Towards the Realization of the Database Designer's Apprentice
Part 1: General Introduction

by
Philippe Mathieu
Towards the Realization of the Database Designer's Apprentice

Part 1: General Introduction

by

Philippe Mathieu

Swedish Institute of Computer Science
P. O. Box 1263, S-164 28 Kista, Sweden
Electronic mail: philippe@sics.se

Contents

1. Introduction ............................................................................................................. 1
   A. Outline of the Project ........................................................................................... 3
      Goal and Objectives ............................................................................................... 3
      A Knowledge-Based System Approach ................................................................. 5
   B. Current State of the Art ....................................................................................... 7
      Introduction ............................................................................................................ 7

© SICS, 1987

Parts 2, 3 and 4 will focus on the three main parts of the Designer's Apprentice: Pattern Analysis, Pattern Synthesis and the verification module.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capturing User Requirements</td>
<td>8</td>
</tr>
<tr>
<td>Conceptual Design</td>
<td>10</td>
</tr>
<tr>
<td>Distribution Design</td>
<td>15</td>
</tr>
<tr>
<td>Logical and Physical Design</td>
<td>18</td>
</tr>
<tr>
<td>Expert System Technology</td>
<td>19</td>
</tr>
<tr>
<td>II. The Project</td>
<td>21</td>
</tr>
<tr>
<td>A. Description</td>
<td>21</td>
</tr>
<tr>
<td>B. Work Plan</td>
<td>22</td>
</tr>
<tr>
<td>Architecture</td>
<td>22</td>
</tr>
<tr>
<td>User Interface</td>
<td>25</td>
</tr>
<tr>
<td>Distribution Design</td>
<td>26</td>
</tr>
<tr>
<td>Existing Tools at the High level Specification</td>
<td>26</td>
</tr>
<tr>
<td>References</td>
<td>27</td>
</tr>
</tbody>
</table>
Towards the Realization of the Database Designer's Apprentice

Part 1: General Introduction

by

Philippe Mathieu
Swedish Institute of Computer Science

I. Introduction
The project presented here is part of the FREYR project\(^6\). Its purpose is to explore the possibilities of applying knowledge-based technologies to the design of databases.

\(^6\) In fact it is part of the SICS-KBS-Lab project "Deductive databases with learning capabilities" a subproject of FREYR.
It is divided into two main parts. The first one presents a state of the art, which should serve as a collection of results to be possibly used in the design of the project. Part of this text (essentially the state of the art) is taken from the description of an ESPRIT project in which the author was involved during the academic year 1984-85. It has been completed to take into account the recent developments in the field.

The second part contains the description of the project. The research proposal comprises as a main theme the use of the so-called designer’s apprentice. The possibility to use such a pastiche of the well-known programmer’s apprentice in conceptual modelling appeared in a 85 report of the author. Interestingly enough, though the programmer’s apprentice seems to fall short of expectation, quite the same technique applies to the conceptual design with more success. Roughly, the reason for this may be that programming (as a coding activity) has not yet found its geometry, while design drawing lends itself to formal description in the theory of graphs. Problems that we are able to connect with a graph become simpler to discuss and large sections of verbal description are replaced by manipulation with pictures; graphs are as close as possible both to their visual image as a whole and to their description using formal rules.

---


2. What follows will be explained in full details elsewhere.

3. There is a recent attempt (Girard).
A. **Outline of the Project**

**Goal and Objectives**

**Summary**
One of the problems which the industry is faced with, though the huge development of the data-processing equipment, is designing databases, both centralized and (possibly logically) distributed. And it is a fact that the task of designing databases is not sufficiently supported by current tools.

The goal of this project is to apply knowledge-based system technology to the design of databases. As regards this, the artificial intelligence paradigm is very fruitful, and the application of knowledge-based systems to specific domains has been very promising.

The objective of the project is to make a prototype of an knowledge-based system for designing generalized tools that cover all the phases of the database design process, namely capturing user requirements, conceptual design, distribution design, logical design, and possibly physical design.

