



universität
wien



Faculty of Computer Science



Method Handbook

Semantic Database Design (SDBD)

University of Maryland

Professor Dr. Nicholas Roussopoulos

Department of Computer Science

University of Vienna

Professor Dr. Dimitris Karagiannis

Faculty of Computer Science

Institute for Knowledge and Business Engineering

December 2008

Table of contents

1. Introduction and Goals	5
1.1. Area of Application of the SDBD Method.....	5
1.2. Perspective.....	5
1.3. Overview	5
2. An Adaptable Methodology for Database Design	7
3. Overview on the SDBD Method	25
4. Development of the SDBD Tool via Customisation Utilising the ADOxx 1.0 Platform.....	39
4.1. Defining the Metamodel	39
4.2. Graphical Representation of Modelling Classes and Relations.....	41
4.3. Define Modeltypes.....	42
4.4. Additional Functionality	43
5. User Manual for the Semantic Database Design Tool (SDBD Tool)	45
5.1. Environment and Requirements Analysis	46
5.1.1. Steps	46
5.1.2. ‘Semi-Formal’ Notation.....	47
5.1.3. Techniques.....	47
5.2. System Analysis and Specification	48
5.2.1. Steps	48
5.2.2. ‘Semi-Formal’ Notation.....	51
5.2.3. Technique.....	52
5.3. Conceptual Modelling.....	52
5.3.1. Steps	53
5.3.2. ‘Semi-Formal’ Notation.....	54
5.3.3. Techniques.....	56
5.4. Logical Schema Design	56
5.4.1. Steps	57
5.4.2. ‘Semi-Formal’ Notation.....	57
5.4.3. Techniques.....	58
5.5. Task Emulation & Optimisation.....	59
5.5.1. Steps	60
5.5.2. ‘Semi-Formal’ Notation.....	64
5.5.3. Techniques.....	65
6. Best Practices	67
6.1. Phase 1 – Environment and Requirements Analysis.....	67
6.2. Phase 2 – System Analysis and Specification	68
6.3. Phase 3 – Conceptual Modelling	71
6.4. Phase 4 – Logical Schema Design	72
6.4.1. Base Relation Pool.....	72
6.4.2. Automatically Generated SQL Create Statements	73

6.5. Phase 5 – Task Emulation & Optimisation.....	74
6.5.1. Generate the Data Input Requirements of Each Task	74
6.5.2. Compute Probabilities of Query Graphs	76
6.5.3. Integrate Query Graphs into a LAP Schema	77
7. How to extend the SDBD Tool	78

Table of figures

Figure 1: Schematic diagram of the methodology (simplified)	6
Figure 2 ADOxx Application Library	39
Figure 3 Class Hierarchy Management Component.....	40
Figure 4 Adding Attributes – Available Attribute Types.....	41
Figure 5 GraphRep – Definition of the Representation of Modelling Classes	41
Figure 6 Definition of Modeltypes.....	42
Figure 7 Modeltypes of the SDBD Tool.....	42
Figure 8 New Model Dialogue.....	43
Figure 9 Phases of the SDBD Method	45
Figure 10 Key Concpets of Each Phase	45
Figure 11 Key Concepts within 'Environment and Requirements Analysis'	46
Figure 12 Recommended Model Group Structure – Information Flow Diagrams.....	46
Figure 13 Key Concepts within 'System Analysis and Specification'.....	48
Figure 14 Recommended Model Group Structure within 'System Analysis and Specification' ..	49
Figure 15 The Concept of 'Pool Models' – Reuse of Tasks and Documents	50
Figure 16 Notebook of Tasks (within Task Pool Model)	50
Figure 17 Assigning 'Task Flow Models' to 'Processes'	51
Figure 18 Modelling Classes of the Modeltype „Task Flow Model“	52
Figure 19 Key Concepts within 'Conceptual Modelling'	53
Figure 20 Tasks and Assigned Input Documents	53
Figure 21 E-R Diagram within the Model Catalogue.....	54
Figure 22 Derivation of the E-R Model.....	54
Figure 23 Modelling Classes of the Modeltype 'E-R Model'	55
Figure 24 Technique 'Relation Table'	56
Figure 25 Key Concepts within 'Logical Schema Design'	57
Figure 26 Modelling Classes of the Modeltype „Base Relation Pool“.....	58
Figure 27 SQL Create Statements.....	58
Figure 28 Base Relations and Attributes.....	59
Figure 29 Key Concepts of 'Task Emulation & Optimisation'.....	60
Figure 30 Logical Access Path Models, Query Graphs	61
Figure 31: Query Graph Notation.....	61
Figure 32: Assignment of Query Graphs to Tasks	62
Figure 33 Specifying Task and Query Frequency.....	62
Figure 34 Query Graph incl. Computed 'Query Probabilities' and Computed 'Update Probabilities'	63
Figure 35 Integrating Query Graphs into a LAP Schema	63
Figure 36 Modelling Classes of the Modeltype 'Logical Access Path Diagram'	64
Figure 37 Input Box for the Specification of 'Query Graphs'.....	65
Figure 38 Automatically Generated Query Graph.....	65

Figure 39 Choosing Single Query Graphs	66
Figure 40 Integrated Logical Access Path	66
Figure 41 Information Flow Diagram.....	67
Figure 42 Task Flow Diagram	69
Figure 43 Pool Models	70
Figure 44 Example of an E-R Diagram	71
Figure 45 Base Relation Pool	72
Figure 46 Query Graph 'Local Projects'	74
Figure 47 Query Graph 'Projects with Exceeding Budgets'	75
Figure 48 Query Graph 'Orders Currently Processed'	75
Figure 49 Query Graph 'Orders Currently Shipped'.....	76
Figure 50 Query Graph 'Orders in Process for Projects with Exceeding Budget'	76
Figure 51 Shipped-Orders Query Graph incl. Probabilities	77
Figure 52 Resulting LAP Schema.....	77

1. Introduction and Goals

1.1. Area of Application of the SDBD Method

Design of database with its applications.

1.2. Perspective

The SDBD method presented in this method handbook assumes that the primary purpose of the future system is to automate current or planned activities of the enterprise. The method assumes (as do all database design methodologies) that different views on the enterprise, conflicts, and political differences will be resolved during the database design process.

1.3. Overview

The proposed process-based methodology consists of five phases: (1) Environment and Requirements Analysis, (2) System Analysis and Specification, (3) Conceptual Modeling, (4) Logical Schema Design, and (5) Task Emulation and Optimisation. A schematic diagram of the methodology is given in Figure 1.

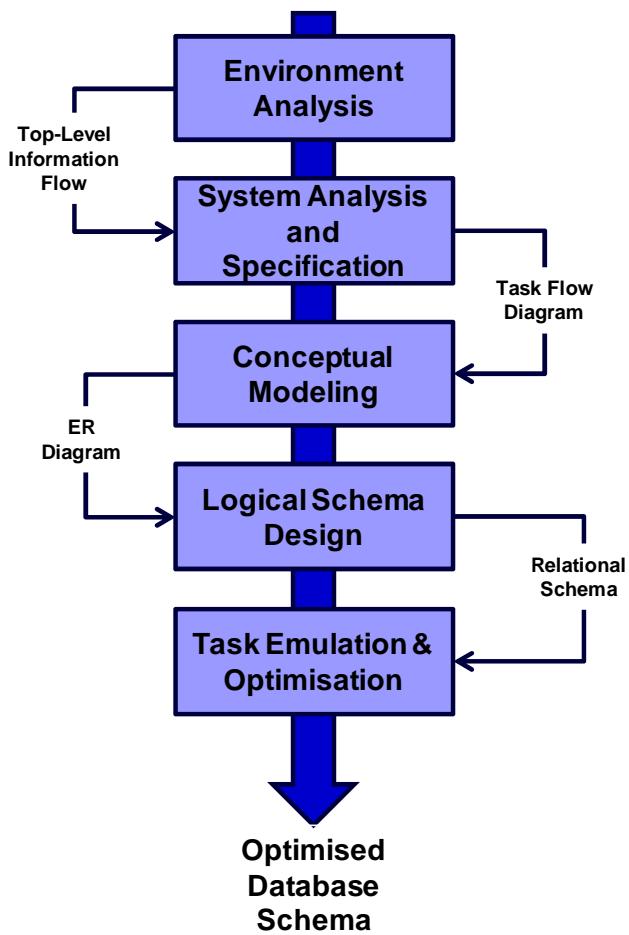


Figure 1: Schematic diagram of the methodology (simplified)

2. An Adaptable Methodology for Database Design

Paper “An Adaptable Methodology for Database Design” of Nicholas Roussopoulos and Raymond T. Yeh, University of Maryland.

In this truly adaptable approach, designers begin by defining the application environment and then work inward until they reach a level that can be handled by available DBMSs.

An Adaptable Methodology for Database Design

Nicholas Roussopoulos and Raymond T. Yeh, University of Maryland

Database design has changed considerably since its inception. A large part of the design used to deal with physical data allocation, load factors, and access methods of secondary storage. These aspects are no longer part of the database design process because they are part of all commercial database management systems and cannot be modified. Most current database designs, and possibly all future ones, will stop at logical access path optimization and use existing database systems with fixed physical designs. Consequently, the emphasis will be on design at the information system level.

The design method presented here is based on this very philosophy—that database design must be done “from the outside in.” We must analyze the proposed system’s environment and proceed progressively inward toward the computer implementation of the application. In most cases the analyst is then directly exposed to the requirements, which is really the key to a successful system design, if the system is to operate effectively within a particular environment. Take, for example, a project that automates existing manual procedures; the computer system is a direct reflection of the current operations, that is, of the environment.

The parts of the database design of most concern to us here are analyzing information processing requirements, constructing a conceptual model that specifies these requirements, and developing and optimizing a logical access path schema. The underlying assumption in our approach to database design is that conceptual modeling can be based on how information flows between an enterprise and its environment and among its components. [One may argue that some kind of model (possibly not precise) has already been conceived, and therefore that conceptual

modeling has already begun before the information flow is conceived. Regardless of how true this supposition may be, our purpose here is to model the database, not to deal with how the human brain handles “chicken-or-the-egg” problems!] We consider conceptual modeling as a technique for specifying, in a formal language, concepts and ideas that can be interpreted by other humans familiar with the context of the enterprise and its environment.

For this reason, we start with the environment of the enterprise rather than the environment of the data processing system. In many cases, the requirements of the data processing system may be distorted by established practices, existing hardware and/or software, or personnel resistance to system upgrading. Starting conceptual modeling at the information system level rather than the data processing level helps avoid these low-level distortions. Furthermore, the goals and requirements of this level are more understandable to management. Hence, managers are more apt to make correct decisions.*

In our methodology, we also analyze the operational behavior of the enterprise, which we call system analysis, the word “system” referring to operational behavior, not to the data processing system. Without a complete understanding of how the enterprise operates, no effective design can be developed either for current operations or for future improvements.

*Several other design methodologies start from the environment and proceed inward. Jefferson et al.¹ database design departs from the requirement analysis at the corporate level where many of the interactions are between the organization and its environment. Kahn² also proceeds from corporate level requirements to information analysis where again interactions include the information flow between the environment and the organization. BIAITI³ has also incorporated these kinds of interactions into its questionnaires.

Another main objective of the outside-in methodology is to provide a framework for the use of well-understood and proven techniques for both the analysis and the specification of the enterprise's objectives, requirements, and design. We have incorporated established data flow techniques,^{4,5} control flow specification techniques,⁶⁻⁸ structured interviewing and questionnaire techniques,⁹ well-understood data modeling techniques,¹⁰⁻¹⁶ and logical optimization techniques.¹⁷⁻²⁰ Emphasis has been placed on the understandability and the completeness of the requirements specification. All referenced approaches deal with either database design or program design. This methodology integrates techniques from both approaches because a database system is neither just a schema (structural component) nor just a set of transactions (procedural component). It is a schema with transactions running against this schema; consequently, the schema and the transactions must be designed hand in hand.

A major advantage of this framework is that significant design decisions are made explicit at the appropriate design phase. For example, obtaining optimal physical parameters, makes no sense if the logical design is ineffective. Similarly, obtaining optimal logical organization and indexing makes no sense if the corresponding conceptual schema is incorrect and violates the integrity of the data. Again, a good conceptual schema is useful only if it corresponds to the right operational behavior of the enterprise within its environment. This framework and the design decisions along with their specification are the important concepts of this proposal. The individual models and techniques to obtain, specify, and document these decisions can be chosen from the wide variety found in literature.

Methodology overview

Our proposed modular methodology consists of four phases: (1) environment and requirements analysis, (2) system analysis and specification, (3) conceptual data modeling, and (4) derivation of a logical access path schema. A schematic diagram of the methodology is given in Figure 1.

The objectives of the first phase are to understand the actual state of the enterprise and to collect all the information needed by the subsequent steps, including information and data processing requirements. Information requirements will be used in the conceptual data modeling phase to generate the conceptual schema of the target system. The data processing requirements will be passed to the logical access path schema derivation phase in which the logical optimization and the application programs are designed.

The objective of the second phase is to identify the applications that will use the database. Using input from the environment analysis phase, we obtain a good understanding of the target system. The output is a set of "tasks" to be performed during the database operation.

The objective of the third phase—conceptual data modeling—is to translate the knowledge about the target system collected in phases 1 and 2 into a formal representa-

tion—a conceptual data model. In this phase, designers are concerned only with the characteristics of the "entities" in the application environment and the relationships among them. They should not be concerned with the way they are represented in the computer.

In the last phase, we have several objectives: (1) to map the conceptual schema into the organization's data model structure, (2) to generate logical access paths of the tasks, (3) to integrate all paths into another schema that models their relationships and their cumulative usage weights, (4) to optimize this access schema, and (5) to realize the tasks' logical access paths against the optimized schema.

Numerous related papers have influenced our work. Phases 1 and 2 have been influenced by the works of Burnstine,³ Bubenko,²¹ Chen,²² Lum et al.,²³ Tsichritzis and Lochovsky,^{24,25} Sheppard-Rund,⁹ and Kahn.² Phase 3 is based mainly on our work on the semantic models of databases that started with the semantic network model of databases^{12,26,27} and its continuation in providing a language to support the design of conceptual schemata.¹³ Approaches by Mylopoulos et al.,¹⁴ Weber,²⁸ Hammer and McLeod,²⁹ Chen,¹⁰ Sowa,³⁰ Falkenberg,³¹ Smith and Smith,³² and Brodie³³ can also be used in phase 3 as can other semantic models. Phase 4 has been influenced by Chang and Cheng,³⁴ Chang and Ke,³⁵ and Schkolnick.¹⁸

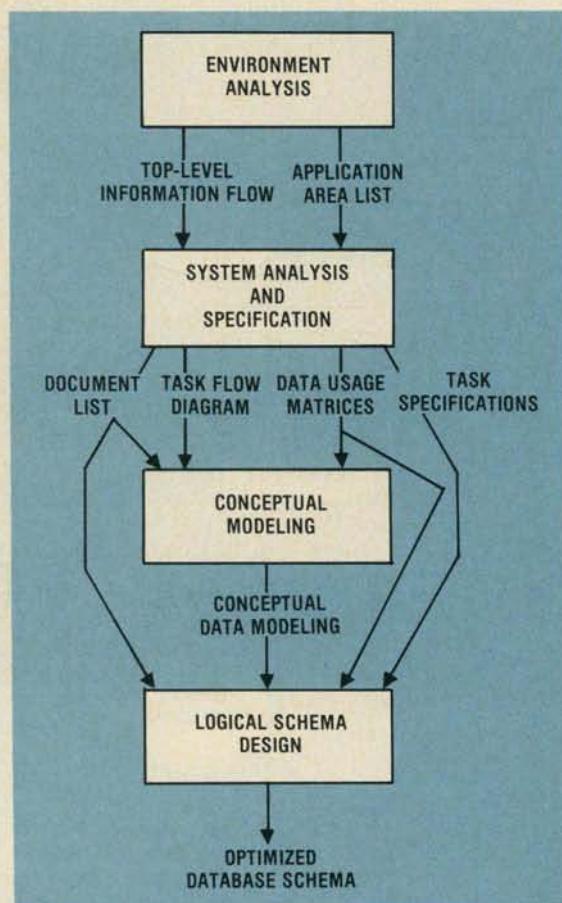


Figure 1. Schematic diagram of the phases of the database design methodology.

Phase 1: Environment and requirements analysis

In this phase, we investigate the information needs of and activities within the enterprise and determine the boundary of the design problem. Much of this phase involves information collection. An overall model of the enterprise as an information processing system is constructed. Since we are not attempting to model or discipline the way humans conceive "understanding," we provide only a list of information sources on which this understanding must be based and a list of things to be generated. In other words, we specify the input and the desired output of this phase, thus leaving the processing to human ingenuity.

Information sources: phase 1 input. The basic technique for collecting information consists of reviewing documents, interviewing people, and analyzing questionnaires to determine facts, policies, objectives, and constraints. These information sources describe the current status of the enterprise, possible inefficiencies, plans for the future,

and constraints that have to be satisfied in conducting business.

Other sources of information include organization charts, reports, forms, files, and software documentation (if such documentation exists).

Functional specification: phase 1 output. In this step, we establish a functional model of the enterprise by identifying its major activities or functions and their relationships. This functional model will take the form of an information flow diagram.

Usually, the function associated with each operational activity is too complex to comprehend and model using a flat, single-level presentation. Therefore, we must usually divide the functions into smaller units called *tasks*.

Identification of activities and tasks is important in the design of the database system. If the system under design is to support the activities of an established enterprise, an organization chart can be used for identifying functions and tasks. However, this process becomes more difficult if (1) a new organization is to be established and its components have not yet been identified or (2) a system is designed to change part or certain aspects of an

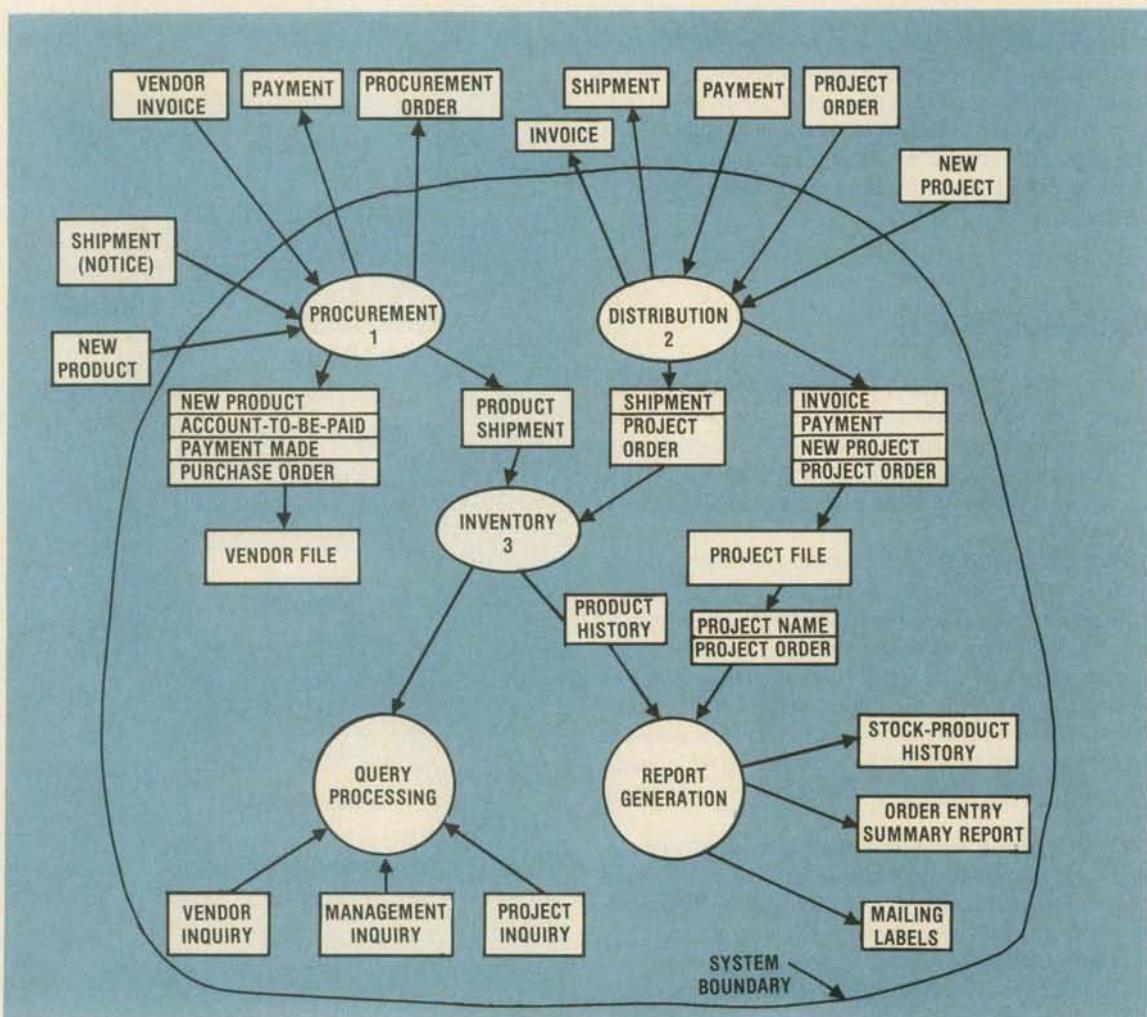


Figure 2. Top-level information flow diagram.

enterprise's activities. In these cases, we must use information collected from the previously described information sources to derive tasks.

The main activities of the enterprise, the interactions of the external entities and the enterprise, and the interactions of the main components can be described by a *system flow diagram* like the one shown in Figure 2.* This flow diagram, which is the goal of the first phase, contains (1) the main activities of the system (PROCUREMENT, INVENTORY, DISTRIBUTION), shown in the ellipses; (2) the data processing functions (QUERY-PROCESSING, REPORT-GENERATION), shown in circles; and (3) the documents that carry the data between components of the organization, shown in rectangles. (A document here is merely a general information-recording medium for information transfer. All messages, forms, files, inquiries, and reports will be referred to as documents.) The boundary of the system shows the interactions of the environment and the enterprise. Clearly, Figure 2 describes not only data flow but also control flow because many of these documents could be simple control messages that trigger tasks in the organization.

Phase 2: System analysis and specification

In this phase, we start with the overall information-flow diagram like that in Figure 2. Each activity is divided hierarchically into more and more detailed tasks and subtasks. The tasks should be reasonably independent so that we will have a small number of task-task interfaces. During the division process, the (data) documents used by each activity are also broken down into their component data elements or subsets of data elements. The tasks are further divided into subtasks until they become small enough to be clearly understood, and the I/O data documents can be conveniently expressed in terms of data elements that cannot be further divided. A new data- and control-flow diagram is developed using the tasks established through this breakdown. The following tasks were obtained for the Distribution activity through this process.

- ENTER-NEW-PROJECT
- ENTER-GRANT-ORDER
- ORDER-PICKING
- SELECT-SHIP-ROUTE
- MAINTAIN-PROJ-TRANSACTION-RECORD
- PROJECT-BILLING

Specifying tasks. For each task, the following attributes need to be specified:

- the person in charge (task performer),
- documents used and/or generated (I/O documents),
- procedure for carrying out the task,
- frequency of the task,
- conditions that specify the prerequisites for the task's activation and the effects after its termination,

*To facilitate an understanding of our methodology, we have chosen a model organization called the ABC Agency. The ABC Agency procures products from vendors and retails them to government projects. Each vendor can furnish several products, and products can exist in several versions or models. Similarly, each product model can be furnished by several vendors. A project may be located in several places. An order may be shipped to any location, but the invoice is always sent to the project's home office.

Naturally, the ABC Agency is only one type of a wide variety of organizations. The steps in our methodology, however, can be broadly applied.

- events triggered during the task's performance, and
- error (anomaly) conditions of the task.

Among these attributes, the I/O specifications, task frequency, and the operations performed are the ingredients needed to emulate the task and construct the logical access path schema for its optimization. The pre- and post conditions, events triggered, and error conditions specify the control and each task's interactions with the rest of the system.

A standardized task form has been designed to assist the responsible designer in extracting the important characteristics, features, and attributes of each task. A sample of this standardized form, which task performers and designers can cooperate in filling out, is shown for the Distribution activity (Figure 3).

This task allows the designer to analyze both existing activities and those considered for implementation that have not been divided into tasks and subtasks. Thus, starting with the main activities of the enterprise, the designer can specify all functional requirements in terms of this task form.

The following rules of thumb can be used to decide when a given task should be divided into subtasks:

Task-number:	S-1
Task-name:	DISTRIBUTION
Performer:	Distribution Department
Authorization:	Various authorizations to be specified in the sub-tasks
Purpose:	Distributes products to the projects and handles accounts
Enabling-conditions:	Permanently executing
Description:	Order Forms (D3) are used to determine which products are to be shipped to which customers.
Frequency:	N/A
Duration:	N/A
Importance:	Very important
Maximum-delay:	N/A
Input:	Documents: D2 (NEW-PROJ-FORM) D3 (ORDER-FORM) D5 (DAILY-ORDER-BATCH) D6 (PROJECT-BALANCE) D11(PAYMENT) D12(ADJUSTMENT)
Output:	Documents: D4 (WEEKLY-ORDER-HISTORY) D5 (DAILY-ORDER-BATCH) D6 (PROJECT-BALANCE) D7 (PROJECT-INVOICE) D8 (PRODUCT-QUANTITY-ON-HAND) D9 (PRODUCT-TO-BE-SHIPPED) D10(SHIPPING-ROUTE) D13(MONTHLY-STATEMENT) D14(EXCEEDING-LIMIT-PROJ-LIST)
Use of documents:	All columns of D2, D3, D6, D11, D12
Operations-performed:	The detailed operations are described in the subtasks.
Subtasks:	ENTER-NEW-PROJECT (S-1-1) ENTER-GRANT-ORDER (S-1-2) ORDER-PICKING (S-1-3) SELECT-SHIP-ROUTE (S-1-4) MAINTAIN-PROJ-TRANSACTION-RECORD (S-1-5) PROJECT-BILLING (S-1-6)
Error-conditions:	N/A

Figure 3. Task form filled for the Distribution activity.

- Many performers are required to carry out the task and each performer has different skills, or each can carry out a part independently.
- Different levels of authorization exist for carrying out different parts of the task.
- Different enabling conditions activate parts of the task.
- Different frequencies and durations apply to different parts of the task.
- Input documents are not used uniformly within the task.
- Different documents are used for different parts of the task.
- Many diversified operations are carried out within the task.
- Many subtasks are controlled by the task.

