YYY-A database design tool*

HARRI LAINE

University of Helsinki, Department of computer science
Teollisuuskatu 23, SF—00510 Helsinki, Finland

Abstract

YYY is an interactive, graphical tool set for designing databases. Presently it contains tools for the design of the enterprise (conceptual) schema and an expert system for generating the schema for the relational database. On the enterprise level the data model supported in YYY is a variant of the Entity — Relationship Model. A relational database schema is represented as SQL 'create table' — statements. This paper discusses the overall structure of the tool set, the data dictionary, the principles of the user interface and the rules that control the generation of the relational database schema.

1. Introduction

During its design the database is described in various levels of abstraction. Most researchers identify three levels. We call these levels the *enterprise level,* the *representation level* and the *internal level.* On the enterprise level (often called conceptual level [ElNa89]) the main concern is on *what* is described in the database, i.e. on how the object of the data is structured. On the representation level (called also logical level or implementation level [TeFr82, ElNa89]) we are concerned about the *logical data structures* that are used in representing the data. On the internal level the technical structures and access paths of the database are considered. As related to each level of abstraction, there is a schema that contains the description. Database design is a process of constructing a schema for each level of abstraction. A natural course of action is to proceed from the user oriented enterprise schema to the computer oriented internal schema. The intermediate representation schema is needed because most of the current database management systems rely on it.

Database design process is often divided into four steps: *requirements collection and analysis, conceptual design, data model mapping* (logical design) and *physical*

* Lecture presented at the 1st Finnish—Hungarian Workshop on Programming Languages and Software Tools, Szeged, Hungary, August 8—11, 1989.

design [TeFr82, ElNa89]. During the first step the needs of the users as related to the contents and the use of the database are collected and written down. In conceptual design these needs are analyzed and the *enterprise schema* is constructed to reflect the needs about the contents of the database. Thus, the enterprise schema should be considered as a part of the documentation of the user requirements. Especially, when the users take active part in the design, the two first steps of the design proceed simultaneously.

Conceptual design is an iterating process. Many preliminary versions of the enterprise schema are usually constructed, evaluated and modified before a satisfactory result is concluded. The schema should be made available in various forms for inspection and evaluation. This is a proper place for a computer assisted tool. Actually, some tools, such as ER—Modeler, IEW, Excelerator, to support the conceptual design have been developed, within the last few years [Xeph88]. These tools differ from each by their technical environment, their functionality, and by the data models that they support. Tools to assist the conceptual design are the kernel components of the YYY database design tool. These tools will be described in section 3.

The next design step, data model mapping, produces the schema for the database management system. If the database management system would use the same data model that is used in representing the enterprise schema this step would not be needed. However, database management systems that are based on semantic data models are not yet common in practice [HuKi87]. Thus, a mapping is needed. The data model, on which the enterprise schema is mapped, is presently increasingly the relational model of data [Codd70]. YYY tool set contains a *mapping tool* that produces relational database schemata. The mapping tool is discussed in section 4.

Section 2 describes YYY database design tool as a whole, and section 5 outlines the further development of the tool set.

2. YYY tool set

YYY database design tool set is the present phase in a series of experimental database design tools constructed during the last ten years at the department of computer science in the University of Helsinki. We started with a semantic data model [LaMP79], and developed a data definition language (HULDA) based on that data model [Lain81]. A compiler to produce the data dictionary representation of HULDA schemata was constructed. Connection matrices and diagram representations could be obtained via a plotter. The tools were implemented in a maim frame (Burroughs) environment. They were experimented in a few database design projects. The conclusions of the experiments are reported in [Lain86]. The data definition language representation of the enterprise schema was found to be good as a detailed document for the adp-professionals. It was not considered good as a document for the end users, nor as an input medium. The further development of the main frame tools ceased when our main frame computer was changed.