The architecture of the system will consist of an inference engine, a knowledge base for design rules, and a database for facts concerning the design of a specific database. Moreover - and this is one of the key themata of the whole project - all the system will be supported by a (dynamic) learning module. It is intended that this module will continue to serve as a learning support during the all life of the database. The design rules will be extracted from the existing or newly developed methodologies for conceptual, distribution, logical, and physical design. The objective of the design tool is that it will guide the user in an interactive way through the design process, assist him by e. g. allowing what-if type of questions, and
automate parts of the design process.

The following quite obvious observation may be regarded as the starting point of the project: The design process should not be fully automated but rather just supported by a cooperating interface between a human designer and intelligent computerized systems.

The fourth generation of specification languages contributes to the advances in database design; and a basic understanding of the design problems has been reached, for instance, with reference to the distinction between the conceptual, logical, and physical design, to the convenience of using database design methodologies, and to the need of the automatic tools for assisting the designer. However, the practical use of these tools remains as a difficult art to get under control. In consequence, the interesting results of attractive theories such as relational databases (dependencies, normalization) or the entity-relationship approach to database design are still the concern of a few experts in the area.

**Objectives**

To make the knowledge currently available in design methodologies, available to designers of databases a knowledge-based approach is proposed. The project should specify and implement knowledge-based tools (e.g. expert syste.ms) for supporting the design of the database schemas and the applications that are centered around a database management system. One of the proposed expert systems should assist either a database designer, or a database administrator, or a database programmer in the following important phases:
1. In capturing the user requirements of data and operations by providing uniform and user-friendly interfaces to the database (e.g. screen formatting, report generation, menu handling facilities);

2. In designing a conceptual schema for both the data and the transactions, by integrating the views of the different users, normalizing the relations based on the dependencies, and specifying allowable sequences based on dynamic integrity constraints;

3. In designing conceptual schema for a distributed database - fragmentation of data, decomposition of transactions, and the allocation of data and transactions have to be determined;

4. In designing logical and physical schemas.

A word of caution: The old subdivision between conceptual, logical, and physical schemas will be maintained in this introduction (as well as in the state of the art), but our subdivision is different and will be explained when describing the project (part II)^2.

**A Knowledge-Based System Approach**

It is proposed to build up a knowledge-based system in three stages.

**First Stage**

The first stage provides the part of the knowledge base that contains general knowledge about methodologies, the models and the interfaces used, and all

---

the general expertise we have in database and transaction design (e.g. semantic and relational model, transaction model, cost model\(^3\)). This part acts both as a documentation database and as a collection of rules describing the design process.

**Second Stage**
The second stage provides the part of the knowledge base that contains semantic knowledge about the requirements common to a certain class of database applications (e.g. banking systems, library applications, office automation).

**Third Stage**
The third stage provides specific knowledge of a given application. It corresponds to specific expertise that a human being may have in a specific domain of application. We may distinguish three subparts of this knowledge according to the design process:

1. User requirements;
2. Conceptual, logical and distribution design information;
3. Physical design parameters.

This part may be viewed as some kind of intelligent data dictionary that will point out the consequences of any change in the data or the constraint definitions for existing application programs.

\(^3\)Mathieu P.: The Cost of a Query, preprint, 1987
B. **Current State of the Art**

**Introduction**
The idea of an automatic aid for databases design and programming is now widespread, mainly because of the growing complexity in structuring large amount of information and of the tedious task of specifying transactions between the data directly in inadequate programming languages. Historically, designers have distinguished between static (i.e., data) and dynamic (i.e., program) aspects. They have approached the two aspects in different ways and have provided appropriate models and tools for each aspect. Different levels of design have been suggested (capturing user requirements, conceptual design, distribution design, logical design, physical design). These abstract levels are intended to make the design methods as independent as possible either of a special software and hardware support or of any specific domain of application.

Remark: By static aspects we mean data structures (objects and relationships) and some static integrity constraints (domains, length, unicity, referencial). Dynamic aspects intend to describe the actions operating on these data structures and the successive transactions modifying the information system from one state to another. This is also referred to as behavioral semantics or transaction (or processing) modelling.

The following subsections discuss methods and tools for the different design levels both for the static and dynamic aspect.

---

\[\text{In this State of the Art, we just refer to the research papers that have contributed to the field with major ideas.}\]
Capturing User Requirements

Designing an information system and implementing it within a database management system requires first of all a correct description of the general organization of the system to be modelled.