There are basically two ways of subdividing a task. In the first, the new task is a subtask of the original task. In this case, a “parent-child” relationship is established and part of the responsibilities are transferred to the subtask. However, control stays with the original task, which has to wait for the subtask to terminate before it can terminate. In the second type of subdivision, the new task is a “sibling” of the original task. In this situation, the “parent”

Task-number:	S-1-2
Task-name:	ENTER-GRANT-ORDER
Performer:	Tom Keener, Distribution Coordinator
Authorization:	Scott Farley
Purpose:	Enters orders in the database and grants orders
Enabling-conditions:	Arrival of ORDER-FORM (D3)
Description:	ORDER-FORMS (D3) are distributed and recorded. Balance of ordering project is checked to determine whether order should be granted. Order history is generated every week.
Frequency:	300 per day
Duration:	N/A
Importance:	Very important
Maximum delay:	2 days
Input:	Documents: D3 (ORDER-FORM) D6 (PROJECT-BALANCE) D5 (DAILY-ORDER-BATCH)
Output:	Documents: D6 (PROJECT-BALANCE) D5 (DAILY-ORDER-BATCH) D4 (WEEKLY-ORDER-HISTORY)
Use of documents:	All columns of D3, D6, D5
Operations-performed:	Upon receipt of D3, it is numbered and distributed. Information on D3 is then entered into the database. The balance of the ordering project (D6) is updated and checked. If the project balance does not exceed its limit, the order is granted. Every morning all granted orders of the previous day are batched in (D5) together for a pickup. Every Friday D4 is generated.
Subtasks:	ORDER-FORMS-RECEIVE-DISTRIBUTE ENTER-ORDER-INFO UPDATE-PROJ-BALANCE INSUFFICIENT-BUDGET-HANDLING ORDER-HISTORY-GENERATION
Error-conditions:	Operation of this task halts if the information of the projects and products does not match with the data in the database.

Figure 4. Task form filled for subtask ENTER-GRANT-ORDER.

of the original task assumes control, but the “sibling” tasks are carried out independently.

During the “parent-child” task division, the input documents of the original task are separated and split among the subtasks. If parts of one or more documents are needed for a subtask, then a new document is defined. The new document will include all the necessary data elements.

During the “sibling-sibling” task division, the original task’s input documents may need to be further divided, as in the “parent-child” case, to account for splitting responsibilities between the two. Some additional control documents may have to be introduced for communication among the tasks if they are required to pass data and/or synchronize some activities.

Figure 3 shows a list of subtasks obtained for the Distribution activity. By examining the input documents of the Distribution activity, new tasks such as ENTER-NEW-PROJECT, ENTER-GRANT-ORDER, and ORDER-PICKING are identified. Further division may then be necessary as the forms for tasks ENTER-GRANT-ORDER and ORDER-PICKING show. (Figures 4 and 5).

Specification of documents. From the I/O specifications of the activities and tasks, we can obtain a list of the documents (reports, files, data elements, inquiries to files, forms, etc.). The list includes documents used within the enterprise or documents discovered by the requirements analysis. Messages received by the task performer during task execution are also considered data documents. Below is a partial document list for our sample enterprise:

D1 - PROJECT-RECORD
D2 - NEW-PROJ-RECORD
D3 - ORDER-FORM
D4 - WEEKLY-ORDER-HISTORY
D5 - DAILY-ORDER-HISTORY
D6 - PROJ-BALANCE
D7 - PROJ-INVOICE
D8 - PRODUCTION-QUANT-ON-HAND
D9 - PRODUCTS-TO-BE-SHIPPED
D10 - SHIPPING-ROUTE
D11 - PAYMENT
D12 - ADJUSTMENT
D13 - MONTHLY-STATEMENT
D14 - EXCEEDING-LIMIT-PROJECT-LIST
etc.

Each document is then expressed in terms of data elements. If a document is already being used, the data elements constituting it are readily available. The newly introduced documents must be divided into new or already established data elements. Documents D1 through D4 and their data elements are shown below

Document D1: PROJECT-RECORD
PROJ-NO
PROJ-NAME
CRDT-CODE
CRDT-LIMIT
PROJ-MAIN-LOC-NO (zip code)
PROJ-MAIN-LOC-ADDR
PROJ-BRANCH-LOC-NO
PROJ-BRANCH-LOC-ADDR
Document D2: NEW-PROJ-FORM
PROJ-NAME
PROJ-MAIN-LOC-NO

PROJ-MAIN-LOC-ADDR
PROJ-BRANCH-LOC-NO
PROJ-BRANCH-LOC-ADDR

Document D3: ORDER-FORM

ORD-NO
DATE-ENTERED
DATE-POSTED
PROJ-NO
PROJ-MAIN-LOC-NO
PROJ-BRANCH-LOC-NO
ITEM-LINE-NO
PROD-NO
PROD-MOD-NO
QUANT-ORDERED

Document D4: WEEKLY-ORDER-HISTORY

ORD-NO
DATE-POSTED
PROJ-NO
ITEM-LINE-NO
PROD-NO
PROD-MOD-NO
QUANT-ORDERED
PRICE

Specification of data usage. From the I/O specification of tasks, and the document specifications, we can construct two usage matrices. The first is the task-document usage matrix, which specifies the task I/O in terms of documents (Figure 6). The second usage matrix is called the task-data element usage matrix and specifies task I/O in terms of data elements (Figure 7). Note that the task-data matrix is constructed from the task-document matrix and the document specifications.

Both data usage matrices can be easily obtained from the task forms and the document specifications. They can also be automatically generated as a report, as PSL/PSA does.⁵

Specification of task flow diagrams. The task-document usage matrix is now used to construct the task flow diagram, a chart that shows the control and data flow among the tasks in the system. Figure 8 shows the TFD of the Distribution function.

Phase 3: Conceptual modeling

The goal of this phase is to translate the knowledge about each activity, collected in the previous phases, into a formal representation called a *conceptual schema*. This data modeling refers only to the structural specification. (The general term of conceptual modeling used in phases 1 and 2 included the procedural specification as well.)

During the conceptual data modeling phase, we are concerned only with the conceptual entities used in each activity, their properties, and the conceptual relationships among them. The physical or computerized representation of the entities and their instances is irrelevant at this point.

To represent conceptual objects, then, we need a computer-independent semantic model that

- can capture the meaning of the conceptual objects;
- has well-defined interpretation rules so that people other than those in the design group (other designers or users of the system) understand the intended meaning; and

- has a friendly human interface with multiple representations (text and graphical) and high-level operators that manipulate the conceptual schema.

A modeling language like the conceptual schema definition language, or CSDL,¹³ is best suited for expressing conceptual entities, their properties, and the relationships among them. The language has been slightly extended to offer two levels of syntax. The first is high-level syntax

Task-number:	S-1-3
Task-name:	ORDER-PICKING
Performer:	Bruce Smith, Inventory Controller
Authorization:	Howard Daniel
Purpose:	Pick up products of granted orders from the inventory.
Enabling-condition:	The arrival of the DAILY-ORDER-BATCH (D5)
Description:	The inventory is checked against the products specified in D5. Available products are then picked up for shipment.
Frequency:	5000 picked products per day
Duration:	One day
Importance:	Very important
Maximum delay:	4 hours
Input:	Documents: D5 (DAILY-ORDER-BATCH) D8 (PRODUCT-QUANT-ON-HAND)
Output:	Documents: D7 (PROJECT-INVOICE) D8 (PRODUCT-QUANT-ON-HAND) D9 (PRODUCT-TO-BE-SHIPPED) D17(STOCK-UP-REQUEST) D18(UNFULFILLED-ORDERS)
Use of documents:	Parts of D5 and D8, all columns of D7, D9 (as shown in Figure 7).
Operations-performed:	Upon arrival of D5 (DAILY-ORDER-BATCH), the inventory is checked to determine whether the quantity on hand for each ordered product in D8 is sufficient to satisfy all orders. If so, shipping tickets (D9) and invoices (D7) are issued.
Subtasks:	CHECK-INVENTORY ISSUING-SHIPPING-TICKETS PRODUCT-PICK-UP PROJECT-INVOICE-GENERATION
Error-conditions:	When the quantity on hand is not sufficient to satisfy all orders, a request for stocking up is issued (D17) and the order is queued (D18) for future fulfillment.

Figure 5. Task form filled for subtask ORDER-PICKING.

DOCUMENT \ TASK	ENTER-NEW-PROJ	ENTER-GRANT-ORDER	ORDER-PICKING
D1 - PROJECT-RECORD	0				
D2 - NEW-PROJ-RECORD	1				
D3 - ORDER-FORM		1			
D4 - WEEKLY-ORDER-HISTORY		0			
D5 - DAILY-ORDER-BATCH	0		1		
D6 - PROJ-BALANCE		1			

Figure 6. Task-document usage matrix. / and O represent input and output, respectively.

very similar to Chen's entity-relationship (E-R) model.¹⁰ The second level of syntax is more detailed and is needed for formal data definition. Both levels have a graphical representation that helps designers to visualize the underlying semantic connections among conceptual objects. In addition, the language's computer implementation provides a number of support facilities, such as data dictionary facilities, display facilities, and zoom-in/zoom-out facilities, which allow the designer to display the conceptual schema at different levels of detail. Such facilities are important in modeling large and complex systems, where unassisted manual modeling is tedious and error prone. A detailed description of CSDL is given elsewhere.¹³

Other modeling languages and/or modeling methodologies are suitable for this phase. The most notable are Taxis,¹⁴ D-graphs,²⁸ semantic database model (SDM),²⁹ E-R model,¹⁰ aggregation-generalization,³² conceptual graphs,²⁹ and object-role model.³¹ This list is by no means exhaustive; many other models can also be used.

Discovering entities, properties, and relationships. Conceptual data modeling consists of identifying the *entities*, the *properties*, and the *relationships* (called EPRs hereafter) in the application. In general, we have no algorithmic methods for identifying EPRs, but we can use some rules of thumb.

In the technique described here, we discover EPRs merely by examining nouns, adjectives, and verbs in the system's requirements. Nouns usually correspond to entities, verbs to relations, and adjectives to properties. The advantage of this technique is that the entities and relationships discovered in this way have meaning for the user because they draw on concepts from the application world. The disadvantage is that the naming process is very subjective, and different designers may come up with different names for the same entities or relationships. Other difficulties are the well-known dilemma of whether we should use a relationship "ship," the entity "shipment," or both to model the concept of shipping. The same difficulties are encountered in all similar techniques. A somewhat structured way for discovering EPRs is to use the document/data element specification as input (see Figures 6 and 7). The method can be used to discover the following:

- *Unique code identifiers* are usually introduced when organizations refer repeatedly to instances of entities (physical or abstract objects). In most cases, these identifiers uniquely identify the objects even though their names are not unique. For example, PROJ-NO and ORDER-NO denote entities PROJECT and ORDER.
- *Properties* can be discovered by asking the question: "Does element *A* characterize *B*?" If so, then *B* is usually an entity, and *A* is a property of *B*. For example, ADDRESS characterizes PROJECT; NAME characterizes PROJECT, etc.
- *Structural relationships (part-of relationships)* are discovered by asking the question: "Is element *D* a component of *C*?" If so, *C* is usually an entity, and an is-part-of relationship is identified. For example, LINE is-part-of ORDER. *D* can be either an entity by itself or a property of *C*. MODEL-NO, PRODUCT-NO, and QTY-ORD are all related to LINE by is-part-of relationships.
- *Is-a (-kind) relationships* are discovered by classifying data elements into generic types of entities; SHIP-LOCATION is-a LOCATION, DATE-ORDERED is-a DATE, etc. Both partners in these relationships are entities.
- *Events and actions* that involve data elements indicate general relationships among entities. For example, the event ORDER-PLACEMENT indicates a relationship between PROJECT and ORDER; similarly, the event FULFILLED-ORDERS indicates a relationship among the entities PROJECT, ORDER, SHIP-LOCATION, and DATE.
- *Numerical dependencies* of the entities in relationships are important; we must discover whether a relationship is one-one, one-many, or many-many. For example, an ORDER may have multiple LINES but each LINE has one PROD-NO and one QTY-ORD.

If we apply the preceding rules to our sample enterprise, we get the following EPRs.

DATA ELEMENT	TASK	ENTER-NEW-PROJ	ENTER-GRANT-ORDER	ORDER-PICKING
PROJ-NO		0	10	0		
PROJ-NAME		10	1			
CRDT-CODE		0				
CRDT-LIMIT		0				
PROJ-MAIN-LOC-NO		10	1			
PROJ-BRANCH-LOC-NO		10	1			
PROJ-BRANCH-LOC-ADDR		10				

Figure 7. Task-data element usage matrix. I and O represent input and output, respectively.

entities	PROJECT	ORDER	LOCATION	SHIP-LOCATION	PRODUCT	DATE
properties	LOCATION	characterizes	PROJECT			
ADDRESS	characterizes	PROJECT				
PROJ-BALANCE	characterizes	PROJECT				
SHIP-LOCATION	characterizes	ORDER				
DATE-ENTERED	characterizes	ORDER				
DATE-ORD	characterizes	ORDER				
part-of-relationships	LINE	is-part-of	ORDER			
PROD-NO	is-part-of	LINE				
QTY-ORD	is-part-of	LINE				
is-a-kind relationships	SHIP-LOCATION	is-a	LOCATION			
DATE-ORD	is-a	DATE				
general relationships	ORDER-PLACEMENT	relates	ORDER, PROJECT, and SHIP-LOCATION			

FULFILLED-ORDERS relates

PROJECT, ORDER,
SHIP-LOCATION,
and DATE

Note that after we discover the EPRs, we need to define the dependencies between the data values. These could be one-many or many-many. We especially need to define data dependencies of relationships requiring numerical quantification and arbitrary constraints, for example, "Every agency must procure every product from at least two suppliers."

Expressing EPRs in a multiple-level language. EPRs can be graphically represented as shown in Figure 9. The numerical quantifiers, can take the form [m] = at least m, [m] = exactly m, [m] = at most m, and [all] = all instances.

These quantifiers are used to express $m-n$ relationships among entities and to define their constraints. For example, in Figure 9, the quantifiers on the line between ORDER-LINE and LINE show that each ORDER has at least one LINE. Further, each LINE is in one-one correspondence with PROD-NO and QTY-ORD through the LINE-PROD-NO-QTY-ORD relationship.

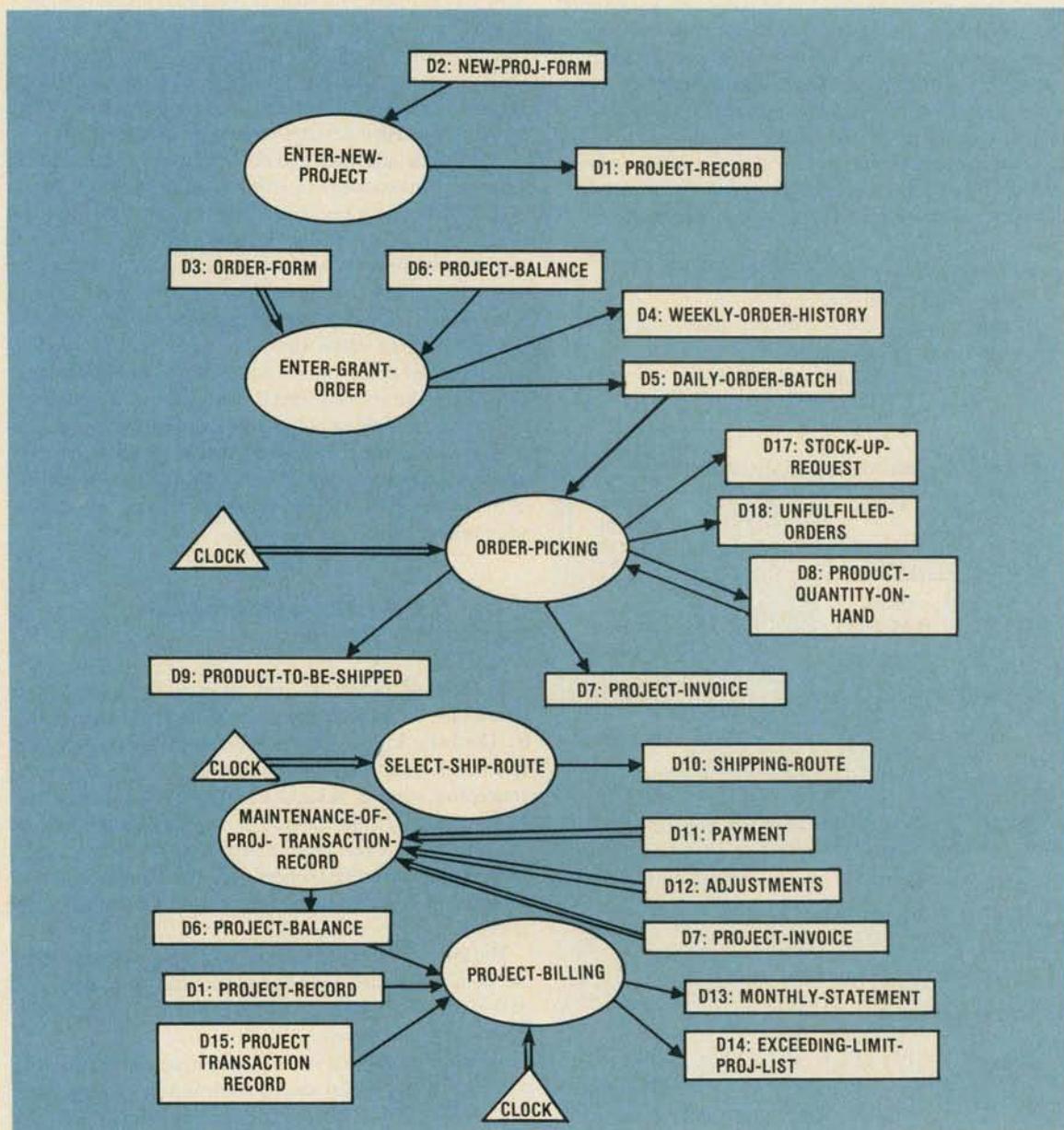


Figure 8. A task flow diagram for the Distribution activity. A triangle represents the clock, and any connected task is activated at a specific time and not by data passed to it by another task. Single-line arrows denote data flow, while double-line arrows denote control flow. For example, the single arrow between D5: DAILY-ORDER-BATCH and ORDER-PICKING means that D5 is used to carry information to the task, but does not activate it. The task, when activated by some other means, uses D5 to carry out its job. The task MAINTENANCE-OF-PROJ-TRANSACTION-RECORD, however, is activated whenever D7, D11, or D12 is passed to it.

This level of syntax is an extended version of Chen's E-R model,¹⁰ the difference being the addition of numerical quantifiers. The implication of no numerical quantifier is that any $m\text{-}n$ relationship $m \geq 1$ and $n \geq 1$, can be satisfied between the related values. However, the first level of syntax cannot express arbitrary quantified expressions because the scope of the quantifiers is not represented. For these types of relationships, we must use a language equivalent to first-order logic. Nevertheless, the preceding notation can express most common data relationships.

The second level of syntax specifies the details of the rectangles (relationships) and ellipses (entities). The second-level syntax of the conceptual schema definition language for some of the previously described EPRs is presented below. This level can be used to specify arbitrary quantified expressions and to provide data dictionary facilities. Every entity, property, and relationship must be defined by a "definitional frame," which fully specifies the system's understanding of the entity, property, or relationship. The keyword *concept* is used to define an entity, and the keyword *frame* to define properties and relationships.

Concept PROJECT (x) primary denoted-by PROJ-NO

```
frame PROJ-DEFINITION
  [x] of-type INTEGER is-defined-by 10000 ≤x≤99999]
frame PROJECT-PROPERTIES
  [x of-type PROJECT has-property:
    [1] y of-type NAME]
```

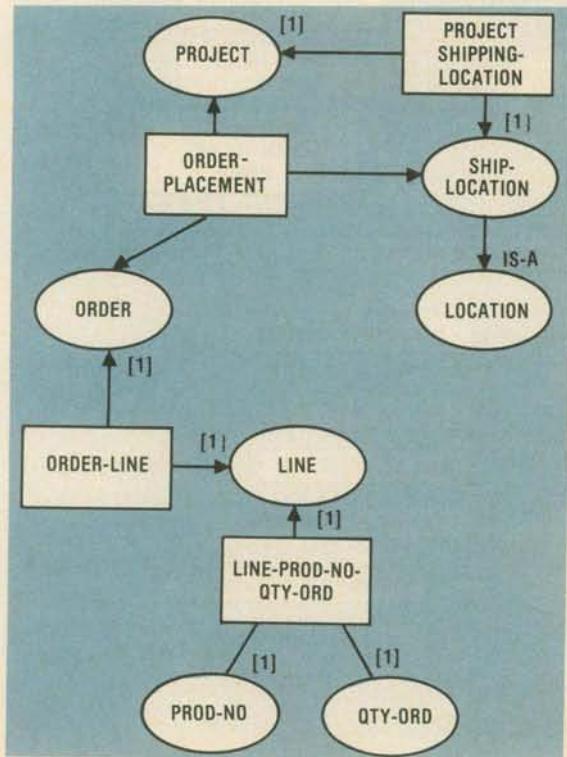


Figure 9. Graphical representation of entities, properties, and relationships, or EPRs, where circles represent entities and rectangles represent relationships and where [1] means at least one and and [1] means exactly one EPR.

```
[1] z of-type LOCATION-NO
[1] w of-type ADDRESS
[1] u of-type PROJ-BUDGET-CODE
[1] v of-type PROJ-BUDGET-LIM
[1] r of-type PROJ-BALANCE
[1] s of-type SHIP-LOCATION-NO
[1] t of-type SHIP-ADDRESS]
```

```
frame ORDER-PLACEMENT
  [all] x of-type PROJ-NO
  [all] y of-type ORDER-NO
  [1] z of-type SHIP-LOCATION;
order (agent:x, object:y, destination:z)]
```

A simple *frame* provides scoping rules for the qualified expressions. Composite *frames* are built by applying standard logical operations such as conjunctions and disjunctions. Typed variables such as *x,y,z*, are variables whose type is explicitly specified. For example, *y* of-type ADDRESS denotes that *y* is a variable of type ADDRESS. The numerical quantifiers are the same as those described earlier. (A detailed description is given elsewhere.¹³)

Other data languages and data dictionaries can be used in place of the conceptual schema design language. However, CSDL offers two major advantages. First, it combines both a data modeling language and data dictionary report capabilities in the same notation. Second, the two levels of syntax allow us to view the conceptual schema specification from different perspectives: the abstract level that provides an easier but more superficial perspective and the expert level, which is equivalent to first-order logic languages. The abstract level can be used by the nonexpert to express concepts and ideas, while the expert level can be used to define all the required details precisely. A new abstract level can then be obtained by masking the details of the second level and used again to verify the conceptual schema developed by the experts.

Phase 4: Derivation of the logical access path schema

In the final phase, we first generate the basic logical schema for the target system's database and then optimize it. The basic logical schema must explicitly or implicitly cover all logical relationships expressed or implied by the conceptual schema. A sequence of database accesses used to generate all (either explicit or implicit) relationships is called a *logical access path*. The collection of many LAPs integrated into a unified schema that combines shared sub-paths by adding their usage frequencies is called the *LAP schema*.²⁰

The input for this phase consists of the conceptual schema obtained by the previous phase, task specifications, document specifications, and usage matrices generated by phase 2, system analysis.

From the conceptual schema, we extract the records and the logical relationships among them and express them in a data model. These relationships correspond to the relations of the basic logical schema, which we call the *enterprise logical base schema*. Next, we generate the I/O data requirements of each task by writing queries against the ELBS. The data input requirements of a task are the queries; the data output requirements are either reports or updates to the database. The LAPs of these queries are col-

lected and integrated into a schema, which is then optimized for maximum efficiency on both retrieval and updates.¹⁹

The output of this phase consists of the LAP schema to be maintained; the task emulations, which are at this point the application programs of the database; and the constraints that must be considered to maintain data integrity.

Mapping to the enterprise logical base schema. Our objective here is to map the conceptual model onto other data models, such as the relational or Codasyl models supported by commercial database management systems. We can then use the resultant ELBS to define the logical access paths of each task by emulating the tasks against the ELBS.

The input to this step is the conceptual schema, expressed in the graphical notation described previously or in equivalent notation. The mapping to the relational data model is given below. A similar one has been done for the Codasyl data model.³⁶ During this mapping, we need not be concerned with issues of efficiency or redundancy. The efficiency and the amount of redundancy allowed in the obtained logical schema will be taken into account later during optimization.

Entities. For every entity, a one-column table (unary relation) is constructed. The name given to the relation is the same as the name of the entity. The attribute name is either the same as the entity name or the same as its database denotation (such as PROJ-NO and SUPPLIER-NO, if such denotation is available). In Figure 10, for example, the entity PROJECT (10a), is mapped onto the unary relation PROJECT (10b). INTEGER is the domain over which attribute PROJ-NO of relation PROJECT ranges. Thus, we have an explicit representation of the domain.

Properties and relationships. Most relationships expressed in the conceptual schema are mapped onto other relations. Property relationships are mapped onto two-column (binary) relations. The name of such a relation is the concatenation of the concept and property names. The attribute names used are PROPERTY-OF and VALUE. Take for example, the mapping of PROJECT-PROPERTIES in the CDSL second-level syntax.

Two things should be noted here. First is that the dependencies (one-one, one-many, or many-many) are constraints, so they are explicitly written in the attribute box of the relations. The quantifier [1] denotes that there is exactly one value of attribute VALUE for every value of attribute PROPERTY-OF. PROPERTY-OF and VALUE range over the domains PROJ-NO and NAME (or LOCATION-NO, ADDRESS), respectively. Second, whenever a concept is used to define a property and this concept has some denotation, this system denotation is used in the domain box of the relations; in Figure 11, LOCATION-NO is used in the domains of PROJECT-LOCATION rather than LOCATION.

For some property relationships, we are interested only in the *intension* and not the *extension*. The intension of a relationship is expressed using symbols and/or properties that define the elements of the relationship. The extension, on the other hand, is given in terms of the actual values

satisfying it. A relationship defined by intension can be found in the frame PROJECT-DEFINITION

$$10001 \leq \text{PROJ-NO}$$

$$\text{PROJ-NO} \leq 99999.$$

One of its extensions is

$$10001 \leq \text{PROJ-NO} = 10001 \leq 99999$$

$$10001 \leq \text{PROJ-NO} = 10002 \leq 99999$$

$$10001 \leq \text{PROJ-NO} = 10003 \leq 99999.$$

Clearly, for most applications we would not be interested in storing such extensions. Instead we would like to include the intensions only and use them as the logical schema integrity constraints, which are activated to validate every new instance of PROJ-NO that enters the system.

Another type of relationship in the conceptual schema is expressed in the form of *procedural attachments* of frames. For example, the "assign" procedural attachment in the PROJECT-DEFINITION frame represents a procedure followed by the DISTRIBUTION-MANAGER in assigning a PROJ-NO to a new project. In this case, the manager examines the characteristics of the new project so

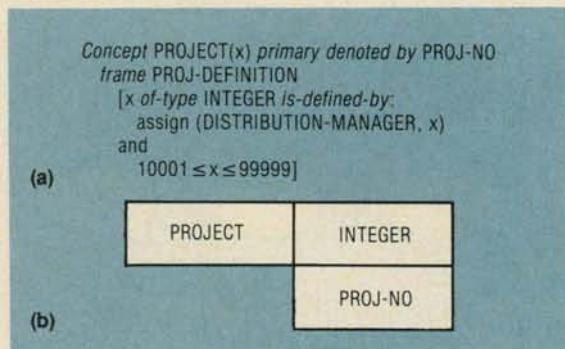


Figure 10. Mapping an entity.

