PROPOSAL FOR A PROTOTYPING KIT

Jan Jantzen
M.Sc., Ph.D.
Skovgaardsvej 18 B st.tv.
DK-2920 Charlottenlund
DENMARK

Research supported by The Danish Council
for Scientific and Industrial Research and
Queen's University, Canada

Abstract

The three-levelled ANSI/SPARC architecture for database systems forms a framework on which software prototypes can be built. The external level corresponds to screen panels, the conceptual level to the data model, and the internal level to the stored files of the prototype. The paper identifies prototype design tools and building-blocks with respect to this architecture. A screen design tool illustrates ideas using nested arrays. A novel aspect is, that the paper takes a step towards an operational formulation of the prototyping approach by emphasizing the computerized implementation.

Introduction

Design of engineering, managerial, and economic information systems is intuitive by nature, and it is often useful to build an operational system model in much the same way that an engineer builds a prototype.

Formally, software prototyping is a stepwise and iterative design technique characterized by practical experiments with operational system models. Beyond that, it is hard to define what prototyping is.

It has, however, been widely and effectively employed: one early report is by Gomaa & Scott (1980), who built a prototype of a management and control system in APL. APL is a useful language since program development is fast, interactive, and the programs are easy to change.

A prototyping approach, which combines a technique and a tool, is described by Mason & Carey (1983). The technique is architecture-based, and the designer works inward from the external appearance of the system. The tool (ACT/1) includes a scenario tool to specify input and output, it contains screen drivers, and communication between screen-defined variables and COBOL or PL/1 defined variables is provided using symbolic variable names.

An APL tool with similar capabilities for screen communication is APE (Application Prototype Environment, see Sorensen & Hagensen, 1979, for a description of an early version). This tool (which has inspired parts of this work) has facilities for panel design, building-blocks for using the panels, and facilities for file handling. It thus aims at the end-user interface as well as the stored database interface, but it does not include facilities for database modelling.

The database modelling tool CANTOR, programmed in APL, fits into this gap (see Schmidt & Theilgaard, 1980, for an overview presented at APL80; see Schmidt, 1980, for details). CANTOR rests on the Relational Model. It has been further developed into a nested array version, which rests on the socalled Roster Model -- essentially an array-theoretic relational model, developed by Schmidt & Jenkins (1982).

CANTOR and APE have been used to design two prototypes, resulting in a proposal for a prototyping approach consisting of three elements: an iterative design procedure; a set of tools; and an information system for controlling project resources, time, and costs (Jantzen, 1982).

The Roster Model is the foundation for the RIPO tool (Rosters In, Prototype Out), which automatically sets up a prototype provided the test data are formatted in a special way (described in these proceedings by Hendren (1984)).

With the appearance of powerful, high level languages for representing and manipulating nested arrays (APL, NIAL), it now seems feasible to formulate the prototyping approach in a systematic and operational way. The purpose of the present paper is to outline a strategy for reaching this goal.

Prototype architecture

The ANSI/SPARC architecture (ANSI/X3/SPARC in ISO, 1982) for database systems is here used as a framework for building prototypes. The architecture is a hierarchy of three levels: external, conceptual, and internal level.

The external level is the screen panels and hardcopy reports of the prototype. This level concerns the end-user's view of the system. A screen panel (report) is a screenful (page) of data with a predefined, fixed format. Each panel (or report) is a particular end-user view; it displays a subset of the entire information contained in the prototype. Each external view is described in an external schema, i.e., a definition of data types, geometry, and labels.

The conceptual level is the data model, for example a Relational Model, which is a representation of the entire information content of the prototype. A conceptual schema defines the objects of the data model, for example relations, attributes, and domains.

The internal level concerns the way data are stored and concentrates on the physical representation of and access to the data. An internal schema defines the level.

Two mappings, an external-conceptual and a conceptual-internal mapping, transform data representations into their counterparts when data flow from one level to another. The purpose is to secure data and device independence by protecting the conceptual level from changes in panel layouts, terminal hardware, or record layouts; the changes are absorbed in the mappings.

This architecture, which is in agreement with the recommendations from the International Organisation for Standardization (ISO, 1983), forms the basis for identifying a set of prototype design tools.

PROTOKIT proposal

The future aim is to build a kit of prototyping tools (a PROTOKIT), from which the designer selects or designs components for a prototype. A proper environment is a "Systems Planning Laboratory" for practical experiments, production of alternative solutions, and quick development (Hansen & Schmidt, 1981). A 'Prototyper's Handbook' is to accompany the PROTOKIT with instructions on how to solve isolated, frequently encountered subproblems.