The user interface and the end users were of main concern when the next generation of our database design tools were developed. Database design tools that are intended to the end users must be based on their native language. There are many examples of database design projects in Finland, where the end users have not

accepted the documents that contain both English and Finnish [Kall89]. Due to this, we decided to use Finnish language as the only language in our tools.

We used fast prototyping as our development technique. Our starting point was to develop an interactive graphics editor that works in IBM compatible micro computer and could be used for constructing both the diagram and the data dictionary representation of the enterprise schema. The first version of the schema editor was programmed in Basic, because it was the only high level programming language which, at that time, provided the necessary graphic primitives for our Olivetti high density graphics. The prototype proved out to be quite handy in education and in constructing small schemata. For the next version, three alternatives were considered for the implementation tool: Windows graphics environment, Turbo Pascal and compiled Turbo Basic. Because of limited resources, the last alternative was selected. This version has been in educational use for over a year and works well also with large schemata.

One of our goals was to develop a tool that is able to work with 640K of memory. All extensions to the schema editor reduce the maximum size of the schemata that can be processed. To facilitate large schemata we split the functions of the conceptual design tool into three separate tools: *the schema editor, the dynamics editor,* and the *output facility.* It was decided that the further development of the tools should be carried out mainly in Turbo Pascal. Pascal versions of the dynamics editor and the output facility are ready and in use. The Pascal version of the schema editor (version 3) is still under development.

In addition to the tools for the conceptual design, YYY tool set contains a data model mapping tool to produce the relational database schema. This tool is implemented in Turbo Pascal and in Prolog. All the tools can be started from a start up menu. The contents of the present YYY tool set is depicted in Fig. 2.1.

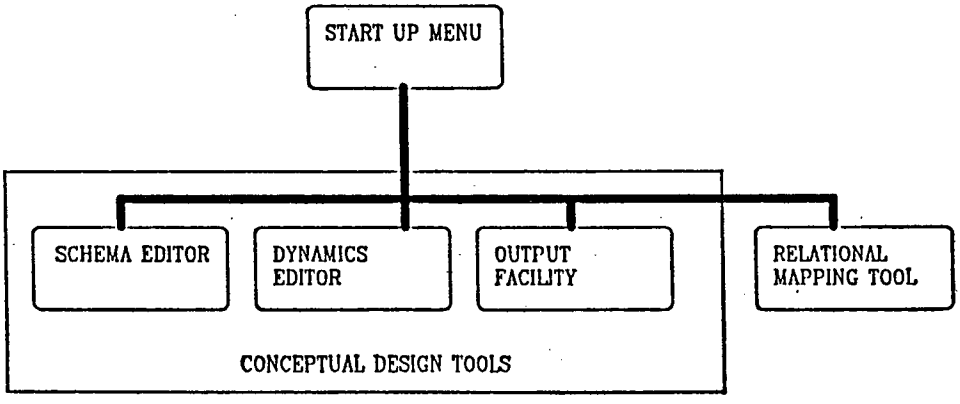


Figure 2.1. Components of the YYY tool set

2.1. Data dictionary

YYY design tools store an enterprise schema as four text files: a file for the static structures (⟨schema⟩.DDA), a file for the definitions related to the static structures (⟨schema⟩.DDB), a file for the description of the dynamics (⟨schema⟩.DDC) and a file for the definitions related to the dynamics (⟨schema⟩.DDD). The schema

files are stored as Prolog like fact files. The structure of the facts is shown in Fig. 2.2. The definition files contain multiple definition text blocks. Each text block starts with a header line, that identifies the object to be defined. This is followed by at most 200 lines of definition text. The text block ends with a footer line. Definition files can be edited with any text editor.

For the processing the whole schema is loaded in memory. Dynamic data structures with multiple indexes are used. The definition files are transformed into indexed files (version 3 of the editor). A definition is loaded into memory only when needed. At the end of the session the indexed definition files are transformed back to text files.