Figure 11 shows three tables mapping property relationships:
 - PROJECT-NAME: Columns PROJECT-NAME, PROJ-NO, and NAME. PROJ-NO is the primary key. NAME is [1] VALUE. Data: (PROJECT-NAME, PROJ-NO, NAME)
 - 10001, watergate
 - 10002, eggplant
 - ...
 - PROJECT-LOCATION: Columns PROJECT-LOCATION, PROJ-NO, and LOCATION-NO. PROJ-NO is the primary key. LOCATION-NO is [1] VALUE. Data: (PROJECT-LOCATION, PROJ-NO, LOCATION-NO)
 - 10001, 21742
 - 10002, 95193
 - ...
 - PROJECT-ADDRESS: Columns PROJECT-ADDRESS, PROJ-NO, and ADDRESS. PROJ-NO is the primary key. ADDRESS is [1] VALUE. Data: (PROJECT-ADDRESS, PROJ-NO, ADDRESS)
 - 10001, white house
 - 10002, monterey
 - ...

Figure 11. Mapping a property relationship.

```

frame ORDER-PLACEMENT (x,y,z)
[[all] x of-type PROJ-NO
[all] y of-type ORDER-NO
[1] z of-type SHIP-LOCATION :
order(agent:x, object:y, destination:z) ]

```

ORDER-PLACEMENT	PROJ-NO	ORDER-NO	SHIP-LOCATION-NO
	[all] AGENT	[all] OBJECT	[1] DESTINATION
	10001	OM-336	21742
	10001	OS-112	21742
	10002	OM-348	95193
	.	.	.
	.	.	.

Figure 12. Mapping an event relationship.

that he can assign the PROJ-NO. Because these characteristics are of no interest to the system or to the system's users, they are hidden. Such assignments may be done, for example, on the basis of the information about project location, priority, and budget limits—notions that will not be needed again by system users.

The preceding discussion has focused on property relationships. Another kind of relationship expressed in the conceptual schema is the *event relationship*. These relations are mapped onto n -column relations. The name of the event is used as the relation name. The domains of the relation are the entities that participate in the relationship. Attribute names are chosen from a relatively small set of "roles" such as agent, recipient, object, source, destination, and time, which define the part each entity plays in the relationship. Figure 12a is an example of the event relationship ORDER-PLACEMENT which is mapped onto a synonymous relation (12b).

Emulating tasks. In this step, we obtain a query and an update model, which respectively reflect the data input and output requirements of each task. The result of the task emulation step is a collection of logical local subschemata, each of which is tailored to the data I/O requirements of a task.

Each task is emulated using the ELBS generated by the previous step. In this emulation, the data input required by each task (see the "input" line in the task form in Figure 3) is expressed by queries against the ELBS. For example, if information is needed about a project's location, a query is made against the PROJECT-LOCATION relation.

The term "query" implies a well-formed expression in a relational query language. However, since every query in a relational query language can be expressed by a relational algebraic expression, we assume that a query is a valid relational expression. Some queries are *generic*; that is, they produce a set of values, each of which may be the answer to a simple query. For example, a query that specifies the retrieval of projects located in New York generates as an answer a set of projects (possibly empty). This result may, in turn, be accessed by another application program to process each of the New York projects individually.

The answers to generic queries, called *views*, are, in general, virtual relations that have a form similar to a base relation except that they may be stored differently. The collection of these virtual relations, which corresponds to the data input of a task, is nothing but a local logical subschema tailored to the needs of this task.

Updates are generated from the output specifications of the task and are expressed in a data manipulation language. Only updates to the *base relations* of the enterprise logical base schema are allowed. Base relations are those that cannot be derived by queries made against other relations. Updates to views are in general problematic because some of these updates may not reflect valid updates to the base relations.³⁷⁻³⁹ Since the only view updates that can be handled correctly are those that can be translated into sets of updates to base relations, we consider only base relation updates. The queries and the updates obtained during each task's emulation and their usage frequencies are used to construct the query and update model, which is then used to optimize the collection of logical access paths. In

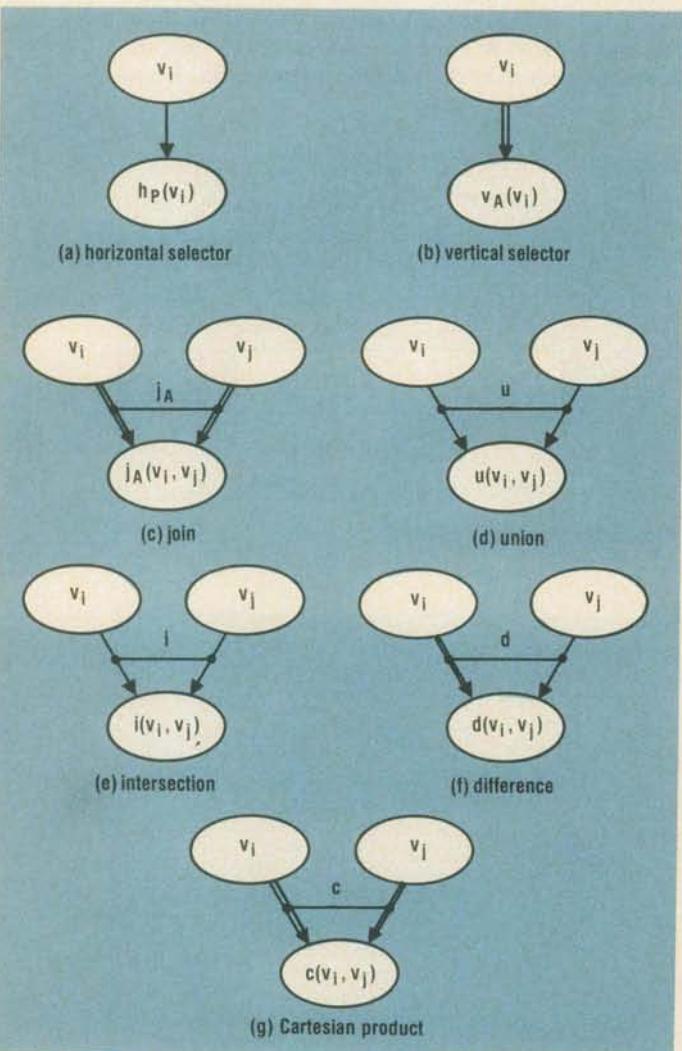


Figure 13. Graphical notation for the relational operators: (a) horizontal selector, (b) vertical selector, (c) join, (d) union, (e) intersection, (f) difference, and (g) Cartesian product. Single-line arrows (arcs) denote "subset" relations; that is, the head node has tuples that are included in the tail node. Double-line arrows denote "project" relations; that is, the head node has as attributes a subset of the attributes of the tail node. Links between arrows represent the derivations of views (dots).

other words, the query and the update models are the composition of the data input and output requirements of all the tasks. The task emulation step is divided into the following substeps: (1) generate the data input requirements of each task, (2) compute the probabilities of all queries, and (3) compute the update probabilities for the base relations.

Generate the data input requirements of each task. Each query made to generate data input requirements of a task is represented by a *query graph* composed of view nodes. A view node (or simply a view) is a representation of the answer that corresponds to a relation derived by a query. For example, the virtual relation EXCEEDING-BUDGET-PROJECT is a view node that corresponds to the answer of the query "Give me the projects whose balance exceeds their budget limit."

A graphical representation for query graphs is introduced (Figure 13) in which each relational operator is represented by one or two linked arcs (depending on whether the operator is unary or binary). Any relational query can be represented by such a query graph because this set includes all relational operators. Figure 14 is a query graph for local projects. Here, LOCAL-PROJECT is a view obtained from PROJECT of ELBS by selecting projects that are local (LOCAL = LOCATION = Washington, DC).

Compute the probabilities of query graphs. The global probability of a query graph is the probability of executing the corresponding query against the database from within any task. These computed probabilities are then used during optimization, a step required by any optimization algorithm.^{18,19,40}

The technique for computing global capabilities uses the frequency of each task found in the task forms and the number of times each query graph is executed from within each task. Weights indicating the importance of each task or application can also be used as an alternative or in addition.

Assume that there is a total of n tasks and m query graphs. We use a simple computation matrix for obtaining the global probabilities of all queries. Figure 15 is the task-query frequency matrix, where the upper $m \times n$ portion contains the frequencies of the query graphs within the tasks. For example, a_{ji} (where $1 \leq j \leq m$ and $1 \leq i \leq n$) is the number of times the query graph q_j is executed from within task t_i . If q_j is never made from within t_i , then the corresponding entry is zero. The $(m+1)$ row, shown in the matrix as

$$\sum_{j=1}^m a_{ji}$$

is used to store the total number of query graphs made by task t_i . The $(m+2)$ row (labeled F_{t_i}) contains the frequencies of the task's activations. The $(n+1)$ column with label p_j stores the computed global probabilities of each query graph. The following algorithm²⁰ computes the query graph probabilities by filling out the matrix in Figure 15.

```

BEGIN
  FOR i=1 UNTIL n DO
    BEGIN
      qnum=0
      FOR j=1 UNTIL m DO qnum=qnum+aji
      am+1,i=qnum
    END
    ftot=0
    FOR j=1 UNTIL n DO ftot=ftot+am+2,i
    FOR j=1 UNTIL m DO
      BEGIN
        psum=0
        BEGIN
          FOR i=1 UNTIL n DO
            psum=psum+(aji + am+1,i)*(am+2,i / ftot)
        END
        aj,n+1=psum
      END
    END
  END

```

The first double loop in the algorithm fills out the $(m+1)$ row. The second simple loop computes the total number of tasks f_{tot} . The last double loop computes the global probability p_{sum} of each query q_j .

Figure 16 is the result of applying the algorithm to the matrix in Figure 15. Note that the probabilities were computed as if there were no updates against the database. In real life, of course, queries and updates are intermixed activities of each database system. Thus, the global probabilities should be computed so that the updates are also taken into account. To account for the influence of the updates onto the probabilities of the queries, we can add to the $(m+1)$ row the number of updates done from

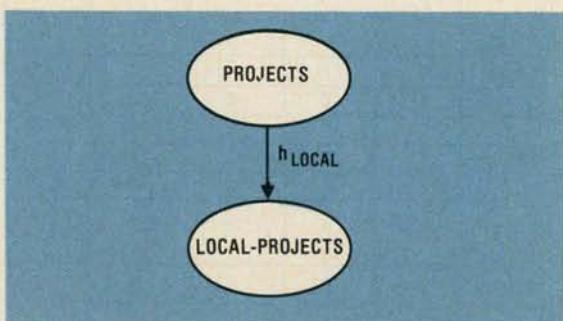


Figure 14. A query graph for local projects.

$q_1 \backslash t_i$	t_1	t_2	t_3	t_4	p_j
q_1	0	2	1	1	
q_2	1	0	0	0	
q_3	0	1	0	0	
q_4	1	1	1	0	
q_5	2	0	0	0	
$\sum_{j=1}^m a_{ji}$					
F_{t_i}	1	2	1	3	7

Figure 15. A task-query frequency matrix in which $(q_1 \dots q_5) \times (t_1 \dots t_5)$ contains the frequencies of query graphs within the tasks. The remainder of the matrix is used to store the total number of graphs made by task t_i and how often the task is activated.

within each task, assuming that the cost of retrievals and updates is about the same. Or, before we add anything to the $(m+1)$ row, we may want to multiply the number of updates by a weight, depending on the cost of the updates when they are compared with the retrieval queries.

Generate update probabilities. We can get the probabilities of updates on base relations from task output. We are not considering updates against views because studies show that only a few view updates can be reflected into valid sequences of valid updates on the base relations used to derive the updated views.³⁷⁻³⁹ We assume here that all updates considered correspond to valid updates on base relations and that they have been translated onto base relation updates. The update probabilities u_j (where $j=1,2,\dots,k$) of base relations are computed in a manner similar to that for queries. Figure 17 is an example of a probability computation matrix for updates. It differs from the query matrix in that the first column of this matrix corresponds to base relations r_j rather than to views. For example, in Figure 17, task t_1 makes two updates: one on the base relation r_1 and one on r_3 . Task t_3 makes two updates on r_2 . Note that the task frequencies for this example are the same as the ones on the query computation matrix. The same algorithm described previously is used to fill out the rest of the computation matrix as shown in Figure 18.

$q_i \setminus t_j$	t_1	t_2	t_3	t_4	p_j
q_1	0	2	1	1	0.641
q_2	1	0	0	0	0.035
q_3	0	1	0	0	0.071
q_4	1	1	1	0	0.177
q_5	2	0	0	0	0.071
$\sum_{j=1}^m a_{ji}$	4	4	2	1	
F_{t_i}	1	2	1	3	7

Figure 16. Figure 15 matrix after applying an algorithm to compute query graph probability.

\setminus TASKS BASE REL.	t_1	t_2	t_3	t_4	q_j
r_1	1	0	0	1	0.21
r_2	0	0	2	0	0.14
r_3	1	0	0	1	0.21
r_4	0	0	0	0	0
r_5	0	1	0	1	0.42
$\sum_{j=1}^m a_{ij}$	2	1	2	3	
F_{t_i}	1	2	1	3	7

Figure 17. Probability computation matrix for updates in which the base relations are the first column rather than views (Figures 15 and 16).

Integrating query graphs into a LAP schema. In this integration step, we use the query graphs generated by the task emulation, the global probabilities, and the update probabilities to generate a logical access path schema that models the use of LAPs in the database. This schema is based on database activities, (retrievals and updates) found in the tasks, and integrates their logical access paths by recognizing common subpaths and increasing the weight of the shared subpaths. The schema provides a comprehensive picture of LAPs and the cumulative use of the shared subpaths and/or intermediate results. It is both a model of the access requirements during database design and a model for optimization during database operation.

Integration is achieved when two query graphs are merged to form a single one. During this merging, a simple algebra is defined that adds the probabilities of the common views on the obtained LAP schema. When the first two graphs have been merged, a third one is merged with the first two, and so on until eventually all query graphs have been merged into the final LAP schema. (More rigorous descriptions of merging algorithms are given elsewhere.^{20,41}) Figure 18 shows query graphs for (a) EXCEEDING-BUDGET-PROJECTS, (b) SHIPPED-ORDERS, and (c) ORDERS-IN-PROCESS. Since 18a and 18b have no common views, they remain disconnected after merging. Figure 19 shows the merging of Figures 18a, b, and c. Note that the query probabilities of the common view ORDERS have been added. In Figure 20, the query graph for “Orders in process for projects exceeding their budgets” is shown.¹ The graph is obtained by making (1) two horizontal selections on PROJECTS and ORDERS to obtain EXCEEDING-BUDGET-PROJECTS and ORDERS-IN-PROCESS and (2) a join on the last two views on the attribute PROJ-NO. When we merge this query graph with Figure 19, we get the graph in Figure 21. As the figure shows, the probabilities of accessing PROJECTS and EXCEEDING-BUDGET-PROJECTS have been increased because they appear in a number of query graphs. The more each view appears in query graphs, the higher its probability of being supported in the optimal schema becomes.

A view (which does not correspond to a base relation) may be affected by an update to any of the base relations used in the derivation of the view. Since we assume that the updates on base relations are independent, the update probability u that a view may be affected by any update is the sum of the update probabilities of all base relations used in the derivation of the view. In Figure 21 we show the update probabilities u_n as they propagate from the base relations to the views. The view ORDERS-IN-PROCESS-FOR-PROJECTS-EXCEEDING-BUDGET is affected by both updates in PROJECT and ORDERS and thus its u probability is $u_1 + u_2$. The graph obtained by this step along with the retrieval and update probabilities is the LAP schema of the database.

Optimizing the LAP schema. The goal of this step is to enhance the execution efficiency of the frequently used (or important) tasks by explicitly indexing the subschemata they operate on. Clearly, not all of the subschemata

should be indexed because a great deal of redundancy is generated, costing both storage and update time.

A view in the LAP schema can either be supported by an index to the base relation tuples that make up the view's tuples or constructed on demand from the base relations it is derived from. The construction is associated with I/O and CPU cost—to be called the “cons” cost. On the other hand, if an index is kept for the view, the cons cost is avoided. In this case, however, storage is required for the index, and updates become more time consuming because the index needs maintenance.

A useful indexing policy is to retain all indexes until an update is required. Consequently, an index is updated if it corresponds to a frequently accessed view but dropped if the view is infrequently accessed. In this manner, existing indexes are used as cache memory aids. These aids do not cost much overhead because they are created to answer queries. The only overhead is for the recognition of such cache indexes. The payoff can be significant because we trade CPU time (required for the recognition/matching algorithm) with extra I/O (required by the construction cost of a view). Given that (1) certain logical access paths among data expressed in some subschemata are used more often than others, and (2) in general not all paths should be indexed, we clearly need an optimization process that decides which paths must be supported.

The optimization is based on the following elements:

- the set of the derived views;
- the sizes of the indexes for those views, when the views are explicitly supported (cardinality times the size of the pointer);
- the construction cost of the derived views assuming that their index is not available;
- the probabilities of making queries against those derived views;
- the update cost of the indexes; and
- the probabilities of the updates.

Estimates of these elements could be easily obtained during the design in terms of upper bounds and complexity order of the algorithms. They are even more easily obtained when the database is operational. In the latter case, the index size and the construction cost of the index can be measured by monitoring the execution of the views. The same is true for the update cost, which can be measured by executing the updates. Optimization would then be based on actual costs of the index mechanism's implementation, the operating system, the database system, the buffering facilities, and all the other influencing parameters of the execution environment.

A cost model is needed to give us the cost of querying views and updating indexes. First we define index allocation and the construction cost based on this allocation. An index allocation A over a set of views V is a mapping

$$A : V \rightarrow \{0, 1\}$$

where $A(v) = 1$ means that view v has been indexed, whereas $A(v) = 0$ means the opposite. For simplicity, we treat base relations as views (even though they are explicitly stored) and assume that $A(r) = 1$ for all base relations r . A construction cost function cons over a set of views V is a mapping

$$\text{cons} : (f, v) \rightarrow [0, \infty]$$

where $v \in V$ is a derived view and f is the set of views or base relations from which v can be constructed. Clearly, more than one set f can be used to construct a given view. Note also that a base relation has no f set because by definition base relations cannot be constructed from others.

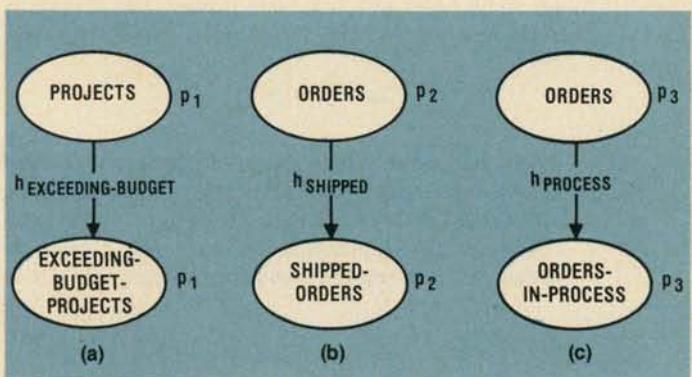


Figure 18. Query graphs for (a) EXCEEDING-BUDGET-PROJECTS, (b) SHIPPED ORDERS, and (c) ORDERS-IN-PROCESS.

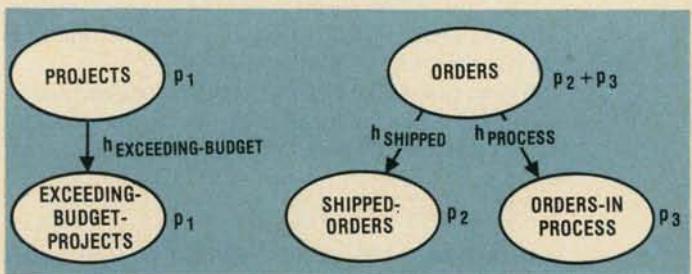


Figure 19. Merging of the three graphs in Figure 18. Note that 18a remains disconnected from the others because it has no views in common with 18b and c.

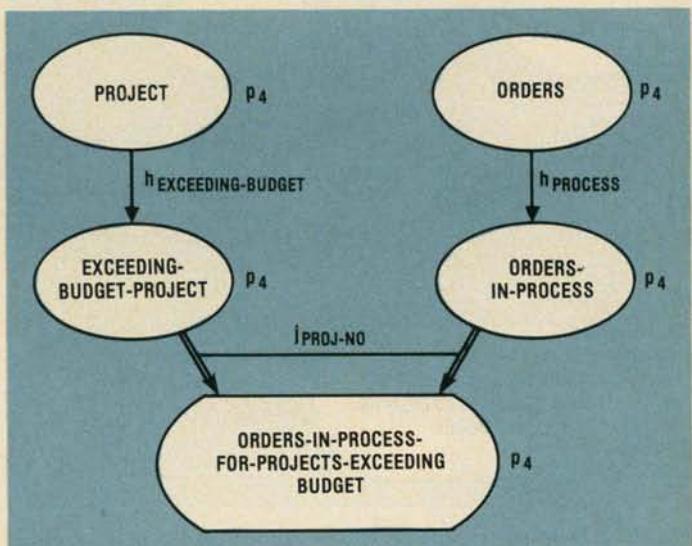


Figure 20. Query graph for “orders in process for projects exceeding their budgets.”

The cost of answering a query that uses view v , given index allocation A , is

$$C(v) = c_r(v) \quad \text{if } A(v) = 1$$

or

$$C(v) = \min_f [cons(f, v) + \sum_{x \in f} C(x)] \quad \text{if } A(v) = 0$$

where $C_r(v)$ is the cost of retrieving the view from its index and is linear with size $|v|$. If $cons(f, v)$ is the cost of constructing the index for v from its ancestors f , then quantity

$$cons(f, v) + \sum_{x \in f} C(x)$$

reflects the cost required to first obtain all elements x in f needed to construct v plus the construction cost of putting together the elements of f to construct v .

The cost of maintaining an index for view v during an update of a base relation used in constructing this view is

$$UC(v) = c(v) \quad \text{if } A(v) = 1$$

or

$$UC(v) = 0 \quad \text{if } A(v) = 0$$

This cost reflects the cost c_r of retrieving and updating the view's index, if one is maintained for it; otherwise, it is zero.

Both costs reflect results in term of I/O operations of retrieving a view or modifying it. Similarly CPU time costs can be used for CPU-bound systems.

We can now define different optimization functions:

(1) Find the best index allocation A on the views such that the total cost of answering all queries is minimal. That is

$$\sum [C(v)p_v + UC(v)u_v]$$

is minimal subject to the constraint that the total storage used for indexing does not exceed SLIMIT. That is

$$\sum |v| A(v) \leq \text{SLIMIT}$$

In the previous formula, p_v is the probability of accessing v , and u_v the probability that v needs to be maintained because of an update for a base relation.

(2) Find the best index allocation A such that for weights w_1 and w_2

$$\sum [w_1 C(v)p_v + w_2 UC(v)u_v]$$

is minimal

(3) Find the best index allocation A such that

$$\sum A(v) |v|$$

is minimal subject to the constraint that no query takes more than TLIMIT time to answer; that is

$$\sum C(v) \leq \text{TLIMIT}$$

If I/O operation cost is used, TLIMIT can be estimated by multiplying the cost by the time in milliseconds to do one I/O operation.

(4) Find the best allocation A such that

$$\sum C(v)p_v$$

is minimal,

$$\sum UC(v)u_v$$

is minimal, and

$$\sum A(v) |v| \leq \text{SLIMIT}$$

Algorithms for obtaining the best index allocation for any of these optimization functions are computationally very difficult (NP-complete⁴²). However, heuristics and knowledge on the accessing provided by the logical access path schema can help optimization. A heuristically directed algorithm, which guarantees that whenever a solution is reached, it is the optimal allocation for the LAP schema, is described elsewhere.¹⁹

We have now completed the last phase of the database design methodology. At this point, we have obtained the optimized logical access path schema and the application programs (task emulations). Now the operational phase of the database can begin.

We have presented a comprehensive step-by-step methodology for an adaptable database design—adaptable because each phase can be facilitated by a number of models and representation primitives.

Many of the tools and analysis techniques in the methodology can be and have been automated. Others, such as tools that help us to analyze objectives, constraints, policies, etc., or formal verifiers for consistency, are more difficult to automate. To help solve these problems, we must rely on user-directed instead of system-directed walkthroughs and/or specialized presentation techniques and reports.

The design specification techniques used in this methodology are valuable for system maintenance, as well. Changes in the environment and/or requirements can be traced back to the specification (output) of each phase to see the effects of these changes. Efficiency improvement from statistical observations and/or changes in access usage can be handled when deriving and optimizing a logical access path schema.

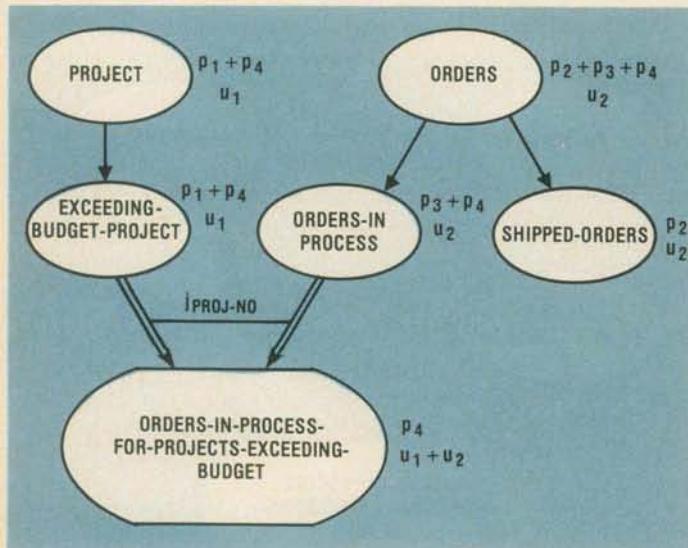


Figure 21. Merging of graphs in Figure 19 and Figure 20. Note that the probability of accessing PROJECTS and EXCEEDING-BUDGET-PROJECTS has increased because each appears in a number of query graphs.

We have successfully used our methodology in a number of applications including the ABC Agency inventory database, Naval Printing Office database, a Telephone Answering Service Processing System, a Happy-Boozier Bar database and a database to support an IFIP conference. We found that the method is easy to teach to people because the designer has the freedom to use familiar models.

We have also been using the methodology in a class of approximately 50 students taking their first undergraduate database course. The students receive one to two lectures for each phase (squeezed between regular lectures.) For the course term project, the students teamed up in groups of two to find the information requirements of an organization, designed the database system, and implemented it on an available database management system. The only step of the methodology not required in completing this project was the optimization of the LAP schema. From the approximately 20 designs and implementations on each application, about five were excellent. Ten were good designs with complete requirements and design specifications.

One of our initial hypotheses was that task emulation would speed up implementation. The experiment with the students verified this assumption. Task emulation helps to ensure that the logical schema is complete, because if some parts of the schema are not covered, some tasks cannot be emulated. Also, each emulated task forms the skeleton of the application program (transaction) that is to carry out the task—the part that deals with the necessary I/O interactions with the database. Therefore, very little work is left to complete the implementation of the application program. *

Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research contract F49620-83-K0018-AFOSR to the University of Maryland. Computer support was provided in part by the Computer Science Center at the University of Maryland.

References

1. D. Jefferson, V. Lum, and D. Sheppard-Rund, "Corporate Requirements Analysis," *New Orleans Workshop Data Base Design*, 1978.
2. B. K. Kahn, "A Structured Logical Database Design Methodology," *Proc. NYU Symp. Database Design*, 1978, pp. 15-24.
3. D. C. Burnstine, "The Theory Behind BIAIT Business Information Analysis and Integration Technique," BIAIT International, 1979.
4. T. De Marco, *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
5. D. Teichroew and A. Hershey, III, "PSL/PSA, A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Trans. Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 71-79.
6. G. Myers, *Structured/Composite Design*, John Wiley & Sons, New York, 1979.