The satisfaction of the users of the future system depends on how accurately the real world is perceived and consequently on the discovery of common grounds between users and designers. The progress made in natural language understanding can help to discover this common ground in two ways:

1. The analysis and the comprehension of simple sentences describing the users' perception, which enables them to participate fully in the development of the information system;

2. The generation of natural language expressions by the system enables control and validation at each stage of the design.

A comparative study of different projects on natural language understanding reveals a diversity of solutions according to the goal assigned to the systems with respect to man-machine interface. Most works on natural language interfaces with databases has dealt with retrieval (SRI system TEAM\textsuperscript{4}, ...); and just some systems are available that support natural language database queries, for example by transforming a (quasi-)natural language query into a formal language like SQL.

Works is now underway on updates to a structured database via natural language (Wiederhold at Stanford University). And there are several projects dealing with deriving "structured" knowledge bases from full text databases (in

\textsuperscript{4}Hendrix G., W. Lewis: Transportable Natural Language Interfaces to Databases, SRI-Al Center, 1981}
natural language).

There is also a growing interest in designing data models that bear a close correspondence to the meaning expressed in a natural language sentence with the fewest possible links with the database technology.

TAXIS\textsuperscript{5} is a good example of the integration of AI and database techniques to facilitate the development of large interactive information systems. TAXIS involved a notion of scripts and tools allowing the definition of user interface languages (vocabulary, syntax, semantics, pragmatics). This point is important because one major part of the designer's work is to make clear to the users the precise definition of the data they are handling keeping in mind the different points of view of the several users. A linguistic analysis of these definitions would assist in the development of a coherent representation of these different views. Moreover, the problem of designing a database can be linked to learning systems because the designer's method also involves generalization, induction, and analogy mechanisms. Such systems are able to derive from examples (or experience) some knowledge which enables them to treat similar phenomena. And there are systems (e.g., the TEAM system of loc. cit.) that apply a natural language understanding component to some learning mechanisms; these could be used in the definition of data for the dictionary and for the database conceptual schema.

Conceptual Design

Static Aspects
Several methods and tools have been developed. Generally, these methods use one or several of the following four categories of tools: algorithmic tools, manual tools, interactive tools, and intelligent tools.

Algorithmic Tools
These tools are usually built around the relational model; they provide programs deriving a normalized relational schema from a set of attributes and constraints. We shall distinguish two main types of algorithms.

- In the first case, relations are automatically deduced from attributes and functional dependencies.\(^6\)

- In the second case, relations are subjectively given by the designer and then tested and possibly decomposed if they present some dependencies.\(^7\) There exist several other algorithms concerned with key searching, null suppressing, and so on.

This last category is one of the most promising in the area of database design. Some theoretical results are already available, but practical use of these tools presents some difficulties. First and with respect of the designer, the theory is not always obvious. Collecting functional and


\(^7\) Fagin R.: Multivalued Dependencies and a New Normal Form for Relational Databases, ACM TODS, vol. 2, 1977
multivalued dependencies is somewhat tedious when exceeding about 100 attributes. In addition functional and multivalued dependencies are not always sufficient to capture all the semantics of a real application. Finally, some change in the application (e.g. adding or removing some constraint or attribute) often leads to the redesign of the database, a task which is of course time and money consuming.

Second, with respect to the relational model itself, there does not exist a general algorithm which takes into account all the constraints (functional and multivalued dependencies) and which handles all the design steps (constraints and attribute acquisition, normalization, finding keys, eliminating null values or functional dependency cycles). Among several algorithms which actually exist and handle some particular cases, some of them have acceptable response time, others are not usable because of their exponential complexity.

As regards the search for dependencies, it should be pointed out that some work has been done in order to determine such dependencies dynamically\(^8\)\(^9\). This attempt to add learning capabilities to databases continues to be a field of interest \(^10\).