Entity statement:

yksilo(
⟨*entity type name*⟩, ⟨*x-coordinate*⟩, ⟨*y-coordinate*⟩,
⟨*position in hierarchy*⟩, ⟨*population size*⟩,
⟨*schema version identifier*⟩).

Property statement:

ominaisuus(
⟨*entity or event type name*⟩:⟨*property type name*⟩/⟨*value type*⟩,
⟨*functionality*⟩, ⟨*necessity*⟩, ⟨*identifier?*⟩,
⟨*changeability*⟩, ⟨*schema version identifier*⟩).

Relationship statement:

yhteys(
⟨*relationship type name*⟩,
⟨*role name*⟩:⟨*entity type name*⟩
(⟨*minimum restriction*⟩, ⟨*maximum restriction*⟩),
⟨*role name*⟩:⟨*entity type name*⟩
(⟨*minimum restriction*⟩, ⟨*maximum restriction*⟩),
⟨*duplicate counter*⟩,
⟨*x-coordinate*⟩, ⟨*y-coordinate*⟩,
⟨*original duplicate counter*⟩,
⟨*schema version identifier*⟩).

Hierarchy statement:

hierarkia(⟨*entity type name*⟩, ⟨*entity type name*⟩).

Version statement:

versio(⟨*yyy version identifier*⟩,
⟨*schema version identifier*⟩,
⟨*drawing technique*⟩).

Event statement:

tapahtuma(⟨*event name*⟩, ⟨*frequency*⟩, ⟨*schema version identifier*⟩).

Effect statement:

vaikutus(⟨*event name*⟩, ⟨*affected object name*⟩,
⟨*affected property name*⟩, ⟨*type of effect*⟩,
⟨*schema version identifier*⟩).

Figure 2.2. Fact structures in schema files

3. Tools for the conceptual design

3.1. The data model

One of the main differences between the conceptual design tools is the data model supported by the tools. There are tools that support multiple data models, but only as far as the drawing technique is concerned. To fully support a data model, is to allow only such constructs that obey the constraints embedded in the model. These constraints vary between different models and also between the variations of the same base model [HuKi87]. The most commonly supported data model in the design tools is the Entity—Relationship Model, and its numerous variations [Xeph88, HuKi87]. The original Entity—Relationship model [Chen76] imposes very few restrictions on the constructs. Relationship types of any degree may be defined. Both entities and relationships may have attributes. Only entities may participate in relationships. On my experience it is hard to work with such a loose model. There are too many ways to model the same phenomena, and the model does not force to select any of them.

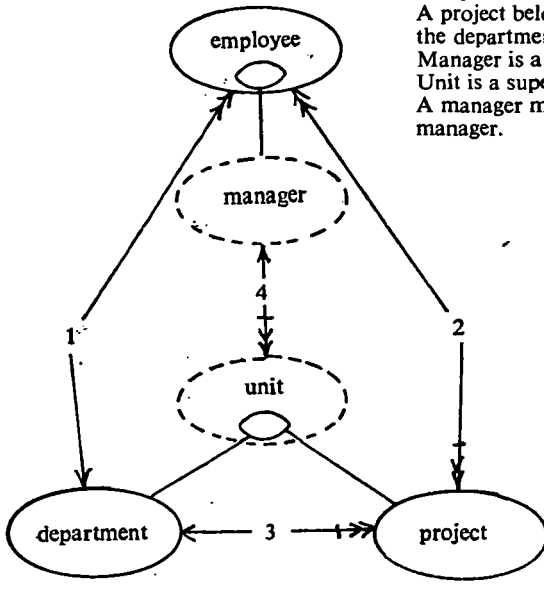
YYY supports an Entity—Relationship Model variant. The kernel of this variant is the simplified Entity-Relationship model variant recommended by the Finnish Standardization Association [SFS88, Sorv88]. YYY data model supplements this model with concepts needed in describing the dynamics of the object system and with constraints that restrict the generalization hierarchy. It divides the phenomena of the object system (mini world) into four categories: *entities, relationships, properties* and *events*. Relationships are restricted to the binary ones. Entities and events may have properties, but relationships can not have properties. Only entities may participate in relationships. Properties are represented with (attribute, value) -pairs. It is assumed that the database contains facts about the existence of entities, the existence of relationships and the existence of the properties of entities. Events are phenomena that affect entities, relationships and properties. They are not represented directly in the database.