MACHINE VISION IMAGE PROCESSING RESEARCH AND DESIGN ENGINEERS

The Environmental Research Institute of Michigan (ERIM) has been a center for creative research and development for over thirty years. We've been a pacesetter in both image sensor and data processing techniques. ERIM scientists and engineers have invented/developed holography, 3-D laser scanning, synthetic aperture radar and Cyotocomputers.

New staff positions are available in machine vision, image processing, pattern recognition, algorithm development, intelligent machines and robotics. The work includes the design and development of real-time laser-based image sensors, image processing systems and alternative computer architectures for artificial intelligence.

The work environment is flexible and stimulating, and those interested in "just a job" should not apply. The most modern equipment is available and we stress individual growth, initiative and development. Leadership potential will be welcome. The following positions are a few of those available:

INTEGRATED CIRCUIT AND COMPUTER DESIGN ENGINEERS

BS in Electrical or Computer Engineering and experience in hands-on digital circuit design and construction required. MS desired. Specific experience should include high speed TTL or MOS circuit and systems design along with programming experience on mini/micro systems. MC 68000, industrial electronics packaging, video circuit design, gate array or integrated circuit design helpful.

ALGORITHM DEVELOPMENT AND SOFTWARE ENGINEERS

Candidates should have significant hands-on experience with invention, development and/or refinement of algorithms for real-time signal/image processing. Software design on mini-micro systems, preferably using PASCAL; MC 68000, C, VMS or UNIX would be helpful. A BS in Electrical or Computer Engineering or Computer Science is required. An MS or experience in image processing is desirable.

United States citizenship required.

Compensation includes an attractive starting salary and a very comprehensive fringe benefits package. Please send your resume to:

John J. Malik
Environmental Research
Institute of Michigan
P.O. Box 8618
Ann Arbor, Michigan 48107



7. E. Yourdon and L. Constantin, *Structured Design*, Prentice-Hall, Englewood Cliffs, N.J., 1979.
8. M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Trans. Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 60-69.
9. D. Sheppard-Rund, *Database Design Methodology—Parts I and II*, Auerbach Data Base Management Series, portfolios 23-01-01 and 02, 1977.
10. P. P. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Trans. Database Systems*, Vol. 1, No. 1, Mar. 1976, pp. 9-36.
11. J. R. Abrial, "Data Semantics," in *Database Management*, North-Holland, Amsterdam, 1974, pp. 1-59.
12. N. Roussopoulos, "A Semantic Network Model of Databases," Dept. of Computer Science, University of Toronto, TR-104, PhD dissertation, 1976.
13. N. Roussopoulos, "CSDL: A Conceptual Schema Definition Language for the Design of Data Base Applications," *IEEE Trans. Software Engineering*, Vol. SE-5, No. 5, Sept. 1979, pp. 481-496.
14. J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong, "A Language Facility for Designing Database-Intensive Applications," *ACM Trans. Database Systems*, Vol. 5, 1980, pp. 185-207.
15. M. R. Gustafsson, T. Karlsson, and J. A. Bubenko, "A Declarative Approach to Conceptual Information Modeling," Dept. of Information Processing & Computer Science, The University of Stockholm, Syslab report 8, 1982.
16. J. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, Mass., 1984.
17. M. Schkolnick, "Secondary Index Optimization," *Proc. ACM Sigmod*, 1975, pp. 186-192.
18. M. Schkolnick, "Physical Database Techniques," *NYU Symp. Database Design*, 1978.
19. N. Roussopoulos, "View Indexing in Relational Databases," *ACM Trans. Database Systems*, Vol. 7, No. 2, June 1982, pp. 258-290.
20. N. Roussopoulos, "The Logical Access Path Schema of a Database," *IEEE Trans. Software Engineering*, Vol. SE-8, No. 6, Nov. 1982, pp. 563-573.
21. J. A. Bubenko, Jr., "Information Modeling in the Context of System Development," *Proc. IFIP*, 1980, pp. 395-411.
22. *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen, ed., North-Holland, Amsterdam, 1980.
23. V. Y. Lum et al., "The 1978 New Orleans Data Base Design Workshop Report," *Proc. Fifth Int'l Conf. Very Large Data Bases*, ACM Press, New York, 1979, pp. 328-350.
24. D. C. Tsichritzis and F. H. Lockovsky, "Designing the Data Base," *Datamation*, Vol. 24, No. 28, 1978, pp. 147-151.
25. D. C. Tsichritzis and F. H. Lockovsky, *Data Models*, Prentice Hall, Englewood Cliffs, N.J., 1982.
26. N. Roussopoulos and J. Mylopoulos, "Using Semantic Networks for Database Management," *Proc. First Int'l Conf. Very Large Databases*, ACM Press, New York, 1975.
27. N. Roussopoulos, "ADD: Algebraic Data Definition," *Sixth Texas Conf. Computing Systems*, University Press, Austin, Tex., 1977.
28. H. Weber, The D-Graph Model of Large Shared Data Bases: A Representation of Integrity Constraints and Views as Abstract Data Types, IBM Research Report RJ1875, IBM Corp. San Jose, Calif., Nov. 1976.
29. M. Hammer and D. McLeod, "Database Description With SDM: A Semantic Database Model," *ACM Trans. Database Systems*, Vol. 6, No. 3, Sept. 1981.
30. J. Sowa, "Conceptual Graphs for a Database Interface," *IBM J. Research and Development*, Vol. 20, No. 4, July 1976.
31. E. Falkenberg, "Concepts for Modeling Information," in *Modeling in Data Base Management Systems*, G. M. Nijsen, ed., North-Holland, Amsterdam, 1976.
32. J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. Database Systems*, Vol. 2, No. 2, 1977.
33. M. L. Brodie, "The Application of Data Types to Database Semantic Integrity," *Information Systems*, Vol. 5, 1980, pp. 287-296.
34. S. K. Chang and W. H. Cheng, "Database Skeleton and Its Application to Logical Database Synthesis," *IEEE Trans. Software Engineering*, Vol. SE-4, No. 1, Jan. 1978, pp. 18-30.
35. S. K. Chang, and J. S. Ke, "Translation of Fuzzy Queries," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 3, July 1979.
36. N. Roussopoulos and R. T. Yeh, *Logical Database Design*, NBS final report, National Bureau of Standards, Washington, DC, Feb. 1982.
37. U. Dayal and P. A. Bernstein, "On the Updatability of Relational Views," *Proc. Fourth Int'l Conf. Very Large Databases*, ACM Press, New York, 1978.
38. F. M. Bancilhon and N. Spryatos, "Update Semantics of Relational Views," *ACM Trans. Database Systems*, Vol. 6, No. 4, Dec. 1981, pp. 557-575.
39. A. M. Keller, "Updates to Relational Databases Through Views Involving Joins," *Second Int'l Conf. Databases*, Academic Press, New York, 1982, pp. 363-384.
40. K. Y. Whang, G. Wiederhold, and D. Sagalowicz, "Separability: An Approach to Physical Database Design," *Proc. Seventh Int'l Conf. Very Large Databases*, ACM Press, New York, 1981, pp. 320-332.
41. R. Masri and G. Wiederhold, "Data Model Integration Using the Structural Model," *Proc. ACM-Sigmod Conf.* Bernstein, ed., 1979, pp. 191-202.
42. K. M. Chandy, "Models of Distributed Systems," *Proc. Third Int'l Conf. Very Large Databases*, ACM Press, New York, 1977, pp. 105-120.



Nicholas Roussopoulos is an assistant professor of computer science at the University of Maryland. His research interests include database design and conceptual modeling of information systems, distributed databases, software engineering, and artificial intelligence. He has held positions at the IBM Research Lab and the University of Texas at Austin. Roussopoulos received a BA in mathematics from the University of Athens, and an MS and PhD in computer science from the University of Toronto.



Raymond T. Yeh is professor of computer sciences at the University of Maryland. He has served as chairman of the Computer Science Departments at the Universities of Maryland and Texas at Austin. He was also director of the Center for Information Sciences Research at Maryland. Yeh received a BS in electrical engineering, an MA in mathematics, and a PhD in mathematics from the University of Illinois. He is the founding editor-in-chief of *IEEE Transactions on Software Engineering*.

Questions about this article can be addressed to either author, Dept. of Computer Sciences, University of Maryland, College Park, MD 20742.

3. Overview on the SDBD Method

A Database Design Methodology: Lecture Slides of Professor Roussopoulos based on the example ‘Analysis, Design and Implementation of the OlympiChroniclesDB System OLYMPICHRONICLES’.



A Database Design Methodology

N. Roussopoulos & R. Yeh
IEEE Computer 1984

1



A Complete Methodology

Area of application:

Design of database with its applications.

Perspective:

The method assumes that the primary purpose of the future system is to automate current or planned activities of the enterprise. The method assumes (as do all database design methodologies) that different views on the enterprise, conflicts, and political differences will be resolved during the database design process.

Life-Cycle:

■ Project Progress Report: Phase I

- Environment & Requirement Analysis
- System Analysis & Specification

■ Project Progress Report: Phase II

- Conceptual Modeling
- Logical Modeling
- Task Emulation
- Optimization (NOT REQUIRED for the 424 project)

■ Project Progress Report: Phase III

- Implementation
- 1 Convert Emulated tasks to code
- 2 Bulk-Loading & Tuning (LIMITED for the 424 project)
- 3 Testing

■ Limitation:

- The methodology does not cover implementation, testing, maintenance, and project management.

2

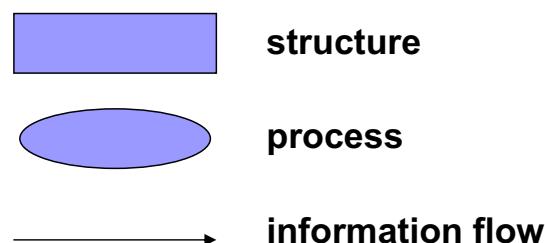
I.1. Environment & Requirements Analysis

- The purpose of this phase is to investigate the **information needs** of and the **activities** within the enterprise and determine the **boundary** of the design problem (not necessarily identical to the boundary of the future computerized system, if any).
- **Input:**
Information describing the current status of the enterprise, possible inefficiencies, plans for the future, and constraints that have to be satisfied in conducting business.
- **Output:**
A **Top-Level Information Flow Diagram** describing the major documents and functions, and the boundary of the design problem. **The documents** include the major input, output, and internal documents. **The functions** model the major activities within the enterprise.
- **Function:**
To collect the information about the enterprise and design the top-level information flow diagram.

3

Guidelines:

- **Techniques:** collect information by contacting interviews of people at all levels of the organization; analyze questionnaires; review short and long term plans, business annuals, files, forms, etc.
- **Tools:** express a top-level information flow diagram to capture the functions and important documents of the enterprise, and to start the design with the i/o documents and work from the outside in towards a "top-level" design.
- The tool we use for designing the top-level information flow diagram is the following **graphic formalism** for representing **structures and processes**:



- Two structures are **never** directly connected.
- Two processes are **never** directly connected.

4



Example



Analysis, Design and Implementation of the OlympiChronicles DB System OLYMPICHRONICLES

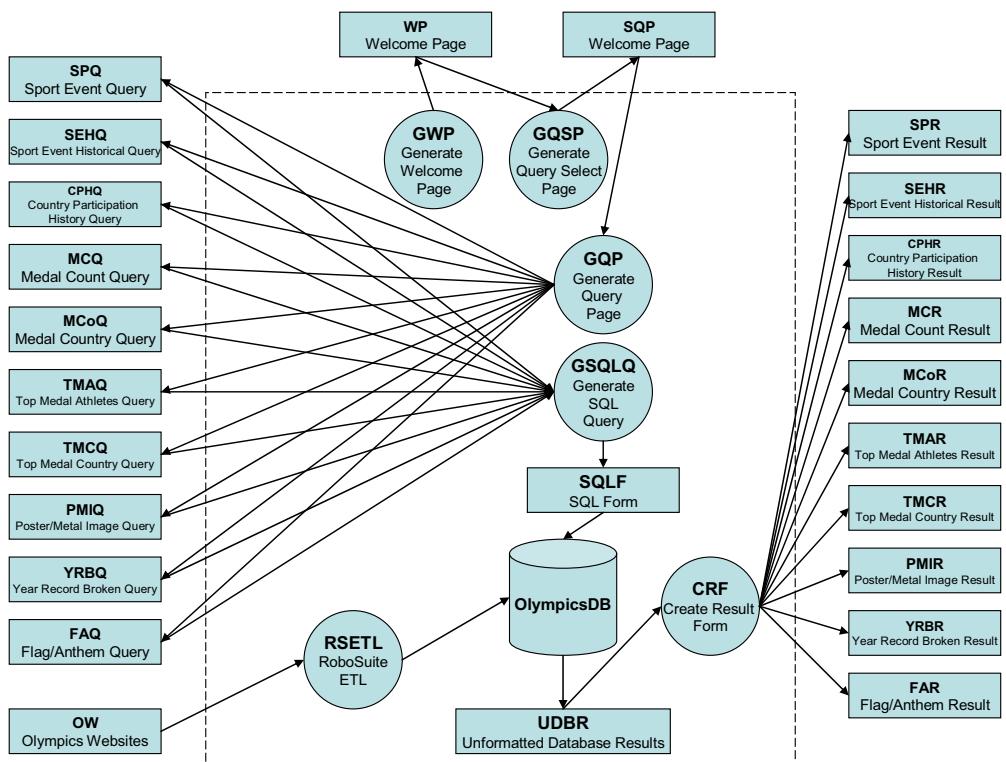


Craig Shapiro
Steffanie Orellana

5



Top Level Information Flow



6

I.2. System Analysis & Specification

The purpose is to divide the functions from the Top-Level Information Flow Diagram hierarchically into tasks. The tasks should be reasonably independent to minimize the task-to-task interfaces (documents). During the division process, the documents used by each function are also broken down. The process is continued until each task is small enough to be clearly understood, and until each document can be conveniently expressed in terms of data elements that cannot be further divided. The result is a detailed Task Flow Diagram and a set of forms describing the documents and the tasks.

- **Input:**
 - The Top-Level Information Flow Diagram and information about the documents and functions from step 1.1
- **Output:**
 - Task Forms; Document Forms; Document and Data Usage Matrices; and, the detailed Task Flow Diagram.
- **Function:**
 - Decompose functions and documents. Specify the resulting Task and component Document Forms. Specify Document and Data Usage Matrices. Design detailed Task Flow Diagram.
- **Guidelines:**
 - **Technique:** top-down hierarchical decomposition.
 - **Tools:** Task Forms; Document Forms; Usage Matrices; and the graphical formalism for Task Flow Diagrams.

7

Examples of Task Forms

3.2.2.2 ETL Task

TASK NUMBER:	ETLT
TASK NAME:	Extract, Transform, and Load Task
PERFORMER:	Kapow RoboSuite 5.5
PURPOSE:	To extract data, transform or reformat it and load it into the OlympicsDB
ENABLING COND:	The creation of the OlympicsDB and any addition of data or updates to the OlympicsDB.
DESCRIPTION:	This tool (Kapow RoboSuite 5.5) extracts specific data from a web page, and load it into a predefined data relation or table.
FREQUENCY:	Once for the creation of the OlympicsDB and during any updates.
DURATION:	Varies
IMPORTANCE:	Critical
MAXIMUM DELAY:	N/A
INPUT:	A selected web page
OUTPUT:	Data into a relation in the OlympicsDB
DOCUMENT USE:	HTML documents
OPS PERFORMED:	Data extraction, data transformation, and data loading.
SUBTASKS:	Web pages Research
ERROR COND:	None

8

Another Task

3.2.2.8 Create Query Result Form Task

TASK NUMBER: CRFT
TASK NAME: Create Result Form
PERFORMER: Server side script
PURPOSE: Provide a formatted result from the OlympicsDB.
ENABLING COND: Database completing operations.
DESCRIPTION: Formats output of the extracted data from the OlympicsDB to a form that can be interpreted by a web browser.
FREQUENCY: Once per user query submission.
DURATION: Depends on the complexity of the query result.
IMPORTANCE: Critical
MAXIMUM DELAY: 5-10 seconds
INPUT: OlympicsDB data
OUTPUT: (SPR) Sport Event Result; (SEHR) Sport Event Historical Result; (CPHR) Country Participation History Result; (MCR) Medal Count Result; (MCoR) Medal Country Result; (TMAR) Top Medal Athletes Result; (TMCR) Top Medal Country Result; (PMIR) Poster/Medal Image Result; (YRBR) Year Record Broken Result, or (FAR) Flag/Anthem Result.
DOCUMENT USE: None
OPS PERFORMED: Transform data from the OlympicsDB output format to a web browser compatible format.
SUBTASKS: None
ERROR COND: If OlympicsDB_output=unknown, then produce error message and stop.

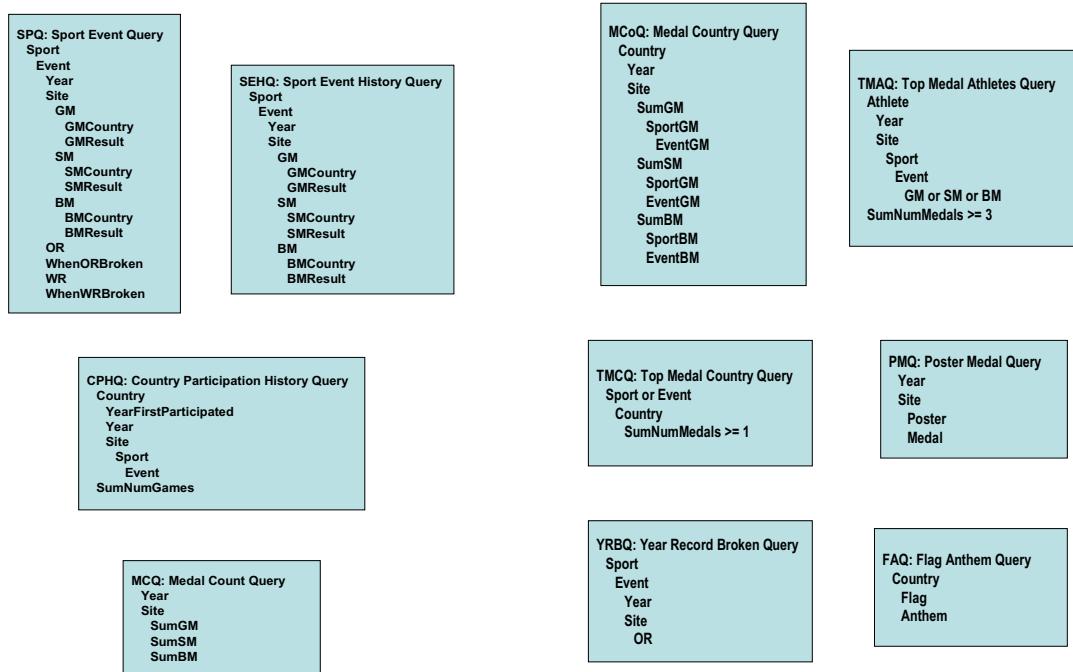
9

Rule of Thumb for Task Decomposition

- Many performers are required to carry out the task and each performer has different skills, or each can carry out a part independently.
- Different levels of authorization exist for carrying out different parts of the task.
- Different enabling conditions activate parts of the task.
- Different frequencies and durations apply to different parts of the task.
- Input documents are not used uniformly within the task.
- Different documents are used for different parts of the task.
- Many diversified operations are carried out within the task.
- Many subtasks are controlled by the task.

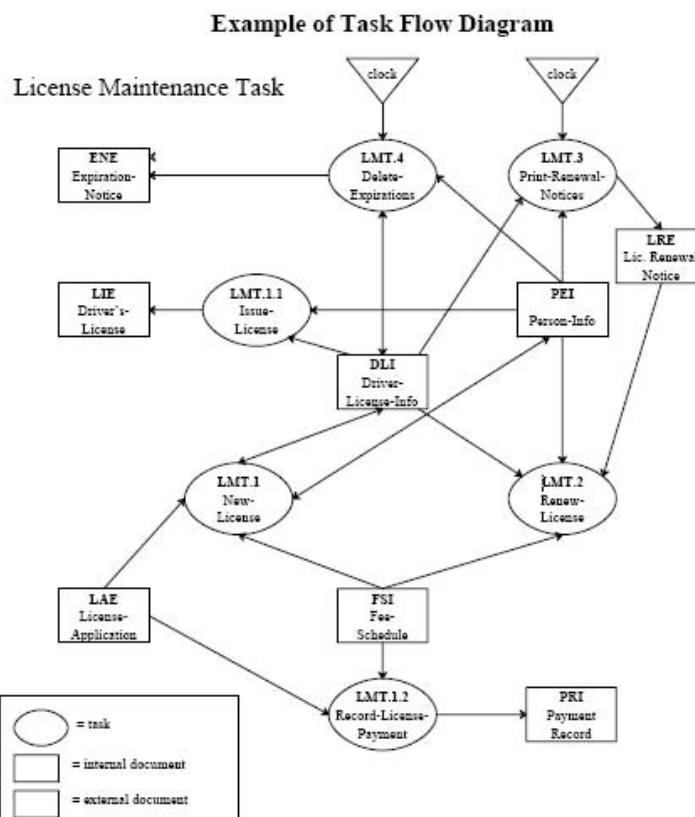
10

Examples of Document Forms



11

Task Flow Diagram



12

Task-Data Usage (Optional)

Example of Task-Data Usage Matrices

DLT Field	LMT.1	LMT.1.1	LMT.1.2	LMT.2	LMT.3	LMT.4
Driver-ID	I/O	I		I	I	I
License- Class	O					
Issue- Date	O					
Expiration- Date	O			I	I	I
License- Status	O			I		

DLT Field	LMT.1	LMT.1.1	LMT.1.2	LMT.2	LMT.3	LMT.4
SSN	I/O	I		I	I	I
Name	I/O	I			I	I
Height	I/O	I				
Weight	I/O	I				
Sex	I/O	I				
Birthdate	I/O	I				
Address	I/O	I			I	I
Fine-Test- Status	I/O	I		I		

13

Phase II.1 Conceptual Modeling

The purpose of this phase is to design a **conceptual schema** of the database. We will use the E-R data model.

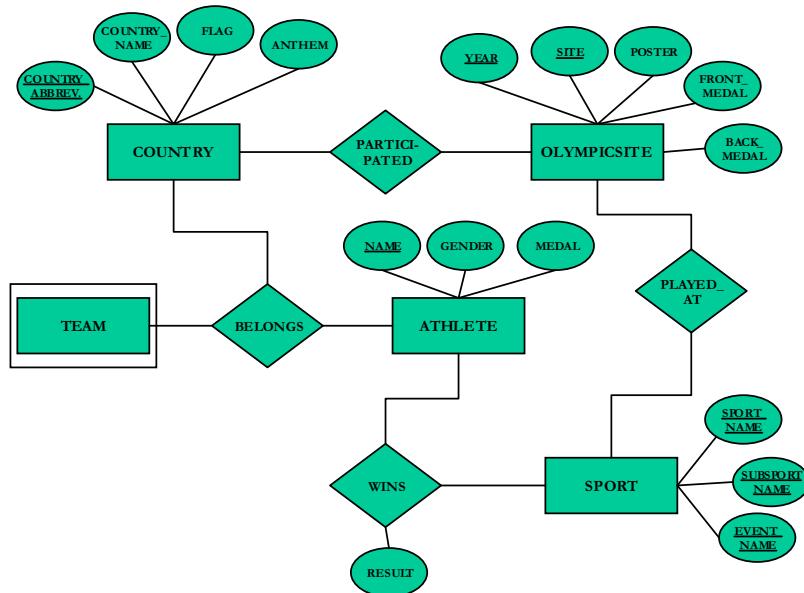
- **Input:**
 - The Document Forms
- **Output:**
 - A Conceptual Schema described in terms of the E-R data model
- **Function:**
 - To design the Conceptual Schema from the Document Forms
- **Guidelines:**
 - **Techniques** for conceptual schema design. E.g. **semantic data modeling and normalization!**

14

The Conceptual Model

Map Data Documents into E-R:

- Find Entities, their keys, and attributes
- Find Relationships, their keys, and attributes
- Discover FD's



15

Phase II.2 Logical Modeling

The purpose of this phase is to convert the **conceptual schema to a logical data model** of the database. We will map the E-R schema to a Relational schema.

■ Input:

- The E-R diagrams
- FDs discovered in II.1
- 1-1, 1-many, and many-many constraints of the relationships

■ Output:

- A Relational Schema (Logical Model) corresponding to the E-R model

■ Function:

- Map the E-R model to tables, their keys, and FDs
- Normalize the relations to obtain BCNF or at least 3NF relations.

■ Guidelines:

- Algorithm for mapping E-R to relations and normalization

16

Relational (Logical) Model

COUNTRY				
COUNTRY_ABBREV		COUNTRY_NAME	FLAG	ANTHEM
YEAR	SITE	POSTER	FRONT_MEDAL	BACK_MEDAL
SPORT				
SPORT_NAME		SUBSPORT	EVENT_NAME	
ATHLETE				
NAME		GENDER	MEDAL	
TEAM				
NAME		GENDER	MEDAL	
PLAYED_AT				
YEAR	SPORT_NAME	SUBSPORT_NAME	EVENT_NAME	
PARTICIPATED				
COUNTRY_ABBREV		COUNTRY_NAME	FLAG	ANTHEM
YEAR	SITE	POSTER	FRONT_MEDAL	BACK_MEDAL
BELONGS				
NAME		GENDER	MEDAL	
WINS				
NAME		GENDER	MEDAL	
SPORT_NAME	SUBSPORT	EVENT_NAME		

Functional Dependencies

For Country entity:

*Country_Abbreviation → Country_Name
 Country_Abbreviation → Flag
 Country_Abbreviation → Anther
 Country_Abbreviation → First_Year_Participated*

For OlympicSite entity:

*Year → Site
 Year → Poster
 Year → Medal*

17

Phase II.3 Task Emulation

The purpose of this phase is to obtain the design and specification of the software that performs the tasks **before** any database implementation starts. In other words, before creating a schema in the DBMS, the application programming is fully specified. This gives the opportunity to correct the logical schema when it is incomplete, superfluous, or even dead wrong. Doing the design of both the database schema and the applications using the schema simultaneously complements these two orthogonal specifications and catches most of the errors before the implementation.

■ Input:

- The Logical Schema from the previous phase
- The Task Forms

■ Output:

- The set of design specifications of the pieces of software that performs the tasks described in the task forms. The design specifications can be given in terms of abstract programs with embedded sequences of DML statements,

18

Phase II.3 Task Emulation

(cont)

■ Function:

- Use the Task Forms describing the tasks. Formulate for each task an abstract program including embedded sequences of DML statements that perform the task using the conceptual schema. (During this phase small corrections of the conceptual schema may be needed to support the tasks: validation).

■ Guidelines:

- **Techniques:** those that apply to the use of the particular DML.

19

Task Emulation

Extract, Transform, and Load

Start RoboSuite 5.5

Configure RoboSuite 5.5

for each website bookmarked

 for each webpage on website [query results]

 RoboSuite.url = webpage.url

 set values to look for

 extract information to a predefined table.

