. Other data are especially designed and implemented. The integration and transparency principle applies to all data. Both the users and the application programmes address the data by using a common data schema. There are no differences when accessing local or removed data.
- Support in an emergency
  The emergency, i.e. a situation where measures for the protection of the population have to be initiated and coordinated, has to be handled in a special way. The queries of deciding authorities are processed with priority, non-critical operations are rejected.
- Soft real-time requirement
  In an emergency, the deciding authorities shall be provided with information on the current situation every ten minutes. This soft real-time requirement shall be met by RODOS, even if the information desired is located on several distant computers.
- Measured-data management
  Measured data describe the current parameters of the environment. They serve as arguments in

decisions. Measured data are characterized by high amounts and special access patterns. This type of data shall be managed in accordance with the requirements.

- Knowledge management
  To initiate decisions for the implementation of protective measures in a europeanwide consistent manner, global management of the facts only is insufficient. In addition, common procedure regulations, i.e. the knowledge as to when a measure is initiated, are needed. This knowledge also shall be managed in the common data stocks.

8.3 Doubts

Several general, technical and organizational doubts were discussed in chapter 7.1. Due to the vague specification of important technical parameters, such as the size of the data stocks or data transfer volume in an emergency, some of them cannot be dispelled. In this situation, they can only be verified by tests of the RODOS prototype. The first prototype, which will be installed on a workstation with a processor, may exhibit an insufficient performance. In case of short-comings, the problems may be solved by increasing the hardware capacities.

8.4 Future activities

The present study has shown that the functionality required for the RODOS system can be implemented. To realize the first prototype in the planned time, software tools have to be applied. The advantage of the use of tools does not only consist in a reduction of the prototype implementation time, but also in a decrease of the maintenance costs and a simple portability of the RODOS system to new hardware and software platforms.

The future activities were presented in chapter 7.4. To achieve the best possible results, the requirements to be met by the system have to be specified very precisely. At the same time, tool evaluation and selection can be started. Implementation takes place upon the completion of this phase only. The supply and testing of the first prototype represent an important step towards the realization of the system.

Chapter 9

References

Proceedings

[BTW93] Herausgeber: W. Stucky, A. Oberweis: „Datenbanksysteme in B_uro, Technik und Wissenschaft", Braunschweig, Deutschland, 1993, Tagungsband

[ColS94] Herausgeber: M. Brodie, M. Jarke, M. Papazoglou: „Proc. of the 2nd Int. Conf. on Cooperative Information Systems", Toronto, Canada, 1994, Tagungsband

[IvU93] Herausgeber: R. Denzer, W. Geiger, R. Güttler: „1. Workshop Integration von Umweltdaten", Dagstuhl, Deutschland, 1993, Tagungsband

Articles, Books

[AbGa92] R.K. Abbott, H. Garcia-Molina: „Scheduling Real-Time Transactions", in ACM Transactions on Database Systems, Vol. 17(2), 1992, pp. 513

[ACM90] ACM Computing Surveys 22: „Special Issue on Heterogeneous Databases", Vol. 22(3), ACM Press, 1990

[BCLM94] M. Bauer, N. Coburn, P.A. Larson, P. Martin: „Managing Global Information in the CORDS Multidatabase System", in [ColS94], pp. 23

[BeGr92] D. Bell, J. Grimson: „Distributed Database Systems", Addison-Wesley, 1992

[BLN86] C. Batini, M. Lengereni, S.B. Navathe: „A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computing Surveys, Vol. 18(4), 1986, pp. 323

[BrToRo94] B. Bruegge, K. O'Toole, D. Rothenberger: „Design Considerations for an Accident Management System", in [ColS94], pp. 90

[Date90] C.J. Date: „An Introduction to Database Systems", Vol. I, Addison-Wesley, 1990

[DSS93] Herausgeber: F. Redmill, T. Anderson: „Directions in Safety-critical Systems", Bristol, UK, 1993

[DTI94] Dr. Trippe Ing. Büro (M. Rafat): „RODOS", DTI, Karlsruhe, 1994

[Fahl94] G. Fahl: „Object Views of Relational Data in Multidatabase Systems", Licentiate Thesis, Universit at Link oping, Schweden, 1994

[FrLa93] J. Frohn, G. Lausen: „Integration heterogener relationaler Datenbankschemata mittels eines objektorientierten Datenmodells“, in [BTW93], pp. 285

[FZI93a] Herausgeber: E. Casais, C. Lewerentz: „Getting Started with OBST - System Documentation“, FZI Karlsruhe, Deutschland, 1993

[HeSc93] I. Henning, F. Schmidt: „Integration von Daten und Methoden im Umweltinformationssys. des Landes Baden-Württemberg“, in [IvU93], pp. 75

[Heue92] A. Heuer: „Objektorientierte Datenbanken“, Addison-Wesley Deutschland, 1992

[HoK_o93] U. Hohenstein, C. Körner: „Object-Oriented Access to Relational Database Systems“, in [BTW93], pp. 246

[HSJHK94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, J. Kusch: „Revised Version of the Modelling Language TROLL Version 2.0“, TU Braunschweig, Deutschland, 1994

[Iona93] IONA: „The Orbix Architecture“, IONA Technologies Ltd., 1993

[Kent89] W. Kent: „The many forms of a single fact“, in proc. IEEE Compcon 89, San Francisco, USA, 1989, pp. 27

[Kram92] R. Kramer: „Steuerung der Anfragebearbeitung in parallelen Datenbanksystemen“, Dissertation, FZI Karlsruhe, Deutschland, 1992

[MeSi93 J. Melton, A.R. Simon: „Understanding the new SQL: a complete guide“, Morgan Kaufmann Publishers, 1993

[MetAl94] P.E. Mantey et al.: „REINAS: Real-Time Environmental Information Network and Analysis System“, University of California, USA, 1994

[Miso93] Microsoft: „Open Database Connectivity“, Microsoft Press, 1993

[OMG92] Object Management Group: „Object Management Architecture Guide“, Object Management Group, 1992

[Oust94] John K. Ousterhout: „Tcl and the Tk Toolkit“, Addison-Wesley, 1994

[PRR91] W. Perrizo, J. Rajkumar, P. Ram: „HYDRO: A Heterogeneous Distributed Database System“, in ACM SIGMOD, North Dakota, USA, 1991, pp. 32

[RaSc94] E. Radeke, M. Scholl: „Federation and Stepwise Reduction of Database Systems“, in proc. Applications of Databases, Link_oping, Schweden, 1994, pp. 381

[SADLO94] M. Stonebraker, P.M. Aoki, R. Devine, W. Litwin, M. Olson: „Mariposa: A New Architecture for Distributed Data“, proc. IEEE Conference on Data Engineering, 1994, pp. 54

[Sch_u93] T. Schütz: „Der Umwelt-Datenkatalog Niedersachsen“, in [IvU93], pp. 20

[TaVa91] M. Tamer Özsu, P. Valduriez: „Principles of Distributed Database Systems“, Prentice-Hall, 1991