The design task is to specify the types of entities, relationships, properties and events. The relations between the specified types must also be determined. In YYY a specification is more than just naming the type. All the types can be attached with textual definitions. Property types must be characterized as identifying or nonidentifying, fixed or varying, single-valued or multi-valued, and necessary or just nice to know. Application depended value types can be specified as related to the property types. A selection of about thirty pre-defined value types is provided. Participation restrictions (minimum and maximum) and the use in identification, are used in characterizing relationship types. A generalization hierarchy can be defined between entity types. Some of the entity types must be defined as *base types*. The others are either super types or subtypes.

The dynamics of the object system is modelled with event types. The kind of effect (creation, change, termination) characterizes the relations between event types and entity, relationship or property types.

Diagrams play an important role in conceptual design. They are the means of providing the overview of the design. YYY provides only entity level diagrams. These diagrams hide out the property types. The diagrams show only the static structures. There is a large variety of drawing techniques attached to Entity—Rela-

tionship models. It is fairly easy for a tool to provide multiple drawing techniques if their symbol sets have a lot of common symbols. YYY design tool provides four drawing techniques that are all rather well known in Finland. Chen's original drawing technique [Chen 76] is not included. Fig. 3.1. gives an overview of the default drawing technique in YYY.



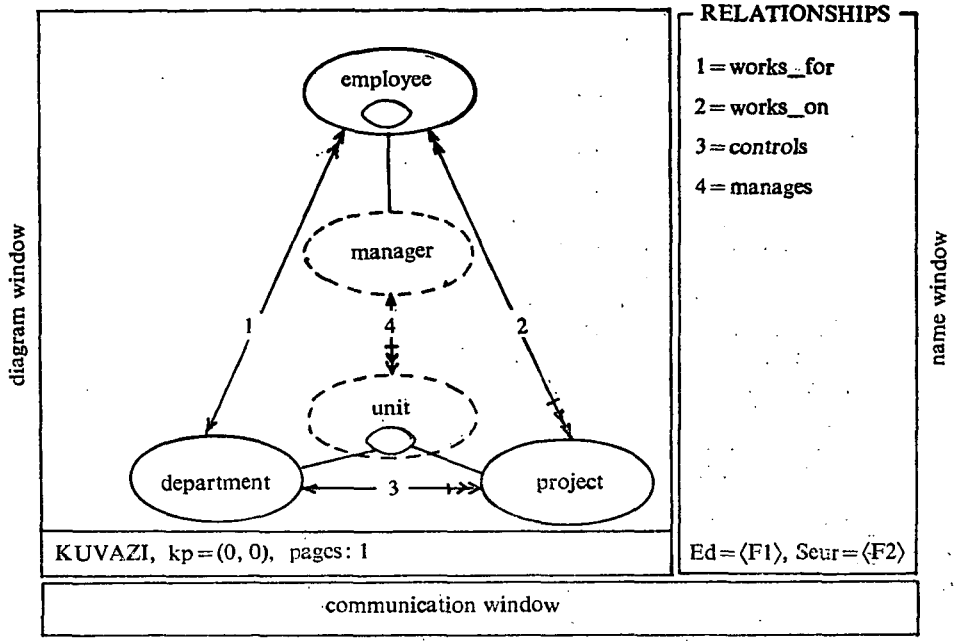
An employee works for exactly one department. A department has many employees, at least one.
A department controls many projects, not necessarily any.
A project belongs to one department and is identified via the department.
Manager is a subtype of employee.
Unit is a super type of department and project.
A manager manages multiple units. A unit has only one manager.