Web Pages Research

{Google query to find Summer Olympic Games sites}

For each website found in Google

 if website has relevant data and if website has complete
 data to be used by the OlympicsDB

 Bookmark

 else

 skip

20

Task Emulation

```
Generate SQL
If query == Sport_Event_Query
    SELECT year, site, sport_name, subsport_name, event_name,
        subevent_name, medal
    FROM Sports, OlympicSites, Medal, Wins, Played_At
    WHERE year=year_chosen and site=site_chosen and
        sport_name=sport_name_chosen and
        subsport_name=subsport_name_chosen and
        event_name=event_name_chosen and
        subevent_name=subevent_name_chosen and
        medal=medal_chosen
Else if query == Sport_Event_Historical_Query
    SELECT year, site, sport_name, subsport_name, event_name
        subevent_name, medal
    FROM Sports, OlympicSites, Medal
Else if query == Country_Participation_History_Query
    SELECT C.year, site, country_abbreviation, C.country_name,
        year_first_participated, count(country_name)
    FROM Country C, OlympicSite O, Participated P
    GROUP BY P.year
Else if query == Medal_Count_Query
    SELECT year, site, country_name, count(medal)
    FROM OlympicSite, Country, Medal, Participated,
        Wins, Belongs
    WHERE year=year_chosen and site=site_chosen and
        country_name=country_name_chosen
    GROUP BY medal
Else if query == Medal_Country_History_Query
    SELECT year, site, country_name, medal
    FROM OlympicSite, Country, Medal, Participated, Wins
    Belongs
```

```
Generate SQL (cont...)
Else if query == Top_Medal_Athletes_Query
    SELECT year, site, first_name, last_name, medal
    FROM OlympicSite, Athlete, Belongs, Participated, Medal,
        Sport, Wins, Played_At
    HAVING count (medal) > 3
Else if query == Top_Medal_Country_Query
    SELECT year, site, country_name, event_name, count(medal)
    FROM OlympicSite, Country, Sport, Medal, Win, Participated
        Played_At
Else if query == Poster/Medal_Image_Query
    SELECT year, site, poster, front_medal, back_medal
    FROM OlympicSite
    WHERE year=year_chosen and site=site_chosen
Else if query == Year_Record_Broken_Query
    SELECT
    FROM
    WHERE
Else if query == Flag/Anthem_Query
    SELECT year, site, country_name, flag, anthem
    FROM OlympicSite, Country
    WHERE year=year_chosen and site=site_chosen and
        country_name=country_name_chosen
```

21

Phase III.1 Implementation

The purpose of this phase is to translate the conceptual schema and the task design specifications into actual schema definitions and application program modules.

■ Input:

- The relational (logical) schema
- the task specifications

■ Output:

- The DDL statements for the DBMS
- The tasks programmed in terms of the host-language with embedded SQL statements.

■ Function:

- To translate schemata into a definition of the schemata using the DDLs.
To translate the task designs into host-language modules.

■ Guidelines:

- Technique:** Not really. Scratch your heads!

22

Example

```
QUERYFA.JSP (Flags/Anthems Query Page)
<jsp:useBean id="user" class="SQLUtilities.UserData" scope="session"/>
<jsp:setProperty name="user" property="*"/>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="SQLUtilities.Utilities" %>
<%@ page errorPage="myError.jsp?from=QueryFA.jsp" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>OlympiChronicles Query : FLAGS and ANTHEMS</title>

[automatically generated code...]

<p>Please select the options for your query:</p>
<form action="QueryResultFA.jsp" method="post" name="form1" >

<select name="select" onChange="MM_showHideLayers('Layer16','show')">
<option value="0" selected>Select Country</option>
<%= Utilities.getCountryList() %>
</select>
<p>
<input type="submit" name="Submit" value="Submit">
<input type="reset" value="Reset">
</p>
</div>
</form>
</div>
</div>
<p> <br>
</p>
</div>
</body>
</html>
```

23

Phase III.2 Bulk Loading & Testing

The purpose of this phase is to load the real stuff and fine tune its performance.

■ Input:

- The schema definitions and the application programs from the previous step
- A set of test data.

■ Output:

- The database system.

■ Function:

- Almost always this is very painful step which can take several weeks or even months. The biggest problem is data errors that need to be cleaned before entered. Bulk loading implies high volume of data (unlike your CMSC 424 project).

■ Guidelines:

- **Technique:** patience!
- **Tool:** bulk loaders and scripting languages.

24

DON'T FORGET

- The secret behind successful Database Design is careful analysis, specification, and design. These are done in the phases I.1-II.3 of the methodology. Having done a careful analysis on these, it would give enough chances not to fail!
- There are always bugs in large databases. Careful testing eliminates only the most obvious. Testing requires a systematic methodology different than the one used by Microsoft!
- Large databases are used for many years. Maintaining a database throughout its life-time typically takes several times more than development. It is impossible to maintain a database with an undocumented design. The documents produced by this methodology is the design specification and will be the heart of the documentation if properly maintained. Without the methodology, there is no common language to exchange design specifications.



4. Development of the SDBD Tool via Customisation Utilising the ADOxx 1.0 Platform

For the reference implementation of the SDBD Tool the metamodelling platform ADOxx 1.0 has been chosen. The main advantage of ADOxx is that methods can be defined with very low effort. Important components like model browser for mapping the models, versioning mechanisms for versioning of mapped models etc. are available out-of-the-box.

In the following section the main steps for implementing a method using the Administration-Workspace of ADOxx 1.0 is presented. A concrete ADOxx configuration is called Application Library and ADOxx can run multiple Application Libraries within the same database. Every ADOxx application library is characterised both by the definition of the modelling method and by the definition of the evaluation mechanisms (techniques).

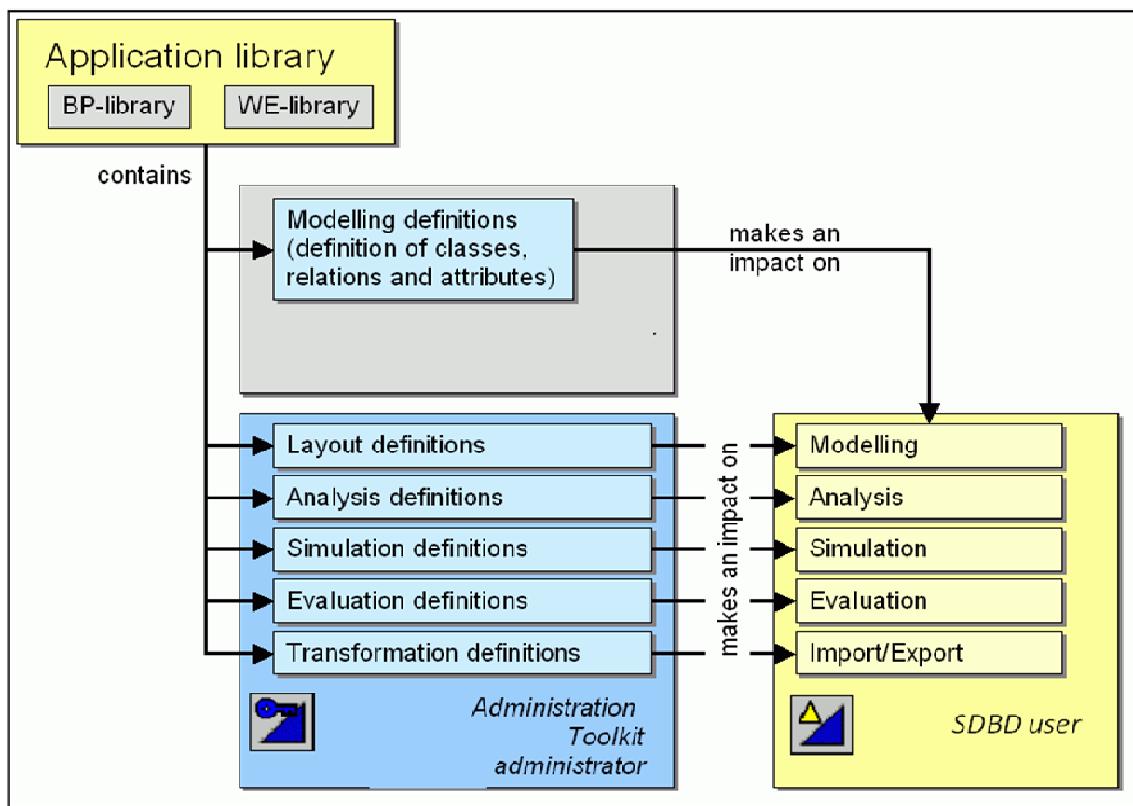


Figure 2 ADOxx Application Library

An application library consists of modelling classes and relations. These classes and relations can be combined in various model types.

4.1. Defining the Metamodel

The classes and relations are defined in the application library. The definition includes specifying the attributes available (modelling definitions) and the graphical representation

(layout definition). Figure 3 shows the ‘Class Hierarchy Management Component’ – the component for defining modelling classes and attributes.

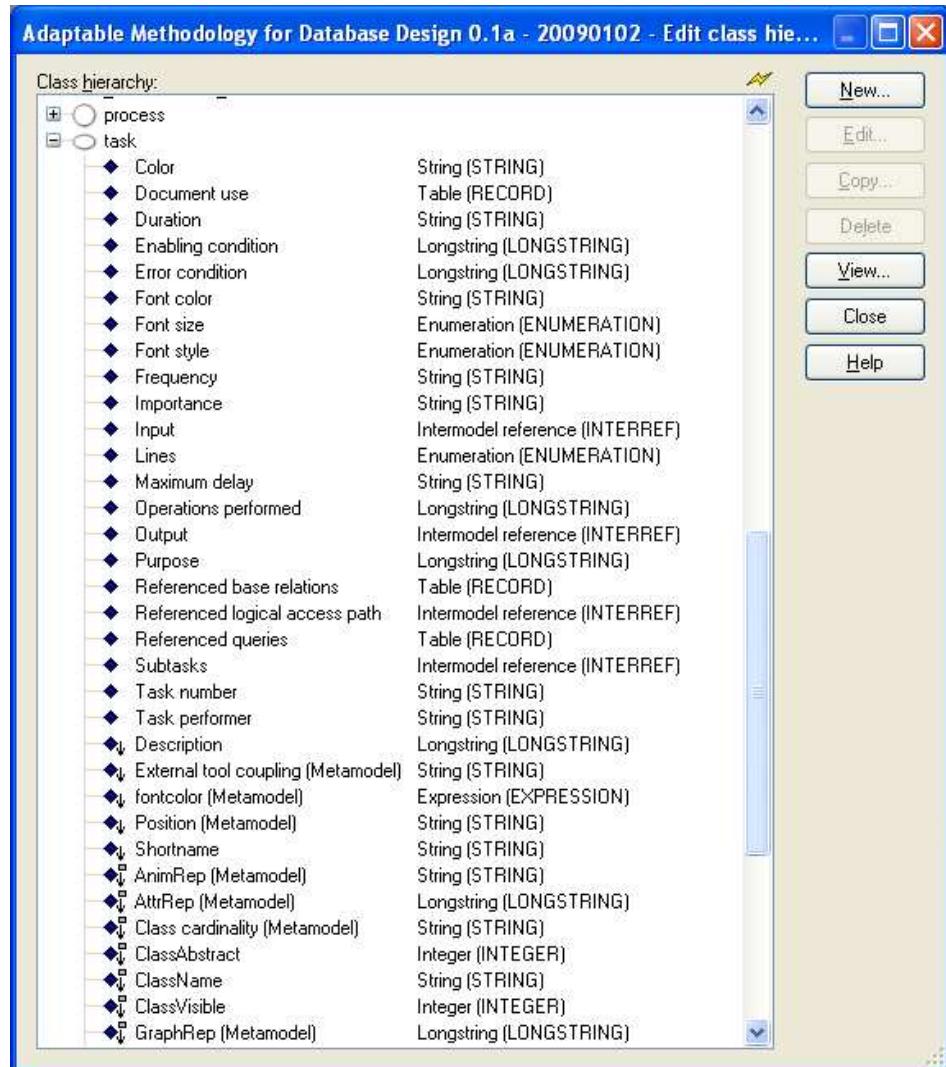


Figure 3 Class Hierarchy Management Component

For each modeling class attributes are defined. For example, for the modeling class “Task” attributes like “Name”, “Description” and “Frequency” are defined. Therefore a variety of attribute types is available. Besides common types like string, date and time complex attribute types like Expression – for defining formulas – are available.

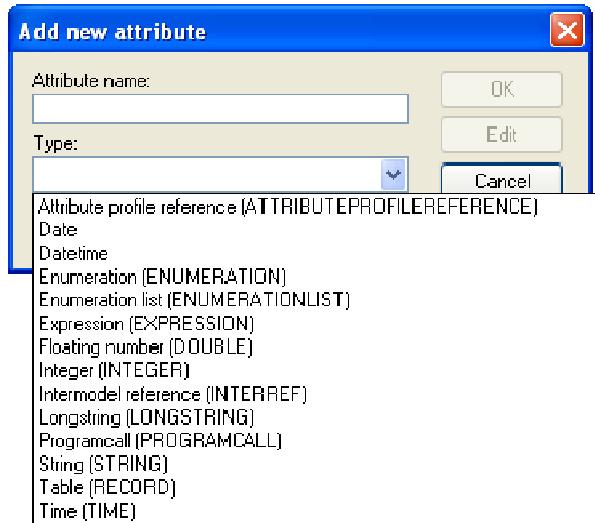


Figure 4 Adding Attributes – Available Attribute Types

4.2. Graphical Representation of Modelling Classes and Relations

The class attribute "GraphRep" controls the graphical representation of a class or relation and the arrangement of any visualised attribute values when working in the Modelling Environment of ADOxx (e.g. in the SDBD Tool).

The language for the graphical representation of objects and relations is based on the GRAPHREP syntax.

Figure 5 depicts the ADOxx Graphrep Editor. In the example at hand, the GraphRep of the modelling class "Task" is specified.

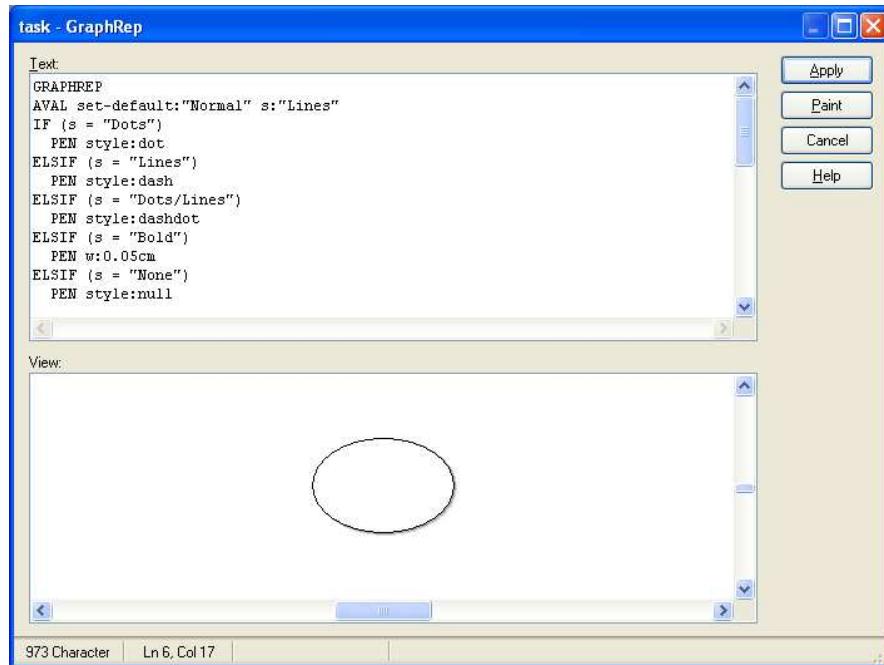
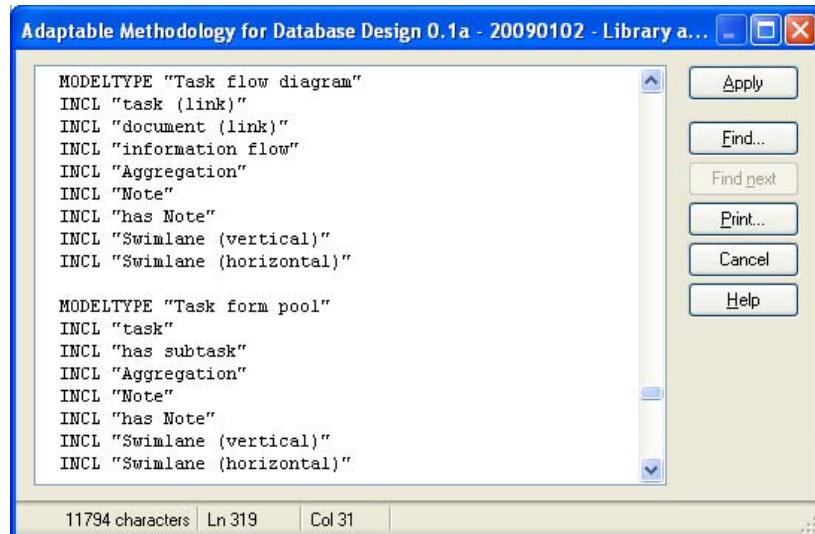


Figure 5 GraphRep – Definition of the Representation of Modelling Classes

4.3. Define Modeltypes

A model type specifies a subset of all instantiable classes and relations. Every model is of a certain model type. Figure 6 depicts the dialog and the language for defining modeltypes.



```

Adaptable Methodology for Database Design 0.1a - 20090102 - Library a...

MODELTYPE "Task flow diagram"
INCL "task (link)"
INCL "document (link)"
INCL "information flow"
INCL "Aggregation"
INCL "Note"
INCL "has Note"
INCL "Swimlane (vertical)"
INCL "Swimlane (horizontal)"

MODELTYPE "Task form pool"
INCL "task"
INCL "has subtask"
INCL "Aggregation"
INCL "Note"
INCL "has Note"
INCL "Swimlane (vertical)"
INCL "Swimlane (horizontal)"

11794 characters | Ln 319 | Col 31

```

Figure 6 Definition of Modeltypes

For the SDBD method the modeltypes, depicted in figure 7 have been implemented.

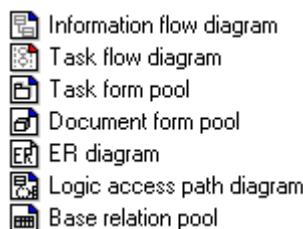


Figure 7 Modeltypes of the SDBD Tool

Each of the modeltypes has been assigned to a method filter which is implemented again utilising the GraphRep language.

Figure 8 depicts the method filter of the 'New Model Dialogue'.

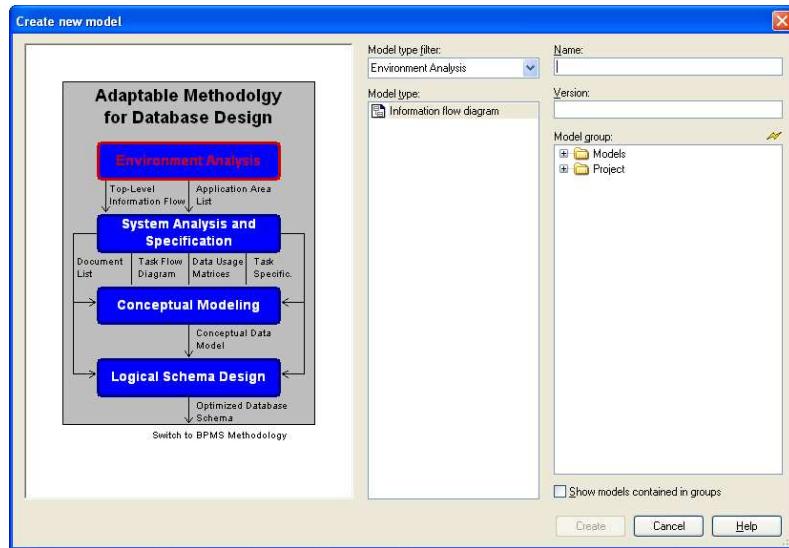


Figure 8 New Model Dialogue

4.4. Additional Functionality

The 'AdoScript' script language allows to add additional functionality to the method. Apart from that, AdoScripts allow to call any external programs and functions in DLLs.

Via AdoScript extended functionality like the functionality for computing the probabilities (see Chapter 5.5.3, Probabilities Calculator), the automatic generation of LAP schema (see 5.5.3, Graph Generator), and the functionality for merging single query graphs into a complete LAP (see chapter 5.5.3, Model Merger) have been realised.

The syntax of AdoScript is based on the following grammar:

```

StatementSeq:          {Statement} .
Statement:           Execute | Send | CC | System | Start | Call |
                      Set | SetL | SetG | Leo | IfStatement | WhileStatement |
                      ForStatement | BreakStatement | ExitStatement | FunctionDefinition
                      | ProcedureDefinition | ProcedureCall.
Execute:             ExecuteFile | ExecuteEx.
ExecuteFile:          EXECUTE file: scriptText [scope: ScopeSpec].
ExecuteEx:           EXECUTE scriptText [scope: ScopeSpec].
ScopeSpec:          separate | same | child.
Send:               SEND msgText to: msgPortName [answer: varName].
CC:                CC msgPortName [debug] [raw] anyLeoElement.
System:             SYSTEM strExpr [hide] [result: varName].
Start:              START strExpr [cmdshow: CmdShow].
CmdShow:            showmaximized | showminimized | showminnoactive | shownormal.
Call:               CALL dll: strExpr function: strExpr {InputParam}
                      [result: varName] [freemem: strValue].
InputParam:         varName: anyExpr.
Set:                SET {VarAssignment}.
SetL:               SETL {VarAssignment}.

```

```

SetG: SETG {VarAssignment}.

VarAssignment: varName: anyExpr.

Leo: LEO {LeoCmd}.

LeoCmd: parse: stringExpr | get-elem-count: varName | set-cur-elem-index:
           intExpr | get-keyword: varName |
           is-contained: varName [: strExpr] |
           get-str-value: varName [: strExpr] |
           get-int-value: varName [: strExpr] |
           get-real-value: varName [: strExpr] |
           get-tmm-value: varName [: strExpr] |
           get-time-value: varName [: strExpr]] |
           get-modifier: varName: strExpr.

IfStatement: IF booleanExpr {StatementSequence}
               {ELSIF booleanExpr {StatementSequence}}}
               [ELSE {StatementSequence}].

WhileStatement: WHILE booleanExpr {StatementSequence}.

ForStatement: ForNumStatement | ForTokenStatement.

ForNumStatement: FOR varName from: numExpr to: numExpr
                   [by: numExpr] {StatementSequence}.

ForTokenStatement: FOR varName in: strExpr [sep: strExpr]
                   {StatementSequence}.

BreakStatement: BREAK.

NextStatement: NEXT.

ExitStatement: EXIT.

FunctionDefinition: FUNCTION functionName [: global] {FormalFuncParameter}
                      return: expression.

FormalFuncParameter: [reference] paramName: TypeName.

ProcedureDefinition: PROCEDURE [global] ProcedureName
                        [MainParameter] {FormalProcParameter}
                        {StatementSequence}.

MainParameter: TypeName: paramName.

FormalProcParameter: [reference] paramName: TypeNameOrReference.

TypeNameOrReference: TypeName | reference.

TypeName: string | integer | real | measure | time | array | expression |
undefined.

ProcedureCall: anyLeoElement.

ProcedureName: keyword.

```

Hint:

For more information on the script language ‘AdoScript’ refer to the ADOxx help available within the SDBD Tool.

5. User Manual for the Semantic Database Design Tool (SDBD Tool)

This chapter discusses the prototypical implementation of the SDBD method - the SDBD Tool. The user manual is structured according to the proposed phases of the SDBD. Figure 9 depicts the phases and their major deliverables as discussed in chapter 2 and 3.

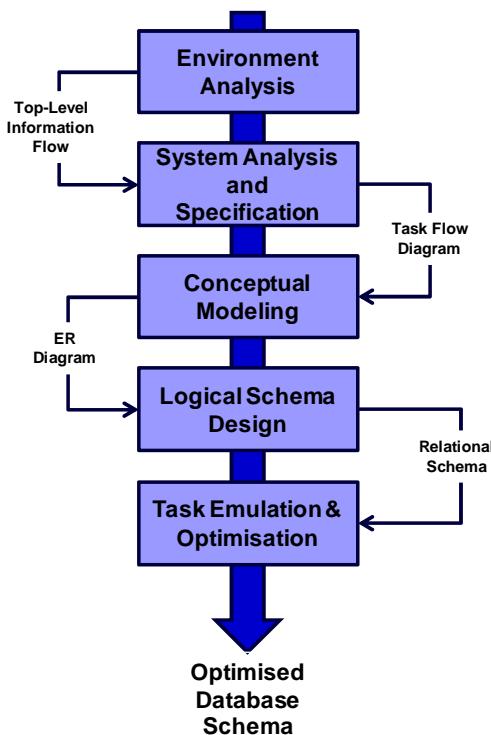


Figure 9 Phases of the SDBD Method

Within the user manual each of the phases is further detailed to the following key concepts:

- A '**semi-formal**' notation for specifying the concepts and ideas,
- the main **steps** to be performed within each of the proposed phases, and
- the **techniques** –the tool functionality provided to support the various steps.

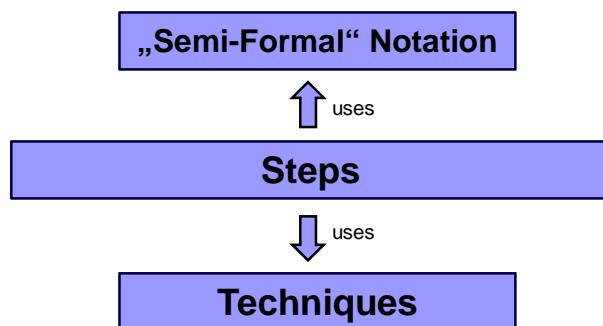


Figure 10 Key Concepts of Each Phase

5.1. Environment and Requirements Analysis

The purpose of this phase is to narrow the design problem by investigating the informational demands and corresponding activities within the enterprise on the one hand, and to locate boundaries of the future system on the other hand.

Inputs consider current status of the enterprise and possible inefficiencies as well as desired future outcomes. The output of this phase is an ‘Information Flow Diagram’, which contains major documents, functions and the system boundaries of the design problem. Figure 11 depicts the key concepts of this phase.

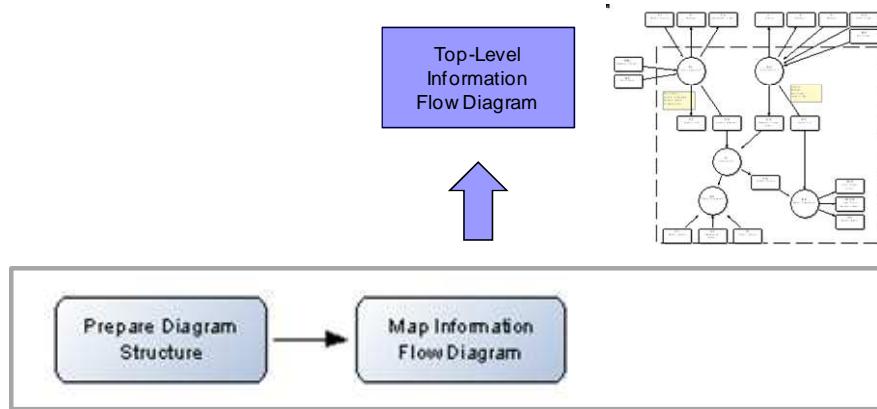


Figure 11 Key Concepts within ‘Environment and Requirements Analysis’

5.1.1. Steps

Step 1: Prepare Diagram Structure

In the first step the model group structure in the SDBD Tool is to be prepared (see figure 12).