Manual Tools
These tools coincide essentially with semantic models which intend to

---

\(^8\)Mathieu P.: Note sur l'apprentissage de dépendances fonctionnelles et questions connexes, INRIA, Tech. Note, 1984

\(^9\)Mathieu P.: Une méthode d'apprentissage des dépendances fonctionnelles dans les bases de données déductives, Proc. of the First Spanish Meeting on AI and Databases, 1985

directly represent the user's perception of the real application, like the entity-relationship model, the semantic hierarchy model, and the semantic data model. These tools suppose that someone is able to capture directly from the application all the concepts of the model used (entities, relations, keys, functional dependencies, ...): the design choices are left to the so-called database administrator; only a few guide for validating the design are offered to him. But it is clear that some tools should be added to help using these models.

Interactive Tools
One of the first automated tool which have been developed is the data dictionary. This tool acts as a memory aid to remind the designer of the choice he has made. Sometimes it is associated with a few programs which carry out the syntactic parsing of the information list or the conceptual schema of the database but can neither guarantee its consistency nor its relevance with respect to the application described.

The second class of tools which has been proposed are those which call for CAD/CAM techniques. These tools offer a set of interactive programs which permit the user to shift the responsibility of some tedious and complex tasks on the program. Interaction between the user and the automated tool is "question-answering" oriented or "graphics-language" oriented. The drawbacks of such tools are numerous; in particular, they are not evolvable: changes in the design often implies reprogramming the tool.

The third class of tools are specification languages. Among the most significant contributions in this area we mention PSL/PSA, SADT, and TAXIS, with their derived or complementary products. The general characteristic of there tools is that they handle both static and dynamic aspects of information systems.
We discuss here the static aspect; for the dynamic aspect see below the section on dynamic aspects. The PSL/PSA computerized aid tools are composed of the so-called problem statement language (PSL) which permits descriptions in a syntactically analyzable form system structure, data structure, system size, volume, ..., and a problem statement analyzer (PSA) which examines the previous description to produce documentation reports such as list of information with their types, properties, and relationships of all successive updates, graphics showing some hierarchies between the system components or some relationships between the system components or some relationships between data, ... All this information is stored in a data dictionary under the control of a DBMS.

The TAXIS specification language also intends to offer computerized tools to describe the semantic components of an information system. Its original contribution is to combine, when building these tools, concepts, and techniques coming from AI (semantic networks), databases (relational model), and programming language (ADT's). TAXIS provides some powerful concepts (like is_a hierarchies) and a design methodology based on a stepwise specialization as compared to stepwise refinement by decomposition in programming. A specific conceptual database design tool (CSDL) and a high level user interface are also provided.
Dynamic Aspects
We point out three categories.

Description Models
These are tools using Petri nets\textsuperscript{11} or derived representations that may look rather different from Petri nets but which are nevertheless heavily influenced.

ADT's
This category of tools can be illustrated by the extended semantic hierarchy model of Brodie\textsuperscript{12}. The behaviour of an object is completely defined by its actions which are the only way of retrieving or modifying this object. An action is the combination of one of the three behavioural abstractions (sequence, choice, repetition) with database operations (retrieve, insert, update, delete) controlled by a pre-condition and a post-condition. A transaction is modelled as a set of actions altering one or more objects.

Specification Languages
PSL/PSA offers some documentation which reports the dynamic properties of the system. These properties are included in "processes" and "procedures" which are specified with their inputs and outputs, their triggering events and the structural relationships they have with other processes.

With the same basic constructs which permit to structure data (tokens,

\textsuperscript{11}De Antonellis V., B. Zonta: Modelling Events in Database Applications Design, Proc. of the 7th VLDB, 1981

classes, metaclasses), TAXIS permits to structure operations on these data. Transactions are declared as classes belonging to the special metaclass "TRANSACTION_CLASS". For each transaction, we may specify its inputs (parameter list) and outputs (returns), a pre-condition (prerequisite) and actions (database operations and arithmetic expressions). A compiler analyzes the specification and generates transactions in PASCAL.

SADT is an original approach based on graphics and menus. Data schemas and program schemas are represented by an hierarchy of boxes interrelated by a unique box, each side representing respectively inputs, outputs, control data, and the processor which can handle this activity. The hierarchy of boxes represents the decomposition of the problem into different level of abstraction. The arrows relating boxes represent interfaces or constraints between boxes. Each box is decomposed, if necessary, into a fixed number of boxes. From these diagrammatic specifications, SADT maintains a data dictionary and generates documentation and management reports.