Figure 3.1. An example of an YYY diagram

3.2. The schema editor

The schema editor is used for defining the static structures of the object system, i.e. the entity, relationship and property types. It provides a graphical interactive user interface. The screen is split into three windows. The *diagram window* shows a part of the diagram. The *name window* shows the names that correspond to the short identifiers appearing in the diagram window. The *communication window* is used for user communication, entering specifications and selecting the functions from the menus. Fig. 3.2. shows an example of the schema editor screen.

The user communication is based on menus and on questions asked by the program. The main menu is explained in Table 3.1.



KuvatUlos — NäytönKuva
Tulostus kestää jonkin alkaa
Tulostus tiedostoon MK2. FIC Voit vaihtaa tiedostonimen alkuosan
— Anna uusi alkuosa tai ⟨Enter⟩ Jos hyväksyt oletusarvon:

Figure 3.2. An example of the schema editor screen

Table 3.1. The main menu of the schema editor

Entity	Operations on entity types: specify a new type, remove an existing type, change or view the specification, re-position the entity type symbol in the diagram, integrate two types.
Relationship	Operations on relationship types: specify a new type, remove an existing type, change or view the specification, change the participant, re-position the center point of the connection line.
Hierarchy	Operations on generalization hierarchies: specify a new subset relation, remove a subset relation.
Diagram	Operations on the diagram: re-position entity type symbols, zoom, re-position the diagram window, change the drawing technique, insert/remove page limit lines, expand the diagram window over the whole screen.
Diagram Output	Operations to obtain sketch quality output: Select the output device, print the screen image, print the expanded diagram window image, print entity names, print relationship names.
Exit	The construction of the data dictionary files. Exit the program, if wanted.

To make the use of the tools easier some of the menu alternatives activate a bunch of actions. For example, the selection 'specify new entity type' activates the following actions:

repeat {
Name the entity type,
Specify the population size, position in hierarchy and textual definition,
Position the entity type symbol in the diagram,
repeat {
Identify an entity type to which the new one is connected,
Specify the relationship
} until empty name is provided,
if the new type is 'super type' then
repeat {identity the type that is 'lower' in hierarchy
} until empty name is provided
else if the new type is 'sub-type' then
identify the type that is 'upper' in hierarchy,
when 'wanted'
repeat {Specify a property type
} until 'no more property types'
} until empty name is provided.

YYY schema editor requires the user to position the entity symbols in the diagram. The connection lines that represent the relationships are drawn automatically, but may be edited later on. Manual positioning of symbols requires more user effort than automatic positioning. An advantage of manual positioning is that the user has the control over the diagram, a small change does not re-organize the hole diagram. YYY schema editor offers a default answer for most of its questions. This makes the construction of the enterprise schema easy. As related to each modification of the schema the editor makes some integrity tests. These tests are not, however, complete. The new version of the editor will perform a more complete integrity and quality test each time the schema is written in the data dictionary.

YYY schema editor has been used in database design education in the University of Helsinki for three years. It has had several hundreds of users. It has proven to be easy to learn and quite useful.

3.3. The output facility

High quality paper documents of the enterprise schema are very important in database design. These documents are used as working papers within the design project. In addition, they are the means to distribute information about the design to the users and to the environment. One requirement for the documents is that they should be obtained quickly. The printing time of about ten minutes per page, which is quite common in many design tools, is just too much.

YYY produces all its paper output via auxiliary files. The schema editor can produce output files of the diagram on screen image basis. The files are produced for IBM Graphics Printer/Epson graphics compatible matrix printers. It takes less than a minute to produce an output file and about a minute to print the file with a matrix printer.

YYY tool set contains a separate output facility for the production of high quality paper documents. Diagrams are coded in PostScript. The output facility provides many alternatives to determine how the diagram shall be split for output.