Next an ‘Information Flow Diagram’ (DOC) is to be generated and stored within the ‘Environment and Requirements Analysis Folder’.

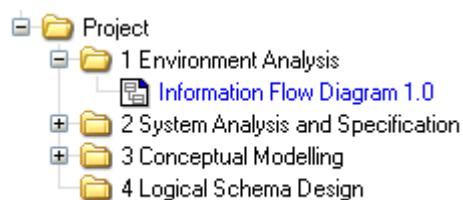


Figure 12 Recommended Model Group Structure – Information Flow Diagrams

Step 2: Map Information Flow Diagram

To acquire the Information Flow Diagram workshops with brainstorming techniques can be carried out. By applying the semi-formal notation discussed in the next chapter the Top Level Information Flow is mapped within the previously prepared ‘Information Flow Diagram’.

5.1.2. ‘Semi-Formal’ Notation

In the following table the required modelling classes and relation classes are briefly discussed.

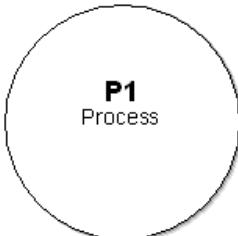
	Process	Processes comprise the main activities of the system. They represent high level activities which are detailed in the phase ‘System Analysis and Specification’.
	Document	Documents are either input for the execution of a process or are created as output when executing a process. A document is merely a general information-recording medium for information transfer. All messages, forms, files, inquiries, and reports will be referred to as documents.
	Clock	Clocks are used for representing time-dependent process triggers.
	System Boundary	The boundary of the system shows the interactions of the environment and the enterprise. Furthermore the scope of the system is indicated.
	Information Flow	Via Information Flow input and output documents are assigned to processes.
	Control Flow	The Control Flow assigns documents (simple control messages) that trigger processes in the organisation.

Table 1 Modelling Classes of the Modeltype ‘Information Flow Diagram’

5.1.3. Techniques

<No techniques implemented to support this phase.>

Usually, the function associated with each operational process is too complex to comprehend and model using a flat, single-level presentation. Therefore, the processes are usually divided into smaller units called tasks. Concepts to be applied are discussed in the following chapter 5.2.

5.2. System Analysis and Specification

The objective of the 'System Analysis and Specification' is to identify the applications that will use the database. Using input from the previous phase, a good understanding of the target system is obtained.

Key deliverable is the structured derivation of task flows detailing the processes which have been defined within the system boundary of the 'Information Flow Diagram'. Figure 13 depicts the key concepts of this phase.

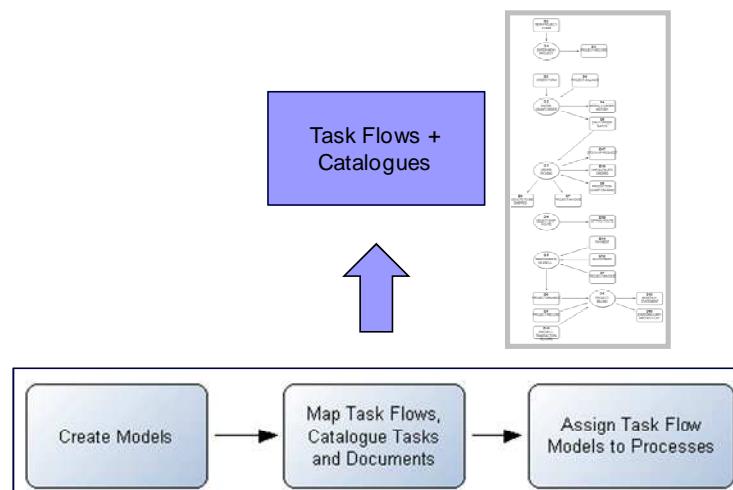


Figure 13 Key Concepts within 'System Analysis and Specification'

5.2.1. Steps

Step 1: Create Models

A model of modeltype 'Task Form Pool' (

 to catalogue all identified tasks is created. Furthermore a 'Document Form Pool' (table border="1") is to be created. This model will store all identified input and output documents, including a detailed description on the documents/data elements. Cataloguing 'Tasks' and 'Documents' within pool models allows reusing of these objects within task flow diagrams.

For each process within the Information Flow Diagram a 'Task Flow Diagram' (table border="1") is created. Figure 14 shows exemplarily the model catalogue, and the assigned models

- 'Task Flow Models',
- 'Task Form Pool' and
- 'Document Form Pool'.

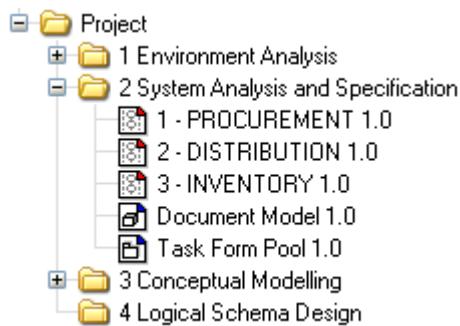


Figure 14 Recommended Model Group Structure within 'System Analysis and Specification'

Step 2: Map Task Flows, Catalogue Tasks and Documents

Within this step the task flows are defined. Again methods like brainstorming sessions and modelling workshops with subject matter experts are used to acquire the relevant information.

All tasks are stored within the task form pool to enable reusability of tasks.

Within the 'Task Flow Models' objects of the modelling class 'Task (Link)', representing pointers to the tasks stored within the 'Task Form Pool Model', are mapped. For documents the same concept applies: 'Document (Links)' are used within the modeltype 'Task Flow Model' and 'Documents' are specified within the 'Document Form Pool'. Each 'Task (Link)' and each 'Document (Link)' needs to be assigned to its corresponding element within the pool models.

Figure 15 illustrates this circumstance. The document 'D1 Project-Record' is used in two different task flow diagrams.

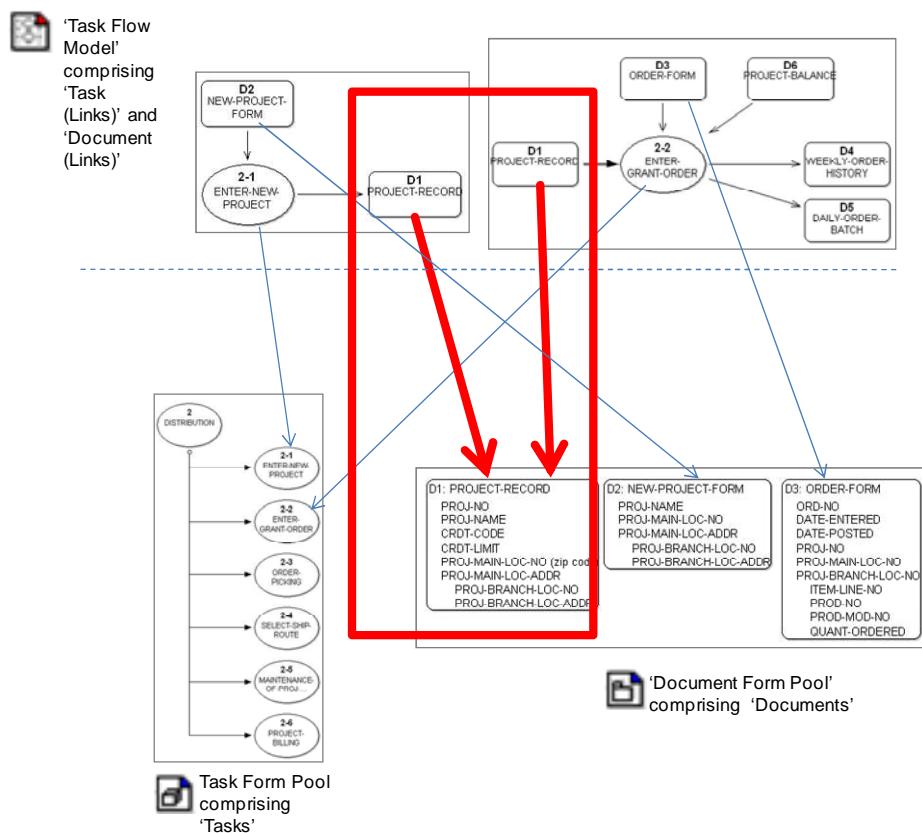


Figure 15 The Concept of ‘Pool Models’ – Reuse of Tasks and Documents

For each task the following information is specified:

- the person in charge (task performer),
- documents used and/or generated (1/0 documents),
- procedure for carrying out the task,
- frequency of the task,
- conditions that specify the prerequisites for the task's,
- activation and the effects after its termination,
- events triggered during the task's performance, and
- error (anomaly) conditions of the task.

ENTER-GRANT-ORDER (task)

Task number:	2-2	Description				
Name:	ENTER-GRANT-ORDER	Classification				
Task performer:	Tom Keener, Distribution Coordinator	Input/Output				
Authorization:	Scott Farley	Operation				
Purpose:	Enters orders in the database and grants orders.	Representation				
Enabling condition:	Arrival of ORDER:FORM (03).					
Description:	ORDER-FORMS (D3) are distributed and recorded. Balance of ordering project is checked: to determine whether order should be granted. Order history is generated every week.					
Referenced queries:	<table border="1"> <thead> <tr> <th>Query</th> <th>Quantity</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>		Query	Quantity		
Query	Quantity					

Close Reset 1/2

Figure 16 Notebook of Tasks (within Task Pool Model)

The required documents (reports, files, data elements, inquiries to files, forms, etc.) are derived from the input/output specifications of the tasks. The list of required documents includes

documents used within the enterprise or documents discovered by the requirements analysis. Messages received by the task performer during task execution are also considered documents.

Step 3: Assign Task Flow Models to Processes

In a last step the “Task Flow Diagrams” are assigned to their corresponding processes in the ‘Information Flow Model’ defined in phase ‘Environment and Requirements Analysis’. Figure 17 illustrates this coherence.

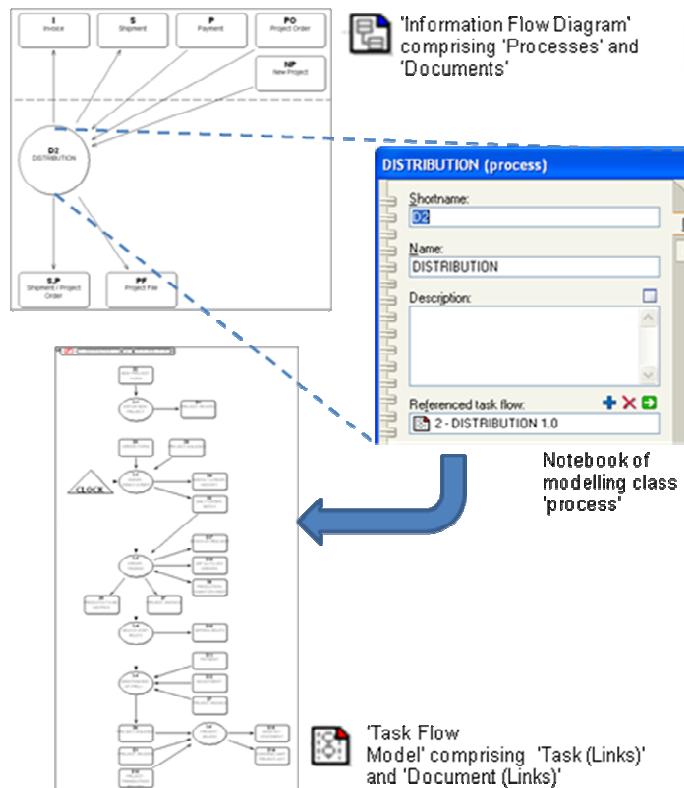


Figure 17 Assigning ‘Task Flow Models’ to ‘Processes’

5.2.2. ‘Semi-Formal’ Notation

In the following table the relevant modelling classes and relation classes are briefly discussed.

T1 TASK	Task	Objects of modelling class ‘Task’ represent atomic activities within a process (task flow).
D1: DOCUMENT ATTRIBUTE 1 ATTRIBUTE 2 ...	Document	Objects of class ‘Document’ provide information about what tasks require to be performed and/or what they produce. For each Document its content (attributes) needs to be defined.

	Task (Link)	'Task (Links)' are used for modelling task flow diagrams, a sequence of tasks forming a process. Each 'Task (Link)' is linked to its corresponding Task within the 'Task Form Pool'.
	Document (Link)	'Document (Links)' are used for depicting 'Task Flow Diagrams'. The original object, specifying the document, is stored in the 'Document Form Pool'.
	Information Flow	The Information Flow assigns input and output documents to tasks.
	Control Flow	The Control Flow assigns documents (simple control messages) that trigger tasks within the process (task flow).

Figure 18 Modelling Classes of the Modeltype „Task Flow Model“

5.2.3. Technique

<No techniques implemented to support this phase.>

5.3. Conceptual Modelling

The goal of this phase is to translate the knowledge about each process and task, collected in the previous phases, into a formal representation called a conceptual schema. In this phase, designers focus on the characteristics of the 'entities' in the application environment and the relationships among them. They should not be concerned with the way they are represented in the computer. Figure 19 depicts the key concepts of this phase.

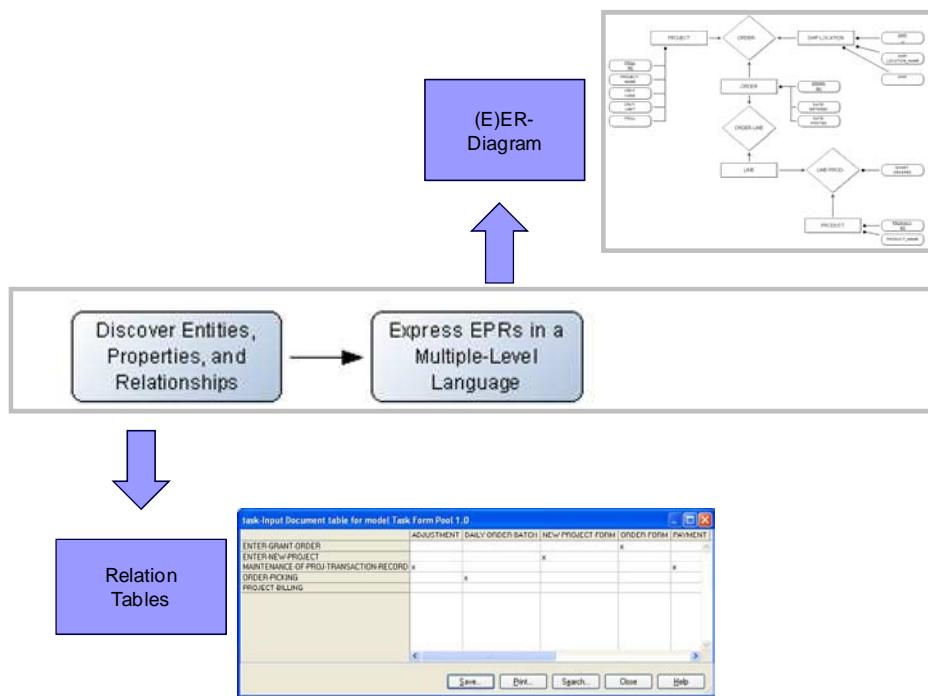


Figure 19 Key Concepts within 'Conceptual Modelling'

5.3.1. Steps

Step 1: Discover Entities, Properties, and Relationships

Conceptual data modeling consists of identifying the entities, the properties, and the relationships (called EPRs hereafter) in the application. In general, there are no algorithmic methods for identifying EPRs, but in chapter 2 some rules of thumb are specified to support this step. As stated in chapter 2, major input for discovering EPRs are the document specifications. Figure 20 depicts relation tables (task-document) as proposed.

	ADJUSTMENT	DAILY-ORDER-BATCH	NEW-PROJECT-FORM	ORDER-FORM	PAYMENT	PRODUCTION-
ENTER-GRANT-ORDER				x		
ENTER-NEW-PROJECT			x			
MAINTENANCE-OF-PROJ-TRANSACTION-RECORD	x				x	
ORDER-PICKING		x				x
PROJECT-BILLING						

Figure 20 Tasks and Assigned Input Documents

Step 2: Express EPRs in a Multiple-Level Language

After analysing the existing models and relation tables an “E-R Model” (ER) is created and the relevant the EPRs are mapped. Figure 21 depicts the E-R Diagram assigned to the model group ‘Conceptual Modelling’.

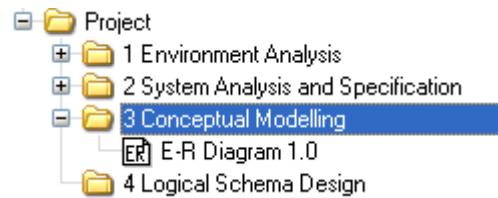


Figure 21 E-R Diagram within the Model Catalogue

After initially creating the E-R Diagram the EPRs are mapped. Figure 22 depicts an example of an E-R diagram. On the top documents incl. their properties (and relations) are depicted. On the bottom of the figure a section of the resulting E-R Model is depicted. For some of the attributes the mapping from the initial document to the E-R schema is presented via arrows.

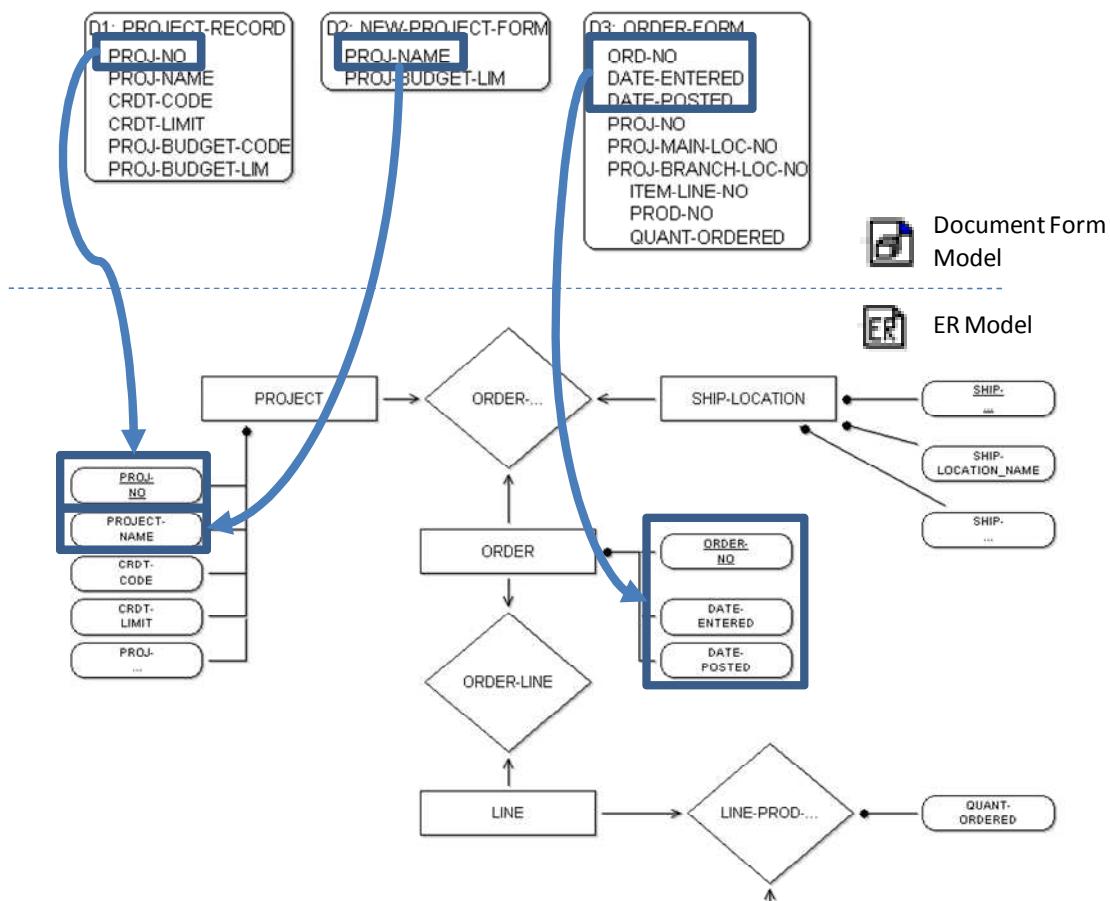


Figure 22 Derivation of the E-R Model

5.3.2. ‘Semi-Formal’ Notation

In the following table the relevant modelling classes and relation classes are briefly discussed.

ENTITY	Entity	An entity may be defined as a thing which is recognised as being capable of an independent existence and which can be
---------------	--------	---

		<p>uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.</p> <p>An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order.</p>
	Relationship	<p>A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: an owns relationship between a company and a computer, a supervises relationship between an employee and a department, a performs relationship between an artist and a song, a proved relationship between a mathematician and a theorem.</p>
	Attribute	<p>Entities and relationships can both have attributes. Examples: an employee entity might have a Social Security Number (SSN) attribute; the proved relationship may have a date attribute. Attributes are represented as ellipses connected to their owning entity sets by a line.</p> <p>Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.</p>
	Associates	<p>Lines are drawn between entity sets and the relationship sets they are involved in.</p>
	has attribute	<p>The connector 'has attribute' assigns attributes to 'Entities' and 'Relations'.</p>

Figure 23 Modelling Classes of the Modeltype 'E-R Model'

5.3.3. Techniques

In this phase the functionality ‘Create Relation Table’ is used as input for deriving the EPRs. Relation tables can be automatically derived from the Task Pool Model. Figure 24 shows these dependencies.

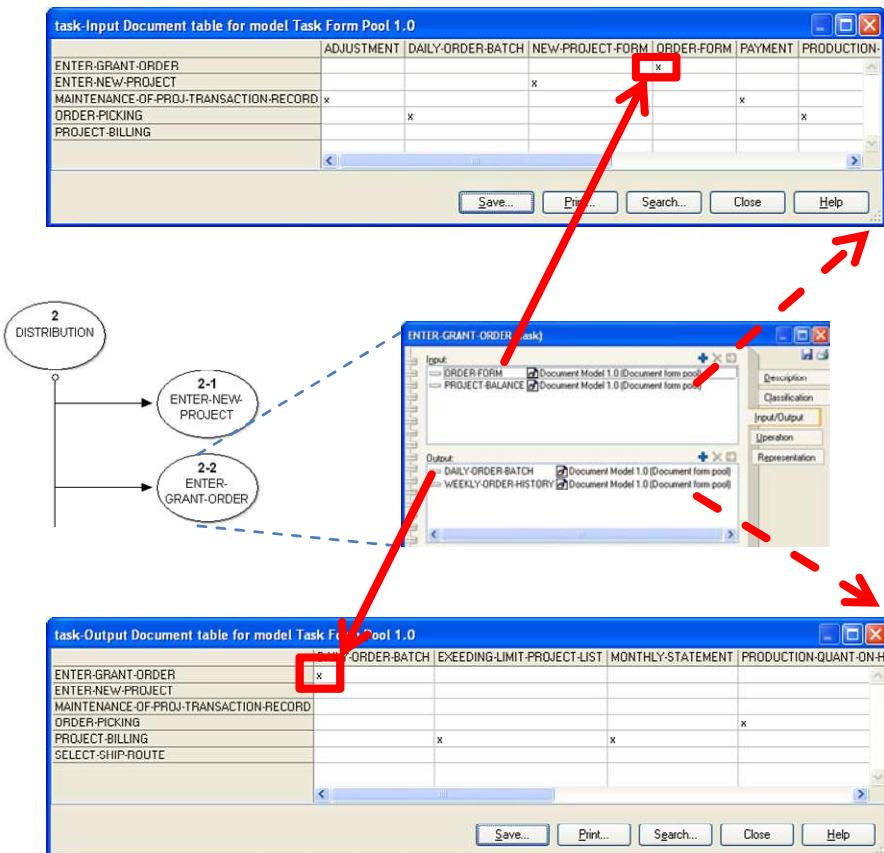


Figure 24 Technique ‘Relation Table’

5.4. Logical Schema Design

The conversion of the conceptual schema into a logical data model is the purpose of this phase. Result of the Logical Schema Design is a Relational Schema corresponding to the Entity-Relationship diagram delivered by the previous phase. Figure 25 depicts the key concepts of this phase.

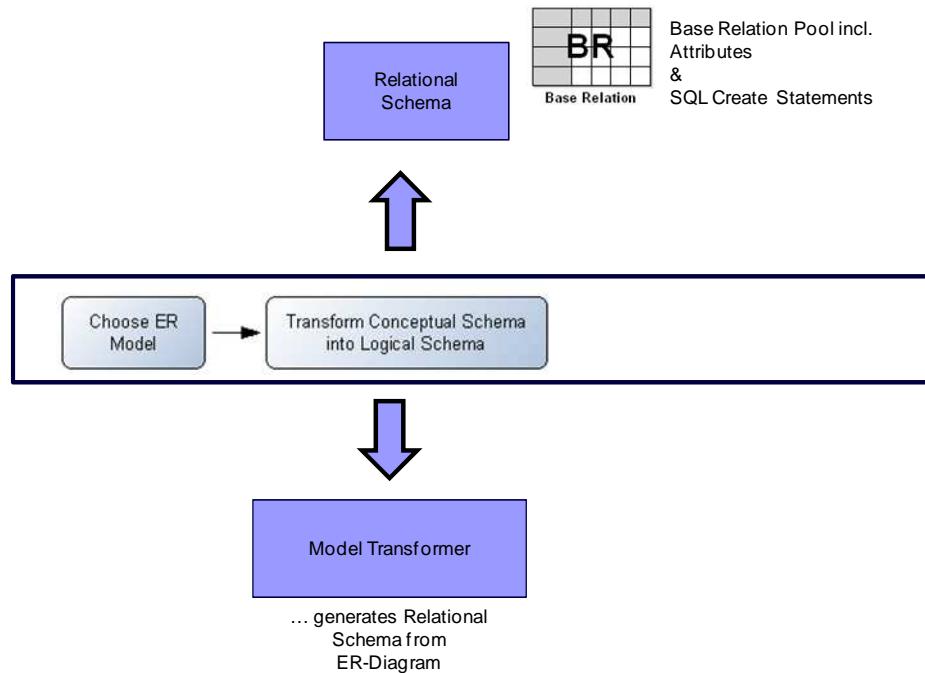


Figure 25 Key Concepts within 'Logical Schema Design'

5.4.1. Steps

Step 1: Choose E-R Diagram

Input for this phase consists of the conceptual schema (E-R diagram) obtained in the previous phase, task specifications, and document specifications.

Step 2: Transform Conceptual Schema into Logical Schema

Transformation Rules for deriving the Logical Schema (Relational Schema) are discussed in chapter 2. The *SDBD Tool* generates the logical (relational) schema automatically in accordance with the proposed rules.

Output of this phase is

- a 'Base Relation Pool Model' comprising the tables and their associated columns, and
- the relevant SQL create statements for creating the database instance.