**Distribution Design**

**Static Aspects**
Two approaches exist.

**Top-down Approach**
This approach is typical for a distributed database system which is developed from scratch. With this approach, an integrated database schema which models all the data of the distributed application is initially
designed; the schema is then partitioned into several subschemas, one at each database site, and the data are distributed accordingly. In particular, global objects (relations) belonging to the integrated schema are decomposed into fragments, and fragments are then assigned to the subschemas at the different sites. Fragmentation is an important aspect of distributed databases, because it allows the user to write programs which use global data objects and are transparent to data distribution and fragmentation, while the system controls the placement in the most efficient way. The top-down distribution design is therefore decomposed into two phases:

1. Design of fragmentation: The goal of this phase is to determine fragments as units of distribution, i.e., portions of global data objects which can conveniently be distributed. The design of fragmentation requires an understanding of the properties of global data objects and of the applications which use them. Therefore, the design of fragmentation can be regarded as the logical phase of the distribution design.

2. Allocation of fragments: The goal of this phase is to determine the sites where the fragments should be allocated, possibly by replacing fragments over multiple sites. Allocation of fragments aims at the optimization of the distributed database performance, and can therefore be considered as a physical decision.

Bottom-up Approach
This approach is typical of distributed databases which are developed as an aggregation of pre-existing databases. With this approach, schemas representing the portion of data stored at each individual site constitute the starting point, and distribution design consists in identifying the data which are common to the distinct schemas, and giving them a common
representation. The integration process should solve conflicts, i.e. different representations of the same data in different schemas.

Dynamic Aspects
The distribution of programs to the sites of the network has two different aspects, which should be kept distinct:

1. The decomposition aspect, which consists of decomposing the overall programs into smaller units which are not further decomposed and must reside each one at a single site;

2. The allocation aspect, which consists of determining the site of the network where each program unit is allocated.

Keeping this distinction in mind, the following two problems are studied:

1. Operation allocation problem: Assuming that the database decomposition and allocation and also the decomposition of applications into elementary operations are fixed, determine the optimal allocation of elementary operations;

2. Application decomposition and allocation problem: Assuming that the data decomposition and allocation are fixed, determine the optimal decomposition and allocation of applications.

Notice that in this approach to distribution, the design of the data is kept separable from the design of programs. There are several reasons why this separation is convenient. The first reason is that separation is simpler. The second one is that often the knowledge of optimization parameters at data design time is different from the knowledge of them at program design time. Usually, data design is done before program design and is based on
assumptions about applications that will run in the system. Applications are
designed later during the lifetime of the distributed database. In this case,
when an application is designed, the data design is already decomposed and
allocated and this aspect of the problem has to be considered fixed.

Logical and Physical Design

Static Aspects
This level is concerned with the physical organization of data. Its main
objective is to produce an efficient data structure, i.e. a data structure
which optimizes both database volume and transaction response time. The
logical design is concerned with the estimation of the amount of data,
transaction frequencies, and the global cost of the information system. The
physical design is concerned with the physical organization of data, the
choice of access paths, data clustering, ... The general purpose of these two
levels is to decide between several candidate data structures which has to be
implemented. Unfortunately, they are numerous and it is not conceivable to
test them altogether. Two types of methods are generally considered to
approach this problem: analytical methods which propose mathematical
models to evaluate data structures, and methods which proceed by
simulation to tune different data structures and to choose the best one.
Several activity models are proposed at the logical level for measuring or
predicting the activity of a database, depending on critical parameters
(transaction frequencies, transaction response time needed, number of
transactions , volume of data, I/O time, CPU time, ...). Generally, these
optimization models are built for hierarchical or networks databases. The
complexity of these activity models depends on the level of detail which we
are interested in. This level in turn depends on the knowledge we have
about the application.
**Dynamic Aspects**

This aspect is concerned with all tools that can contribute to an efficient production of programs.

Some works has been done that permit the user to declare his data views and his transactions in the same integrated language.

A substantial effort has been done to enhance database description capabilities with some dynamic constraints; this approach intends either to extend traditional integrity constraints control by adding some dynamic constraints (or triggers) or to extend the capabilities of schema description by adding virtual relations.