These include: the whole diagram into one A4-sheet (recommended if the diagram is not very large, example as Appendix 1), one or two diagram pages in one A4-sheet and user specified area per one A4-sheet. Diagram output ca be obtained in any of the four drawing techniques supported by YYY.

Text documents represent the data dictionary information as a formatted report. Users may control the amount of definition texts in the report (without definition, with entity and relationship level definitions only, and with exhaustive definition texts). The document can be restricted to cover only those objects that have changed since a certain version of the schema. Fig. 3.3 shows an example of a textual report.

SCHEMA: EXAMPLE 04. 0.8. 1989 15:46:06 page: 2

Entity type: employee — base type, population about: 100

Property	Value type	Characteristics
ssn	sos—sec—no	identifier
fname	name	necessary
lname	name	necessary
sex	sex	necessary fixed
bdate	date	necessary fixed
address	address	necessary
salary	money	necessary

Relationship type	Role	Holder	Restr.
works-on	employee	employee	0—n
	project	project	1—n
works-for	employee	employee	1—1
	department	department	1—n

Hierarchy upper	Hierarchy lower
employee	manager

Figure 3.3. An example of a textual report *(translated into English)*

3.4. The dynamics editor

The dynamics editor can be used in describing the 'real world' events and their effects on entities, relationships and properties. No graphical symbol is used for an event type. In the diagram the affected objects are high-lighted with color. It is possible to define event types that affect multiple objects, both entity types and relationship types. When effects are defined the affected objects are selected from a pick list. The present version of the dynamics editor is the first prototype and it is not yet complete. Output facilities are still missing and the user interface should be improved. Our intention is to implement the textual reports and the effect matrices in the near future. We are also going to implement a cluster analysis program and apply it to the effect matrices to see whether it can assist the specification of the database update programs.

4. The mapping tool

YYY contains also an expert system for generating the relational database schema. The reulting schema is represented as a collection of SQL 'Create table'-statements. Four variations of SQL are supported ORACLE—SQL [Orac87], DB2—SQL [DaWh88], INGRES—SQL and a slightly extended ISO SQL-standard (proposed features of the forthcoming SQL2 standard are included) [ISO87]. The resulting schema specifies the structures of the relations, their primary keys and also their foreign keys (inclusion dependencies). When the SQL variation does not contain means to define the primary keys or the foreign keys, information about them is included as a comment.

The generation is carried out in three phases. The first phase pre-processes the enterprise schema and eliminates certain characters of the Scandinavian character set that cannot be used in SQL.

The second phase applies the transformation rules. It produces a file that specifies the relations, their attributes, their keys and their foreign keys. It also gives an explanation on what rules were used for each decision. The transformation can be done either in automatic or in interactive mode. In automatic mode default rules are used in certain problematic situations, such as, whether to represent a subtype (by default no) or a super type (by default yes) as a relation of its own, and whether to introduce artificial identifiers to replace large user keys that are widely used as foreign keys (by default no). In interactive mode the situation is explained to the user, a solution is proposed and acceptance or rejection is requested.

The third phase constructs the SQL statements. Application dependent value types are transformed into SQL data types. A conversion table is included for the pre-defined value types. Users may add new items in this table or replace it with a table of their own. The conversion table defines the conversion between the application dependent value sets and the SQL standard. Rules that are embedded in the program take care of adjusting the type definitions for the supported database management systems. If there is no conversion rule for some value type the corresponding data type will become 'undefined'. Column names may be problematic in the generation of the SQL statements. The SQL variations accept names with the length from 8 to 32 characters. The enterprise schema accepts property names of up to 24 characters and the second step of the transformation may generate foreign key names of up to 128 characters. No automatic cut off is provided. If the user works in interactive mode, it is possible for her to rename all the tables and columns. The names that are longer than what is allowed are indicated. YYY keeps track on the items that have been renamed. When the relational schema is re-generated the renaming made during the past generation runs can be taken into account, even in the automatic mode of the transformation.