Both deliverables are discussed in more detail in chapter 5.4.3.

Note:

The automatically generated logical schema gives just a first indication on the final schema. It needs to be reworked manually.

5.4.2. 'Semi-Formal' Notation

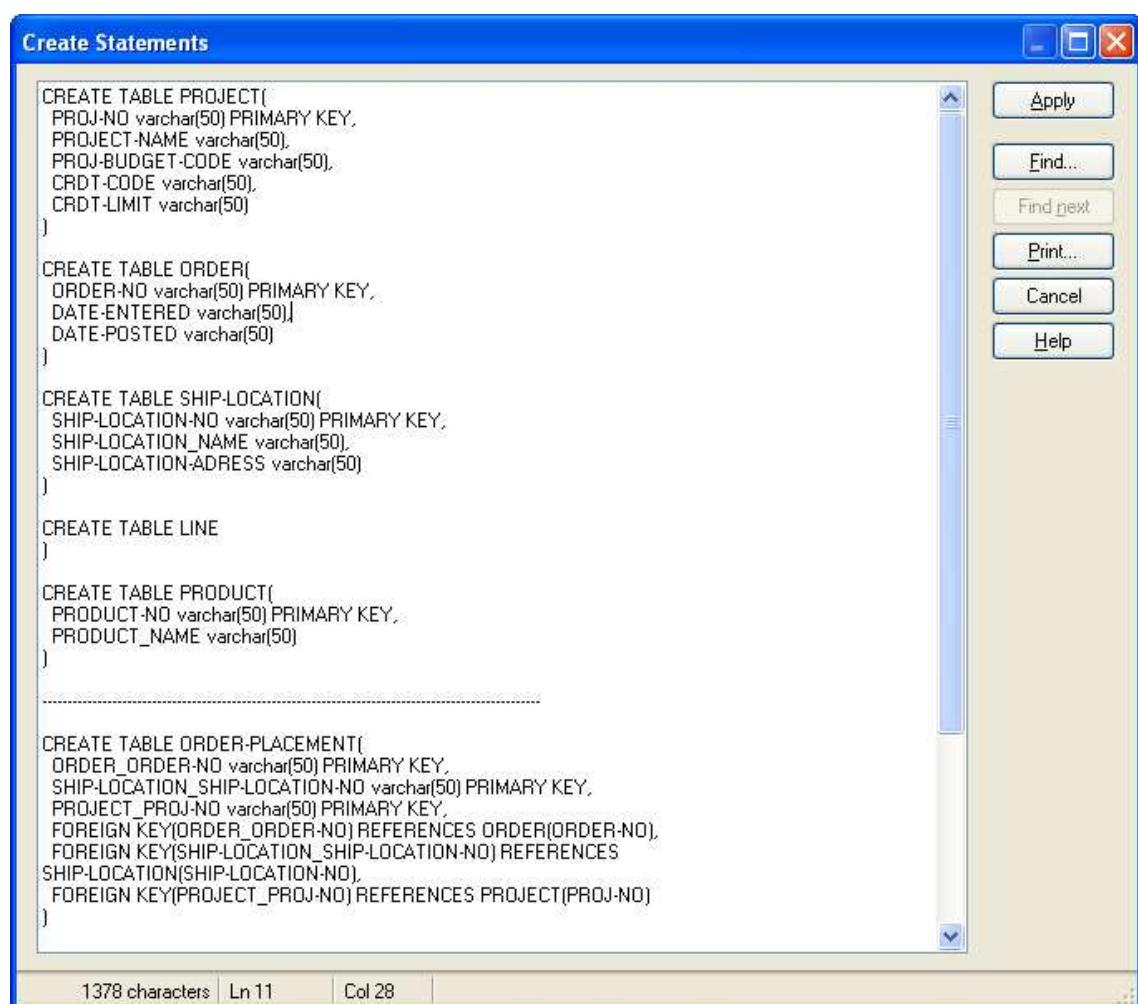
In the following table the relevant modelling classes and relation classes are briefly discussed.

 Base Relation	Base Relation	Base relations are tables of the relational schema. They cannot be derived by queries made against other relations.
---	----------------------	---

Figure 26 Modelling Classes of the Modeltype „Base Relation Pool“

5.4.3. Techniques

To generate the logical schema the menu entry “Tools|Generate Logical Schema” is to be activated. In a next step the relevant E-R model needs to be chosen. The SDBD Tool automatically generates the SQL Create Statements and presents them via info box (see Figure 27).



```

CREATE TABLE PROJECT(
  PROJ-NO varchar(50) PRIMARY KEY,
  PROJECT-NAME varchar(50),
  PROJ-BUDGET-CODE varchar(50),
  CRDT-CODE varchar(50),
  CRDT-LIMIT varchar(50)
)

CREATE TABLE ORDER(
  ORDER-NO varchar(50) PRIMARY KEY,
  DATE-ENTERED varchar(50),
  DATE-POSTED varchar(50)
)

CREATE TABLE SHIP-LOCATION(
  SHIP-LOCATION-NO varchar(50) PRIMARY KEY,
  SHIP-LOCATION-NAME varchar(50),
  SHIP-LOCATION-ADDRESS varchar(50)
)

CREATE TABLE LINE
)

CREATE TABLE PRODUCT(
  PRODUCT-NO varchar(50) PRIMARY KEY,
  PRODUCT_NAME varchar(50)
)

CREATE TABLE ORDER-PLACEMENT(
  ORDER_ORDER-NO varchar(50) PRIMARY KEY,
  SHIP-LOCATION_SHIP-LOCATION-NO varchar(50) PRIMARY KEY,
  PROJECT_PROJ-NO varchar(50) PRIMARY KEY,
  FOREIGN KEY(ORDER_ORDER-NO) REFERENCES ORDER(ORDER-NO),
  FOREIGN KEY(SHIP-LOCATION_SHIP-LOCATION-NO) REFERENCES
  SHIP-LOCATION(SHIP-LOCATION-NO),
  FOREIGN KEY(PROJECT_PROJ-NO) REFERENCES PROJECT(PROJ-NO)
)

```

1378 characters | Ln 11 | Col 28

Figure 27 SQL Create Statements

Furthermore all derived base relations are generated into a ‘Base Relation Pool Model’ (Diagram). Each base relation is documented including its attributes. Figure 28 depicts the base relations

derived from the E-R diagram (discussed in chapter 5.3.3). For the base relation “PROJECT” its attributes (columns) are shown.

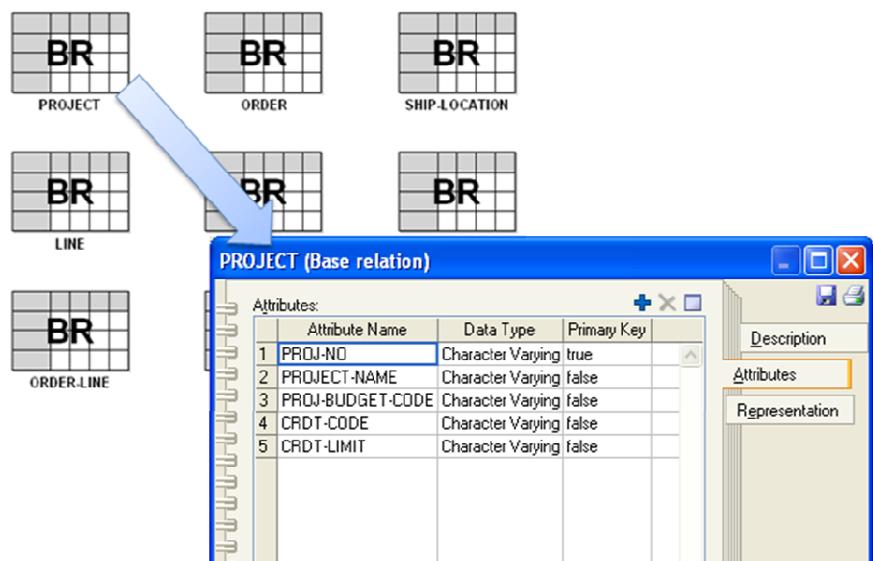


Figure 28 Base Relations and Attributes

5.5. Task Emulation & Optimisation

The purpose of this phase is to obtain the design and specification of the software that performs the tasks before any database implementation starts. In other words, before creating a schema in the DBMS, the application programming is fully specified. It is needed to verify that the tasks designed during the ‘System Analysis’ can be executed against the Logical Schema Design of the previous step. The final optimisation step includes, index selection, view materialisation selection, and tuning of the physical data layer.

Figure 29 depicts the key concepts of this phase.

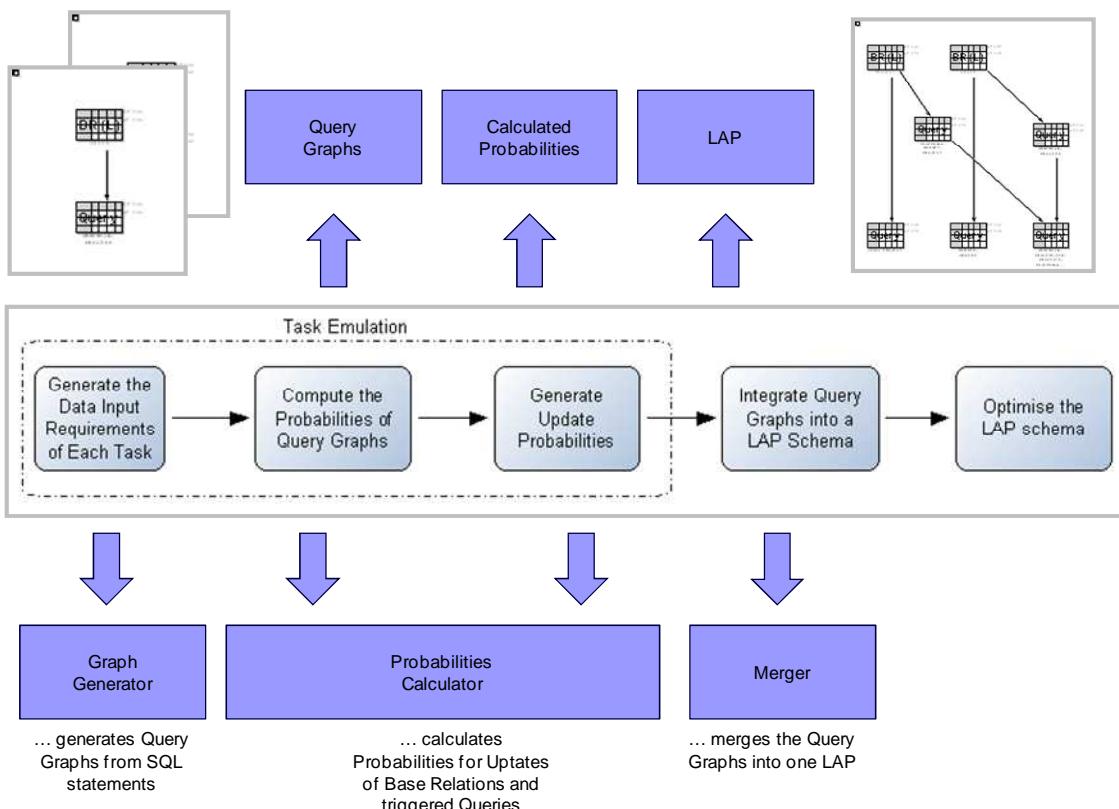


Figure 29 Key Concepts of 'Task Emulation & Optimisation'

5.5.1. Steps

Step 1: Generate the Data Input Requirements of Each Task

In the first step the data input requirements of each task are obtained. The relevant queries/updates are identified by analysing the input and output relations of each task. This step is again supported by utilising relation tables (see chapter 5.3.3).

The input requirements are represented via query graphs. Output requirements are represented via updates to base relations.

- Define Query Graphs.* Each task is emulated using the logical schema generated in the previous phase. The data input required by each task is expressed by queries against the logical schema. For describing the queries a graphical representation - the 'Query Graphs' – is introduced. Via the functionality 'Graph Generator' query graphs can be automatically derived from the SQL statements. Each query graph is stored in the form of a model of type "Logic Access Path Diagram" (Diagram icon) in the model catalogue (see Figure 30).

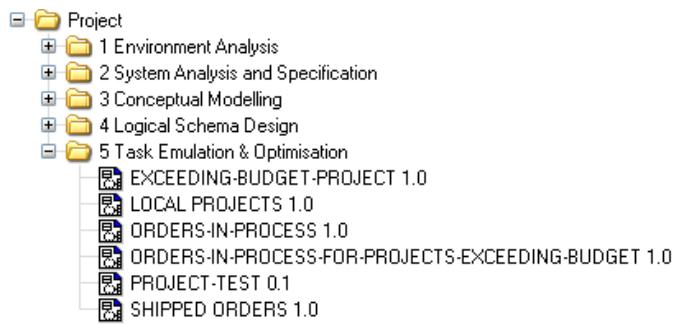


Figure 30 Logical Access Path Models, Query Graphs

Figure 31 illustrates examples of query graphs using the proposed graphical notation (see chapter 2 for details).

As discussed in the SDBD method any relational query can be represented by such a query graph because the set (presented in chapter 2) includes all relational operators.

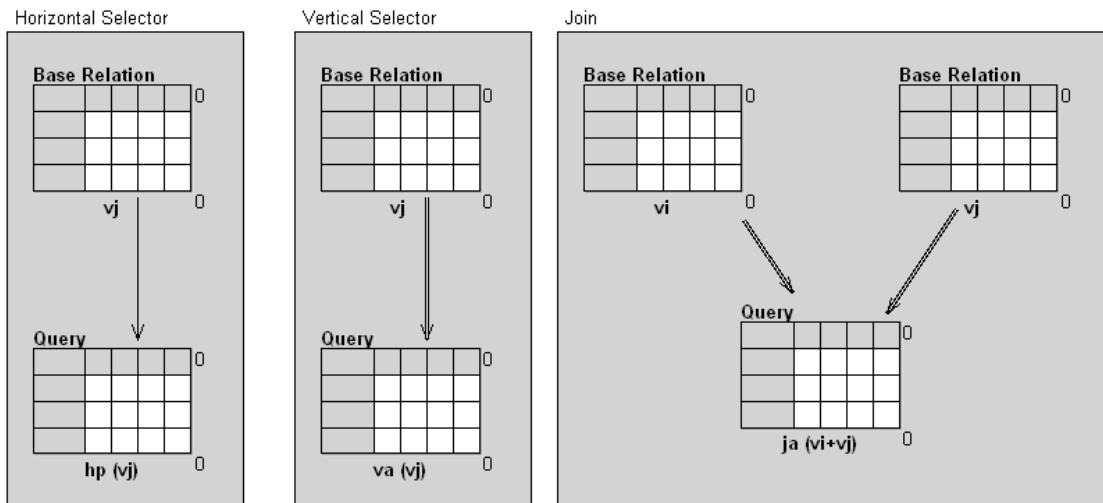


Figure 31: Query Graph Notation

- b) *Assign Query Graphs to Tasks.* Each of the query graphs is assigned to its corresponding task object(s). Figure 32 depicts a task, its notebook and the assigned query graphs.

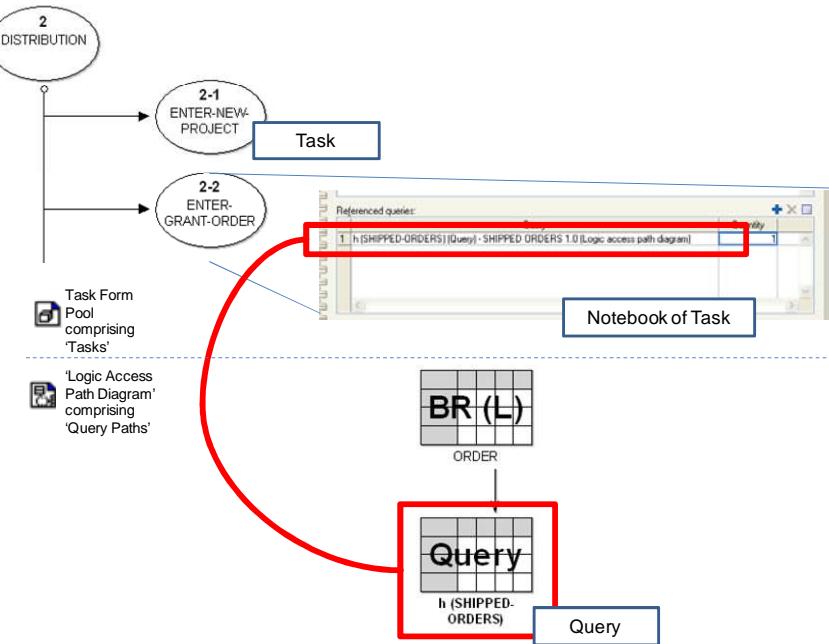


Figure 32: Assignment of Query Graphs to Tasks

Step 2: Compute Probabilities of Query Graphs

The second step is to compute the probabilities of ‘Query Graphs’. The global probability of a query graph is the probability of executing the corresponding query against the database within any task. This computed query probabilities are then used during optimisation.

In order to calculate the global probabilities for the query graphs

- the frequency of each task flow,
- the frequency of each task within a task flow and
- the number of times each query is executed when executing a task

need to be specified. These input parameters are specified with the model attributes of the ‘Task Flow Models’ and within the notebook of Task (Links) and of Tasks.

Figure 33 depicts exemplarily the relevant section of the notebook of the modeling class ‘Task (Link)’.



Figure 33 Specifying Task and Query Frequency

According to the algorithm proposed in the SDBD method, the *SDBD Tool* calculates the probabilities. The computed query probability is automatically assigned to the corresponding query graph. Figure 34 displays a query graph incl. its computed query probability.

Step 3: Generate Update Probabilities.

Base relations are those that cannot be derived by queries made against other relations. The probabilities of updates on base relations are obtained from the task's outputs. Again the SDBD method presents an algorithm for calculating the global update probabilities. This mechanism is implemented in the *SDBD Tool*.

The computed update probability is automatically assigned to the query graphs. Figure 34 depicts a query graph incl. the computed probabilities.

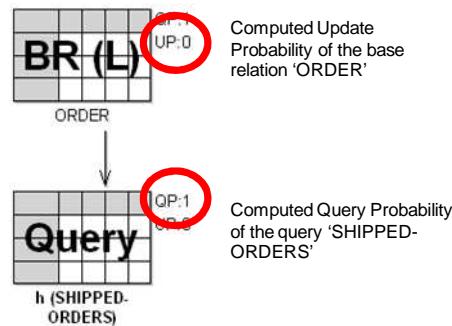


Figure 34 Query Graph incl. Computed 'Query Probabilities' and Computed 'Update Probabilities'

Step 4: Integrating Query Graphs into a LAP Schema

In this step the query graphs are merged into a hierarchically structured query tree, representing base relations (tables of the database) and queries. By doing this, the overall probability of queries and updates is computed, by summing up the initial probabilities of each query table. Again this is done automatically by choosing all relevant Query Graph Models and merging these into a global query graph, according to the rules stated in the SDBD method.

Figure 35 depicts an example of the integration of two query graphs into a global query tree – the LAP Schema.

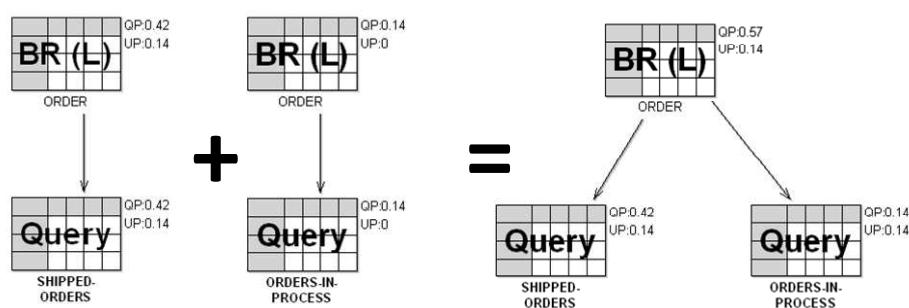


Figure 35 Integrating Query Graphs into a LAP Schema

Step 5: Optimising the LAP Schema

The goal of this step is to enhance the execution efficiency of the frequently used (or important) tasks by explicitly indexing the subschemata they operate on. For this phase the SDBD method introduces a cost model determining the costs of querying views and updating indexes.

The cost model is based on the following elements:

- the set of the derived views;
- the sizes of the indexes for those views, when the views are explicitly supported (cardinality times the size of the pointer);
- the construction cost of the derived views assuming that their index is not available;
- the probabilities of making queries against those derived views;
- the update cost of the indexes; and
- the probabilities of the updates.

This cost model will allow for obtaining the optimised logical access path schema and the application programs (task emulations). For a detailed discussion on the cost model refer to chapter 2. Algorithms for obtaining the best index allocation for any of these optimisation functions are computationally very difficult (NP-complete). However, heuristics and knowledge on the accessing provided by the logical access path schema can help optimization. A heuristically directed algorithm, which guarantees that whenever a solution is reached, it is the optimal allocation for the LAP schema, is discussed in chapter 2.

Note:

The step ‘Optimising the LAP Schema’ currently is not supported in the Version 1.0 of the SDBD Tool. Nevertheless the SDBD Tool will be enhanced in future versions (see chapter 7).

5.5.2. ‘Semi-Formal’ Notation

Each task is emulated using the logical schema generated by the previous phase.

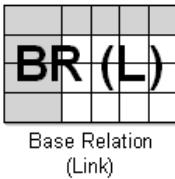
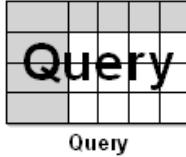
 BR (L) Base Relation (Link)	Base Relation (Link)	<p>This modelling class is used as placeholder for objects of type ‘Base Relation’. Each ‘Base Relation (Link)’ needs to be assigned to its original ‘Base Relation’ object within the ‘Base Relation Pool’.</p> <p>Base Relations reflect the data output requirements (update requirements) of tasks.</p>
 Query Query	Query	<p>Queries reflect the data input requirements of tasks.</p>
	Link	<p>Via this relation the subschemata (query graphs) are defined.</p>

Figure 36 Modelling Classes of the Modeltype ‘Logical Access Path Diagram’

5.5.3. Techniques

Graph Generator

The ‘Graph Generator’ is activated via ‘Tools| Generate Logic Access Path Diagram’. The input/output data requirements of each task are obtained by writing queries against the Logical Schema, defined in the previous phase.

Figure 37 depicts the input box for entering the query, which is to be written in SQL.

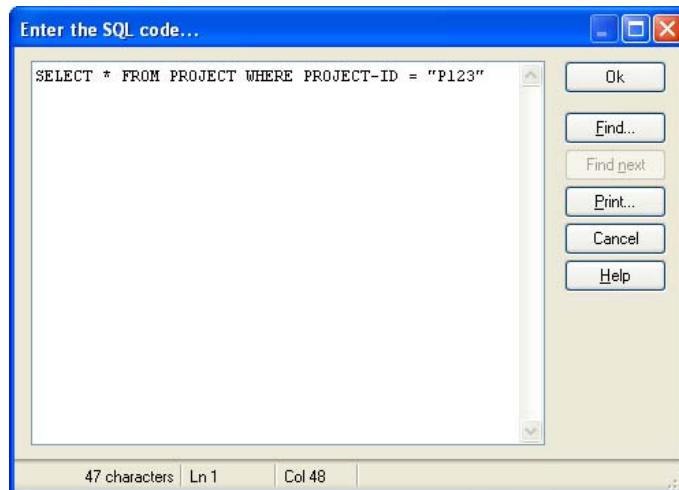


Figure 37 Input Box for the Specification of ‘Query Graphs’

The *SDBD Tool* parses this query and automatically generates an appropriate ‘Query Graph’. Figure 38 depicts the resulting ‘Query Graph’ which is documented within a ‘Logical Access Path Model’.

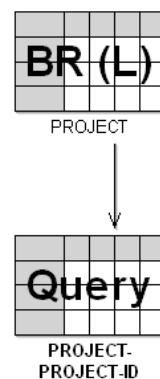


Figure 38 Automatically Generated Query Graph

Note:

In the version 1.0 of the SDBD Tool only simple SQL queries are parsed. Complex queries need to be modelled manually.

Nevertheless the SDBD Tool will be enhanced in future versions (see chapter 7).

Probabilities Calculator

The probabilities of each ‘Query’ against Base Relations and the probability of each ‘Update’ of a Base Relation can be calculated by triggering the functionality ‘Tools|Compute Probabilities’. Within the appearing select box all relevant ‘Task Flow Diagrams’ need to be selected. Through the dependency ‘Task (Link) – Task – [Query | Base Relation (Link)]’ the probabilities are calculated and assigned.

Model Merger

The LAPs of the single queries are collected and integrated into a schema, which is then optimised for maximum efficiency on both retrieval and updates. Via choosing the menu entry ‘Tools|Merge Query Graphs’ a select box – offering all existing query graphs - appears (see figure 39).

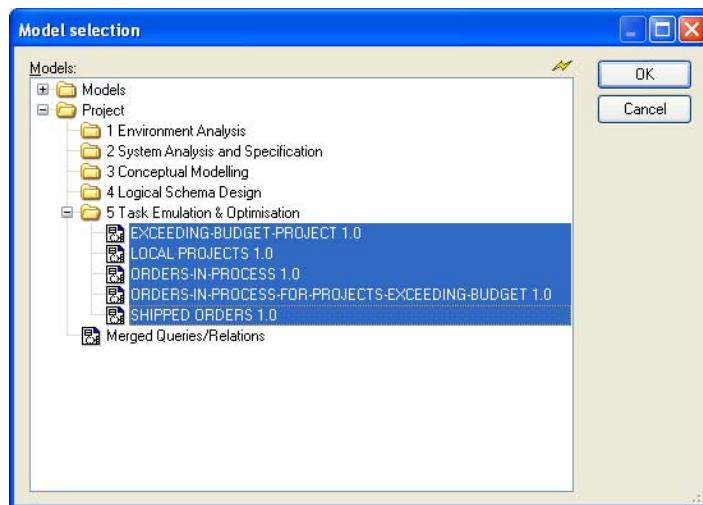


Figure 39 Choosing Single Query Graphs

The user selects all relevant query graphs. A merged query graph (complete LAP) is automatically generated and stored within a ‘Logic Access Path Diagram’. Figure 40 presents an example.

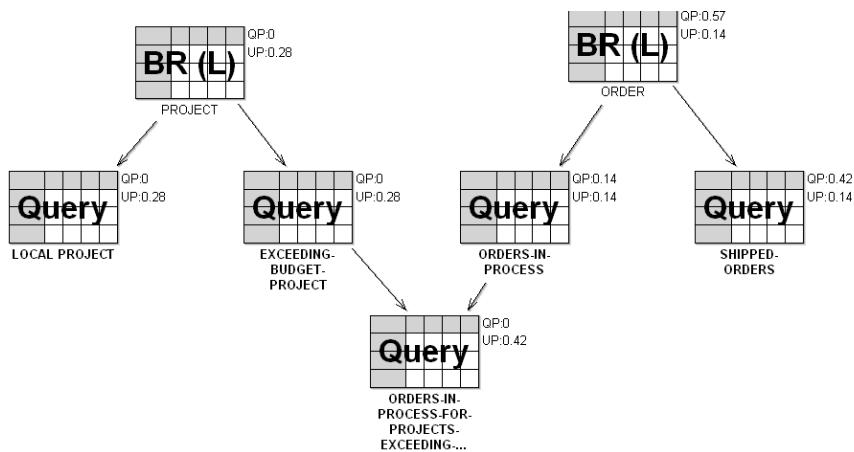


Figure 40 Integrated Logical Access Path

6. Best Practices

6.1. Phase 1 – Environment and Requirements Analysis

Figure 1 depicts an example of an Information Flow Diagram. It contains

- the main processes of the system: P1 PROCUREMENT, D2 DISTRIBUTION and I3 INVENTORY,
- the documents that carry the data between components of the organisation; and
- the defined system boundary.