Another aspect is the mapping of the specifications of the conceptual level to the logical and physical levels: how do we produce efficient programs in a particular language from specifications given in TAXIS, Petri nets, ...

**Expert System Technology**

The expert system field emphasizes domain specific problem-solving strategies. The strength of expert systems is recognized to come from the ability to capture knowledge of people able to act efficiently in complex situations. Practitioners of a domain are able to act when using and dealing with different representations of a system to use, to understand, or to control. Knowledge-based system technology uses different ways to translate this human ability, e.g., production rules, which is a way to decompose the knowledge of the expert in chunks of knowledge translated in IF...THEN... rules where in general the THEN part represents an action. Other systems rely on semantic networks, which are an image of the manner in which the human mind through a meta-use of a natural language creates symbolic representations of the application. Moreover, in
expert system technology, we often have a language to translate the knowledge added to an inference engine.

Of course, one of the major problems is to express control structures that efficiently guide the use of a given (large) knowledge base. In Units, for example, this control is obtained via the interaction and the mixing of different ways of representing the knowledge; in other systems (e.g., Prospector), it is obtained by using semantic networks.
II. The Project

A. Description
The objective of the project is to specify a design tool for databases using knowledge-based system technology. The design tool will be used for capturing user requirements, conceptual design, distribution design, and logical and physical design. The design tool is constructed in a modular way such that changing the knowledge in the logical and physical design layer makes it possible to generate a design for a database management system with a different data model.

The whole design process will be supervised by a "designer's apprentice", a pastiche of the well-known programmer's apprentice. This learning component that is of great importance in any design - or programming - process, seems to have been underestimated in the past. In our project it has a major role from the very beginning of the design process, and it will continue to serve as long as the database is in use.

Each work package will go through more or less the same phases. First, a high level functional specification will be obtained. Second, by designing the interfaces between the different design layers in the previous phase, the final high level functional specification will be completed. Third, by developing methodologies for the different design layers, the functional specification will be made more specific, resulting in a low level functional specification.
B. **Work Plan**

Some of these tasks should be considered as part of the FREYR project to which the reader is referred for a complete description\(^a\).

**Architecture**

All the architecture is based both on a new approach to the division of the design levels and on an interpretation of the programmer's apprentice paradigm to the database design: what we call the designer's apprentice. First, it seems that the old conceptual level and the logical level should better merge into the **information and function level**; they give raise to a **representation level** that is coded into some physical support. To understand these concepts, just think about a structure described at the information and function level by a stack (a stack is indeed endowed with several functionalities); this type of structure can for example be represented - at the representation level - by an array or a list.

This very simple example generalizes to database theory. At the top level - the information and function level - we get an abstract data model that we would like to regard as an ADT (abstract data type). It could be a structured set of relations (resp. of trees, of graphs). At the representation level we get the usual database models: the relational model (resp. the hierarchical model, the network model). It should be clear that the possibility to incorporate functionalities in ADT's is an important tool to capture the semantic of an application.

To capture more semantics, we want to relax the quite involved theoretical

---

\(^a\) Especially for the learning component.
structure of the ADT's by considering in parallel other semantics models (such as the entity-relationship model). But this one should always be interfaced with an abstract data model.

The concept of a designer's apprentice aims to develop computer representation of applications and knowledge about these applications which can facilitate the interaction between the designer and the apprentice.

The system should be based on three forms of descriptions:

1. Definition of structured data objects, their parts, properties, and the relations between them. This is the previous semantics model.

2. Transactions. This is the abstract data model (ADT) with its functionalities.

3. A hierarchical representation of the internal structure of the application, that we call plans (as in the programmer's apprentice paradigm).

Plans include in particular the data and control flow relationship between parts of the application as well as logical dependencies. Plans should also be used for the compilation of a plan library of general knowledge about design processes in general.

As in the case of the programmer's apprentice, we would like to distinguish between two types of plans:

---

*a* Compare with the programming's apprentice paradigm.
1. The surface plans that give a description of the control and data flow between the parts of the program; they form the surface structure.

2. The deep plans that aim to explain how and why the application works; they help in understanding the very logical structure of the application, and constitute the deep structure.