Several rule sets to map an Entity—Relationship schema to a relational schema have been proposed e.g. [MaSh89, Chen76, ElNa89]. Our rule set consists of twenty rules. We aim to a minimal set of relations. Thus we accept null values, express the properties of a subtype in the relation attached to the corresponding base type, and represent all 1-to-n relationships with foreign keys. It is not allowed to specify identifying properties for sub- and super types. If a super type is represented as a relation of its own (the default) this relation will contain an artificial key, and an add-on attribute 'type' to indicate the base type. We have also the rule "If the user key of the

relation is big and it should be used in many references then consider an artificial key".

To produce normalized relations, our mapping rules as all the other rules, presuppose that the Entity—Relationship Schema is normalized, i.e. redundancy is eliminated, properties are attached to correct entity types, and the relationship types are defined between the correct entity types. The analysis of the quality of the enterprise schema cannot be done automatically based on the information in the schema. But, it is possible to look for the potential problems and show them to the user to reconsider.

5. Future development

In the present system the definition texts must be written with a separate editor. This problem will be corrected already in the next version of the schema editor that will contain an embedded text editor. The tool set does not contain any tool for view integration. A simple integration tool that renames the objects and reorganizes the diagrams so that the schema editor can be used for integration has been designed. An add-on function that makes it possible to directly utilize data item lists as pick lists in specifying properties is planed, as well as the improved integrity and quality test for the enterprise schema.

References

[Chen76] CHEN P. P., The Entity-Relationship Model — Towards a Unified View of Data, ACM Trans. on Database Systems, 1, 1 (March 1976), pp. 9—36.

[Codd70] CODD E., A Relational Model for Large Shared Data Banks, Communications of the ACM, 13: 6, 1970.

[DaWh88] DATE C. and WHITE C., A guide to DB2, Second Edition, Addison-Wesley, 1988.

[ElNa89] ELMASRI R. and NAVATHE S. B., Fundamentals of Database Systems, Benjamin Cummings, 1989.

[HuKi87] HULL R. and KING R., Semantic database modeling: Survey, applications and research issues, Computing Surveys, 19, 3 (Sept. 1987), pp. 201—260.

[ISO87] ISO: ISO 9075, Information processing systems — Database language SQL + addendum 1, 1987.

[Kall89] KALLIALA E., IEW — Case software and SRM, Univ. of Helsinki, Dept. of Computer Science, Series C—1989—35, (in Finnish), 1989.

[Lain81] LAINE H., A Data Definition Language and its Use in Data Base design, Report of the Fourth Scandinavian Research Seminar on Systemeering, Univ. of Oulu, 1981, pp. 188—197.

[Lain86] LAINE H., User Interface in a Database design tool, Report of the Eighth Scandinavian Research Seminar on Systemeering, Aarhus University, Denmark, 1986, pp. 262—277.

[LaMP78] LAINE H., MAANAVILJA O., and PELTOLA E., Grammatical Data Base Model, Information Systems, 4:4, 1978.

[MmSh89] MARKOWITZ V. M. and SHOSHANI A., On the Correctness of representing Extended Entity-Relationship Structures in the Relational Model, Proc. ACM SIGMOND—89 Conf., 1989, pp. 430—439.

(Orac87] ORACLE CORP., SQL*Plus Reference Guide, 1987.

[SFS88] Finnish Standardization Association: Methods of Systems Development; ERTI—Conceptual Analysis, Handbook 106, 1988 (in Finnish).

[Sorv88] SORVARI J., ERTI — A method for Conceptual Analysis and Modeling of Information, Joint Finnish Soviet Software Symposium, Helsinki 15—18. 11. 1988, Proceedings to appear in 1989.

[TeFr82] TEOREY T. and FRY J., Design of Database structures, Prentice Hall, 1982.

[Xeph88] Xephon Inc.: Case Systems: Xephon Buyers Guide, 1988.

Appendix 1: High quality diagram output-all in one page