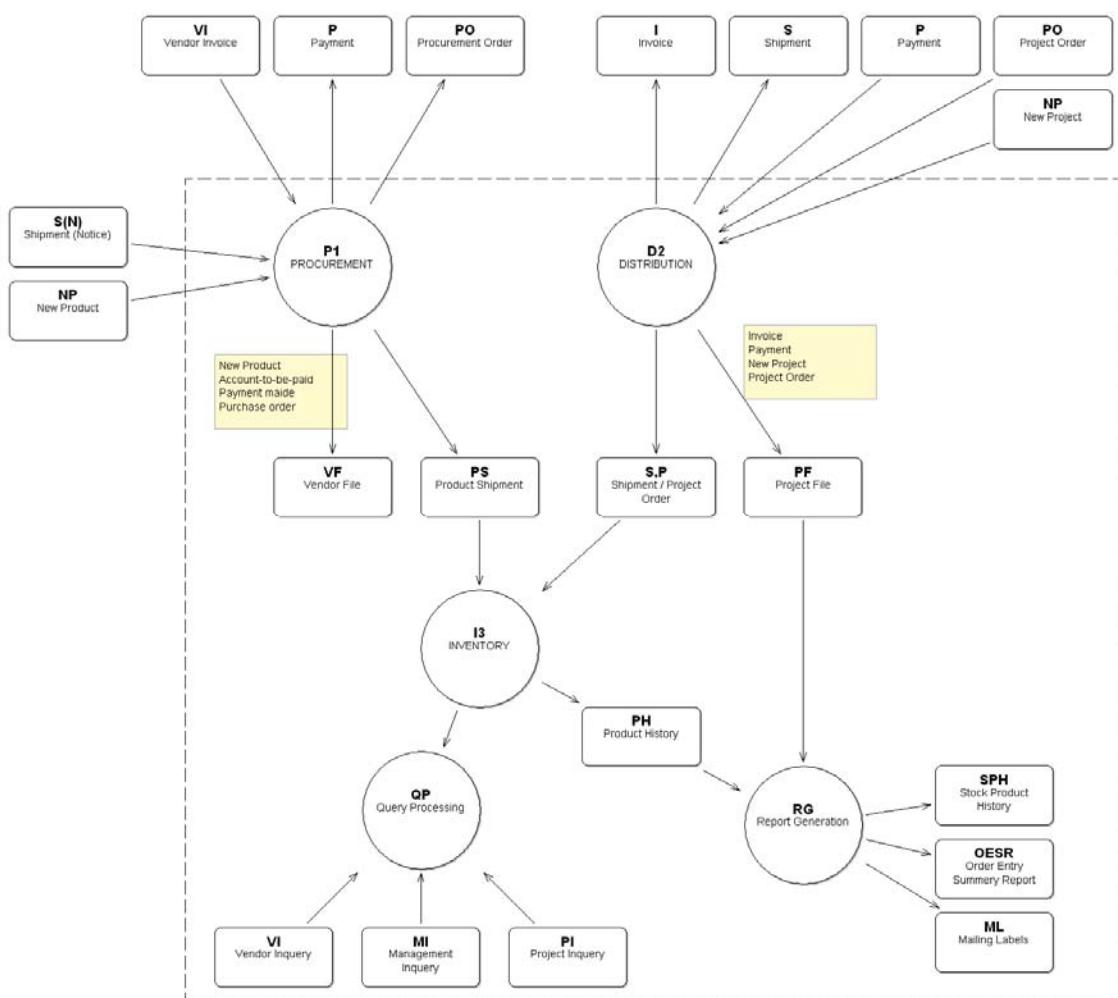


Figure 41 Information Flow Diagram

6.2. Phase 2 – System Analysis and Specification

Figure 42 illustrates an example of a ‘Task Flow Diagram’. The depicted ‘Task Flow Diagram’ shows the details of the process “D2 DISTRIBUTION”. It comprises tasks like “ENTER-NEW-PROJECT”, “ENTER-GRANT-ORDER” and “ORDER-PICKING”. For each of the tasks input and output documents are specified. Those are assigned to the tasks via the relation class ‘Information Flow’.

Prerequisite to the task “ENTER-NEW-PROJECT” is a filled-in ‘NEW-PROJECT-FORM’. Output is a new ‘PROJECT-RECORD’.

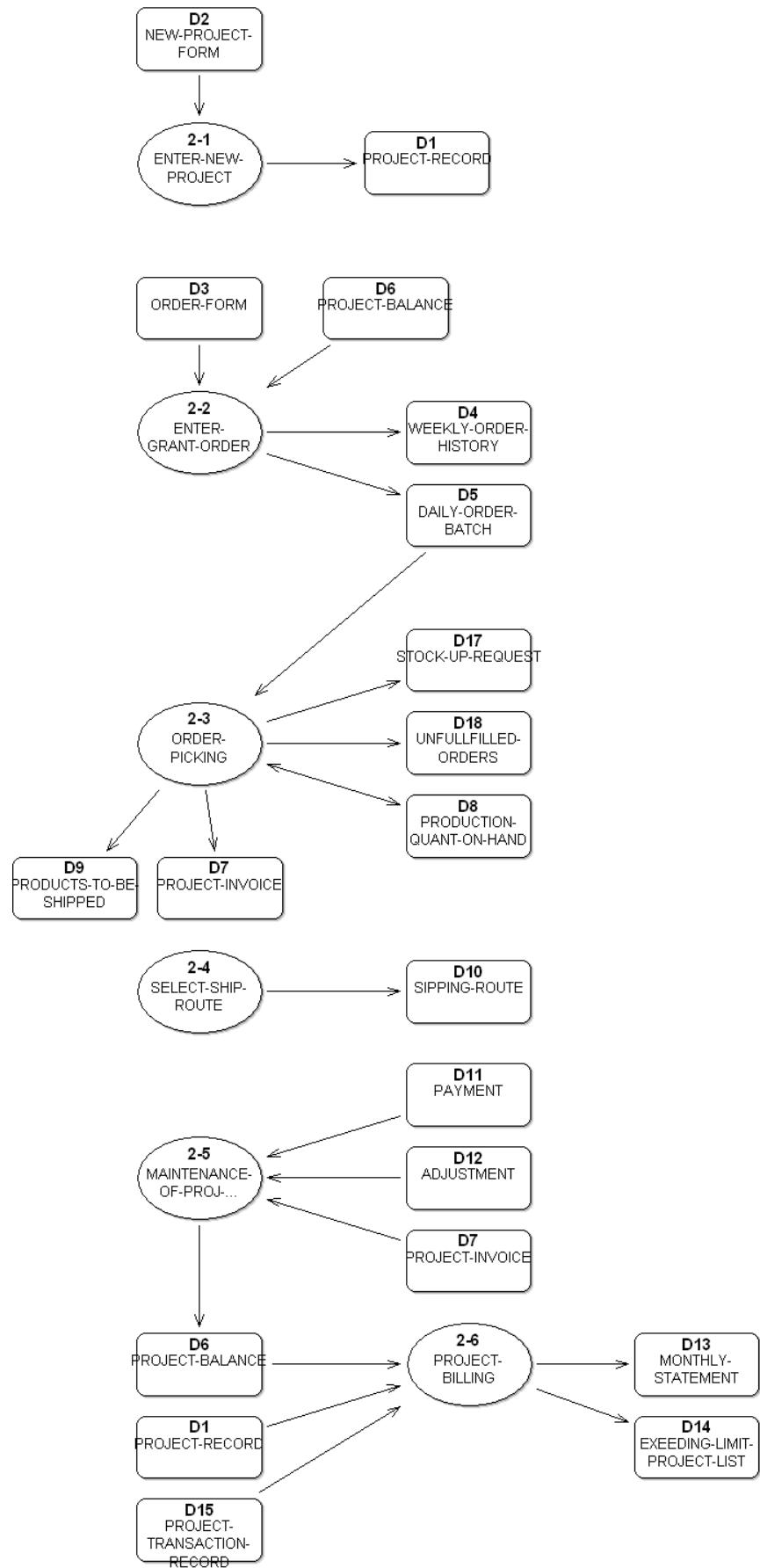


Figure 42 Task Flow Diagram

Figure 43 depicts the associated pool models.

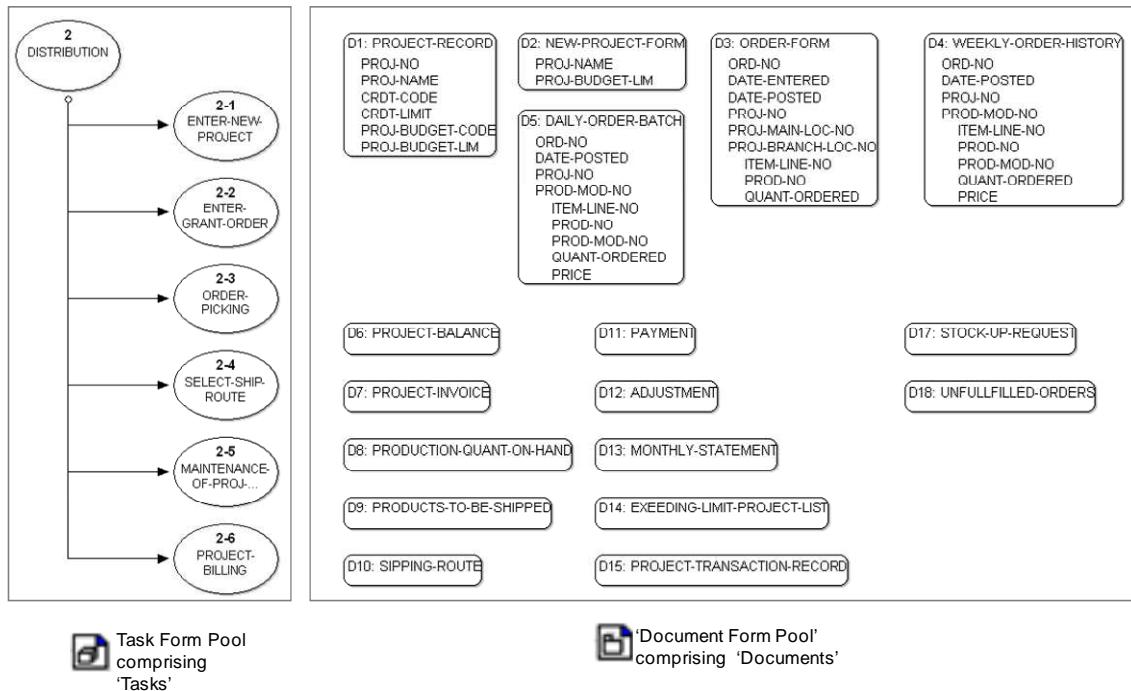


Figure 43 Pool Models

Each task is described in detail. The following table shows exemplarily the attributes of the task 'ENTER-GRANT-ORDER'.

Task-number:	2-2
Task-name:	ENTER-GRANT-ORDER
Performer:	Tom Keener, Distribution Coordinator
Authorization:	Scott Farley
Purpose:	Enters orders in the database and grants orders
Enabling-conditions:	Arrival of ORDER-.FORM (03)
Description:	ORDER-FORMS (D3) are distributed and- recorded. Balance of ordering project is checked: to determine whether order should be granted. Order history is generated every week.
Frequency:	300 per day
Duration:	N/A
Importance:	Very important
Maximum delay:	2 days
Input:	D3 (ORDER-FORM) 06 (PROJECT-BALANCE) D5 (DAILY-ORDER-BATCH)
Output:	D6 (PROJECT-BALANCE) 05 (DAILY-ORDER-BATCH) D4 (WEEKLY-ORDER-HISTORY)
Use of documents:	All columns of 03, D6, D5

Operations-performed:	Upon receipt of D3, it is numbered and distributed. Information on D3 is then entered into the database. The balance of the ordering project (06) is updated and checked. If the project balance does not exceed its limit, the order is granted. Every morning all granted orders of the previous day are batched in (D5) together for a pickup. Every Friday D4 is generated.
Subtasks:	ORDER-FORMS-RECEIVE-DISTRIBUTE ENTER-ORDER-INFO UPDATE-PROF-BALANCE INSUFFICIENT-BUDGET-HANDLING ORDER-HISTORY-GENERATION
Error-conditions:	Operation of this task halts if the information of the projects and products does not match with the data in the database.

6.3. Phase 3 – Conceptual Modelling

Figure 44 depicts the EPRs. Entities like ‘PROJECT’, ‘SHIPPING-LOCATION’, and ‘ORDER’ have been derived from the ‘Task Flow Diagrams’ and the Relation Tables. Each entity has attributes assigned. The entity “PROJECT” comprises attributes like ‘PROJ-NO’, ‘PROJECT-NAME’ and ‘CRDT-CODE’.

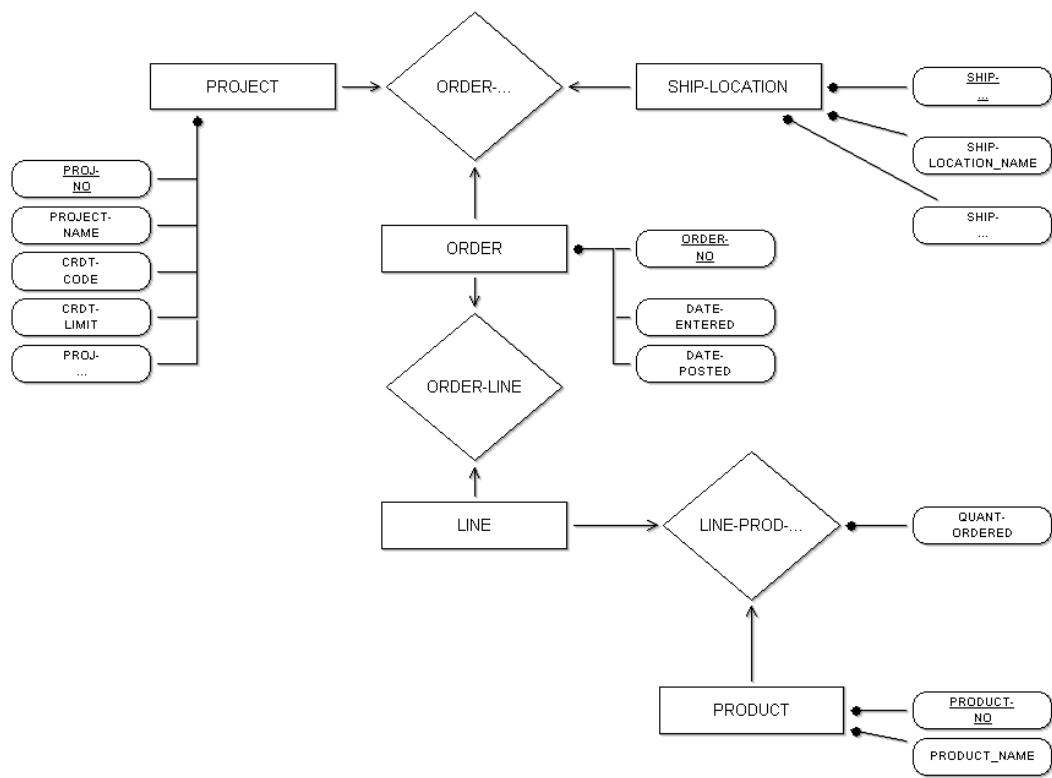


Figure 44 Example of an E-R Diagram

6.4. Phase 4 – Logical Schema Design

6.4.1. Base Relation Pool

Figure 45 depicts the logical schema, automatically derived from the ER-Diagram of the previous chapter, in the form of a ‘Base Relation Pool’.

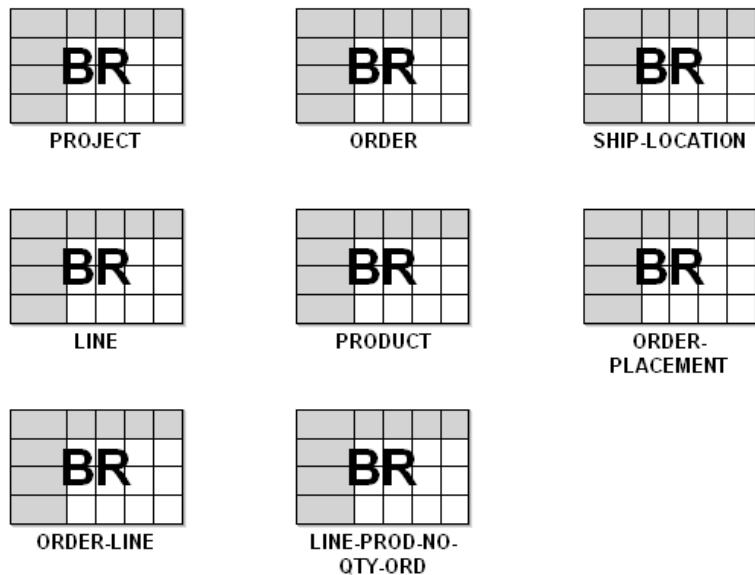


Figure 45 Base Relation Pool

LINE

(no attributes specified)

LINE-PROD-NO-QTY-ORD

Attribute Name	Data Type	Primary Key
PRODUCT_PRODUCT-NO	Integer	X
QUANT-ORDERED	Integer	

ORDER

Attribute Name	Data Type	Primary Key
ORDER-NO	Integer	X
DATE-ENTERED	Date	
DATE-POSTED	Date	

ORDER-LINE

Attribute Name	Data Type	Primary Key
ORDER_ORDER-NO	Integer	X

ORDER-PLACEMENT

Attribute Name	Data Type	Primary Key
ORDER_ORDER-NO	Integer	X
SHIP-LOCATION_SHIP-LOCATION-NO	Integer	X
PROJECT_PROJ-NO	Integer	X

PRODUCT

Attribute Name	Data Type	Primary Key
PRODUCT-NO	Integer	X
PRODUCT_NAME	String	

PROJECT

Attribute Name	Data Type	Primary Key
PROJ-NO	Integer	X
PROJECT-NAME	String	
PROJ-BUDGET-CODE	String	
CRDT-CODE	String	
CRDT-LIMIT	String	

SHIP-LOCATION

Attribute Name	Data Type	Primary Key
SHIP-LOCATION-NO	Integer	X
SHIP-LOCATION_NAME	String	
SHIP-LOCATION-ADRESS	String	

6.4.2. Automatically Generated SQL Create Statements

In the following the automatically generated SQL Create Statements are shown.

```

CREATE TABLE PROJECT(
    PROJ-NO integer(50) PRIMARY KEY,
    PROJECT-NAME string(50),
    PROJ-BUDGET-CODE string(50),
    CRDT-CODE string(50),
    CRDT-LIMIT string(50)
)

CREATE TABLE ORDER(
    ORDER-NO integer(50) PRIMARY KEY,
    DATE-ENTERED date(50),
    DATE-POSTED date(50)
)

CREATE TABLE SHIP-LOCATION(
    SHIP-LOCATION-NO integer(50) PRIMARY KEY,
    SHIP-LOCATION_NAME string(50),
    SHIP-LOCATION-ADRESS string(50)
)

CREATE TABLE LINE
)

CREATE TABLE PRODUCT(
    PRODUCT-NO integer(50) PRIMARY KEY,
    PRODUCT_NAME string(50)
)

```

```

CREATE TABLE ORDER-PLACEMENT(
    PROJECT_PROJ-NO integer(50) PRIMARY KEY,
    ORDER_ORDER-NO integer(50) PRIMARY KEY,
    SHIP-LOCATION_SHIP-LOCATION-NO integer(50) PRIMARY KEY,
    FOREIGN KEY(PROJECT_PROJ-NO) REFERENCES PROJECT(PROJ-NO),
    FOREIGN KEY(ORDER_ORDER-NO) REFERENCES ORDER(ORDER-NO),
    FOREIGN KEY(SHIP-LOCATION_SHIP-LOCATION-NO) REFERENCES SHIP-LOCATION(SHIP-
LOCATION-NO)
)

CREATE TABLE ORDER-LINE(
    ORDER_ORDER-NO integer(50) PRIMARY KEY,
    FOREIGN KEY(ORDER_ORDER-NO) REFERENCES ORDER(ORDER-NO)
)

CREATE TABLE LINE-PROD-NO-QTY-ORD(
    PRODUCT_PRODUCT-NO integer(50) PRIMARY KEY,
    QUANT-ORDERED integer(50),
    FOREIGN KEY(PRODUCT_PRODUCT-NO) REFERENCES PRODUCT(PRODUCT-NO)
)

```

6.5. Phase 5 – Task Emulation & Optimisation

6.5.1. Generate the Data Input Requirements of Each Task

Query for selecting local projects

- a) Select Statement:
`'SELECT * FROM PROJECT WHERE PROJECT-LOCATION = "Vienna"'`
- b) Resulting LAP Subschema (Query Graph):

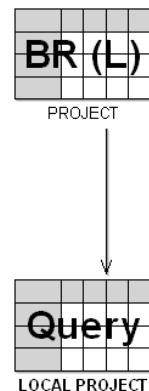


Figure 46 Query Graph 'Local Projects'

Query for selecting projects with exceeding bugets

- a) Select Statement:

'SELECT * FROM PROJECT WHERE PROJECT-BUDGET < CURRENT-PROJECT-EXPENSES'

- b) Resulting LAP Subschema (Query Graph):

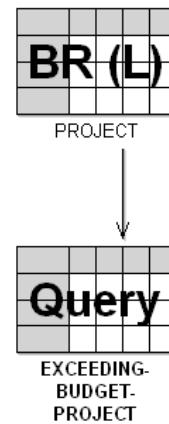


Figure 47 Query Graph 'Projects with Exceeding Budgets'

Query for selecting all orders currently processed

- a) Select Statement:

'SELECT * FROM ORDER WHERE STATE = "Processed"'

- b) Resulting LAP Subschema (Query Graph):

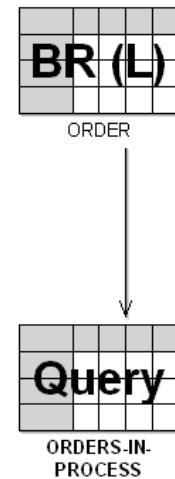


Figure 48 Query Graph 'Orders Currently Processed'

Query for selecting all orders currently shipped

- a) Select Statement:

'SELECT * FROM ORDER WHERE STATE = "Shipped"'

- b) Resulting LAP Subschema (Query Graph):

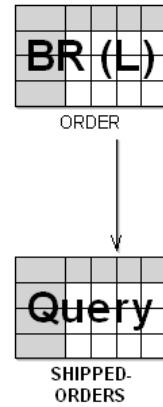


Figure 49 Query Graph 'Orders Currently Shipped'

Query for selecting all orders in process for projects with exceeding budget

- a) Select Statement:

```
'SELECT * FROM ORDER WHERE STATE = "Processed"
INNER JOIN PROJECT WHERE PROJECT-BUDGET < CURRENT-PROJECT-EXPENSES
ON ORDER.ORDER-NO=PROJECT.ORDER-NO'
```

- b) Resulting LAP Subschema (Query Graph):

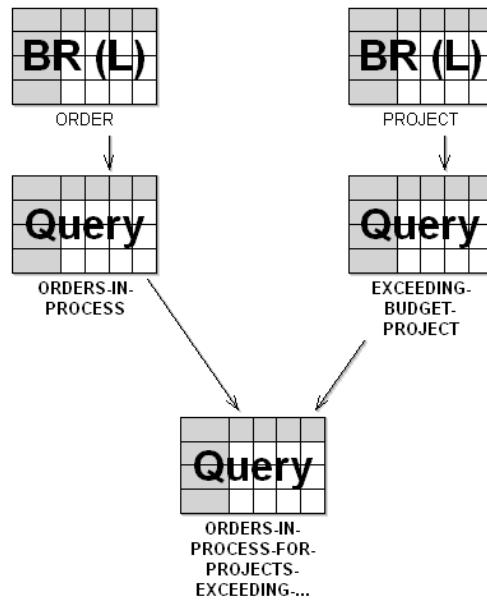


Figure 50 Query Graph 'Orders in Process for Projects with Exceeding Budget'

6.5.2. Compute Probabilities of Query Graphs

After Computing the probabilities for each query graph the update and query probabilities are assigned and shown within the diagrams. Figure 51 depicts exemplarily the query and update probabilities of the 'Shipped-Orders' query.

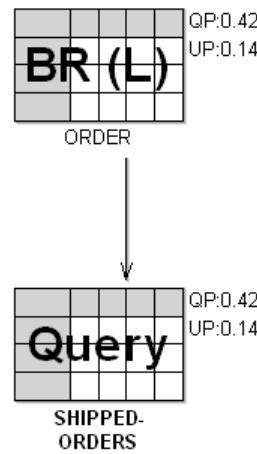


Figure 51 Shipped-Orders Query Graph incl. Probabilities

6.5.3. Integrate Query Graphs into a LAP Schema

Figure 52 depicts the resulting LAP Schema.

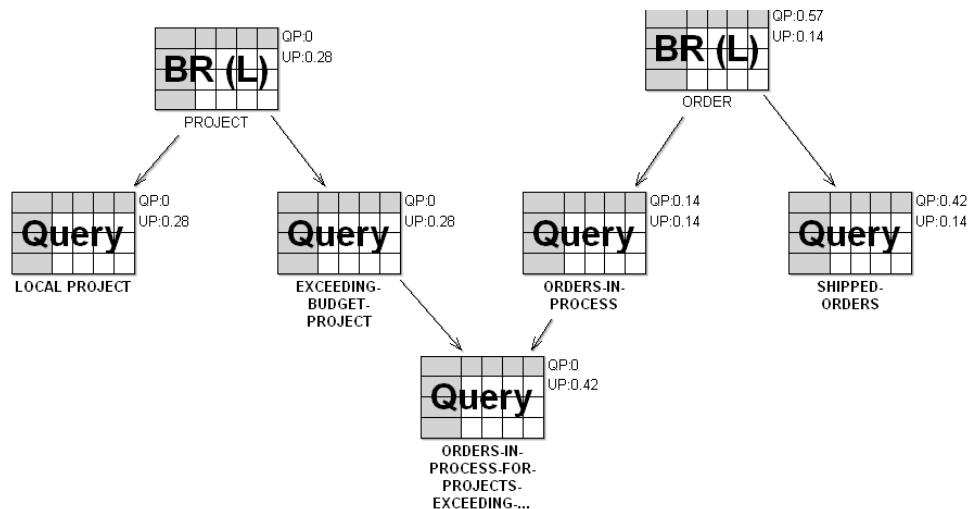


Figure 52 Resulting LAP Schema

The complete LAP schema is the input for the last phase – for optimising the LAP schema.

7. How to extend the SDBD Tool



Be Part of the Open Model Community!

Access to ADOxx 1.0 and to the SDBD Tool:

www.openmodels.at