In the simplest case, the deep structure could be a set of PROLOG clauses (deductive databases).

In the figure 1, the designer's apprentice is represented with its functionalities. We describe them hereafter in more details. First, all the functionalities are governed by the knowledge-based structure. Pattern analysis involves both the conceptual level and part of the logical level; the second part of the logical level is dealt with when doing pattern recognition or pattern synthesis (e.g., recognition of "hidden" functional dependencies). This phase may involve data analysis on a population of data out of the application. The verification function is needed to assure consistency. All the functionalities give a feedback to the knowledge-based structure resulting in a plan library.

The module "institution model"\textsuperscript{13, 14} has been introduced to permit to combine several abstract data models that share some common functionalities (semantics).

Of particular importance is the translation of the semantic model into the

\textsuperscript{13}Burstall R., J. Goguen: An Informal Introduction to Specifications Using CLEAR, in: Software Specification Techniques, Addison-Wesley, 1985
institution model. This include, e. g., the translation of Petri nets to the relational data model. One of the most important aspects to be translated is the several constraints\textsuperscript{8}. We argue that a rich grammar formalism should be here one of the key points\textsuperscript{15, 16, 17}.

\textit{User Interface}

The goal is here to cover the design of that part of the user interface to capture requirements by means of natural language and several formal languages (graphics, ...). Among the specifications, a definition will be given that will include a choice e. g. menu-driven interface, graphics interface, ... Here too, languages based on the ADT formalism should be defined\textsuperscript{16}.

When specifying the (quasi-)natural language interface, Winograd's paradigm\textsuperscript{19} should be envisaged as one of the most promising environment\textsuperscript{b}. Of course, some functions to be considered here are analysis and generation: the analysis and comprehension of sentences describing users' perception\textsuperscript{20}; the generation of natural language expressions by the system to enable control and validation at each design stage.

\textsuperscript{8} See Borkin's thesis.
\textsuperscript{15} Mathieu P.: ADT's, Non-Classical Grammars and AI Programming Languages for CAD/CAM (Introduction Only), Proc. of the NordDATA Conf., 1987
\textsuperscript{16} Mathieu P.: Attribute Grammars and CAD/CAM, Report for the ESPRIT project AMICE, 1987
\textsuperscript{17} Reps T.: Generating Language-Based Environments, The MIT Press, 1983
\textsuperscript{19} Winograd T.: Language as a Cognitive Process, Addison-Wesley, 1983
\textsuperscript{b} In this paradigm, the cognitive personality of the user influences on the choice of the interface grammar.
\textsuperscript{20} Popov E.: Talking with Computers in Natural Language, Springer, 1986
**Distribution Design**

We want to emphasize the logically distribution design. This is part of the well-known hybrid representation problem in knowledge base theory\(^8\). This view is important when the databases are queried. The institution model should serve as a tool for redistribution of the query.

**Existing Tools at the High level Specification**

For the description of the semantic model systems that are based on strong case grammars should be studied (for example KEE).

The institution model could be describe using one of the so-called "institution language" (for example CLEAR).

For the abstract data model a language such as ACT_ONE would be very useful. Note that it can serve as a specification language for the OSI communication system and is part of the LOTOS language, a language for describing Milner's CCS communication process. LOTOS should be considered as a language for the description of the interfaces between the several abstract data models.

---

\(^8\) See the description of the FREVR project.
References


Hendrix G., W. Lewis: Transportable Natural Language Interfaces to Databases, SRI-AI Center, 1981


Mathieu P.: Note sur l'apprentissage de dépendances fonctionnelles et questions connexes, INRIA, Tech. Note, 1984

Mathieu P.: Une méthode d'apprentissage des dépendances fonctionnelles dans les bases de données déductives, Proc. of the First Spanish Meeting on AI and Databases, 1985


Mathieu P.: ADT's, Non-Classical Grammars and AI Programming Languages for
CAD/CAM (Introduction Only), Proc. of the NordDATA Conf., 1987

Mathieu P.: Attribute Grammars and CAD/CAM, Report for the ESPRIT project AMICE, 1987

Mathieu P.: The Cost of a Query, preprint, 1987


Winograd T.: Language as a Cognitive Process, Addison-Wesley, 1983