Figure 1 shows a proposal for a PROTOKIT. The following paragraphs describe the integration of tools into the architecture.

The external level is designed by means of a panel design tool. The tool sets up formalised definitions (an external schema) of panels, which are then used by panel manipulation operations for the data communication between panels and workspace. The panel design tool and the panel manipulation operations interface via the external schema. Hardcopy reports are defined using a report generator. The report generator accesses arrays of the data model and consults data definitions in the conceptual schema. This requires a well-defined interface between report generator and the data modelling tool.

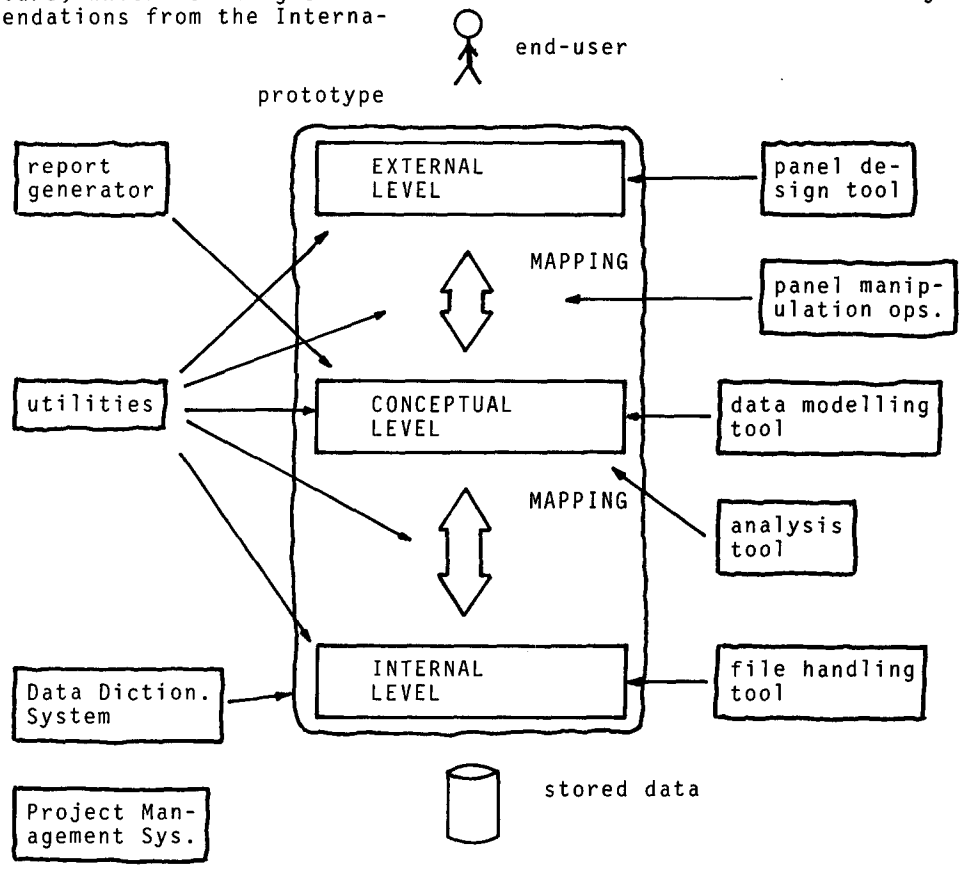

Figure 1. Tools of PROTOKIT applied to the prototype.

The conceptual level is defined by means of a data modelling tool. Because of the data independence, due to the mappings, the data modelling tool can be regarded as an independent tool, not integrated with other tools.

The internal level, is defined by the file handling tool independently of other tools. The internal schema holds information like position, type, and length of fields in records. File manipulation operations provide facilities for reading and writing records using the internal schema.

All three schemas are controlled by a data dictionary system in the PROTOKIT. The system communicates with the design tools. It produces documentation in the form of listings and cross-references between schemas, program names, variable names, file names, panel names, etc. An important role of the system is to secure that the prototype works with existing data definitions from the company environment as far as possible -- rather than just creating new data.

Analysis of the prototype structure is performed independently of the design tools. Representing the structure of the conceptual level as a digraph (Jantzen, 1982) this tool -- basically a library of digraph operations -- will test the consistency by examining predecessor (successor) relationships, reachability properties, cycles, and so on.

An independent project management system is for planning and control of the prototyping project. The system supports the designer in estimating costs and time of the implementation of the prototype, as well as the operation of the final system.

The laboratory is used in an experimental and iterative manner in which the three levels are built in succession starting with the external level.

External level design

This initial prototype, which concerns the external level, is usually a dummy solution. It consists of screen panels and fictitious data only. The terminal operator can jump from one panel to another, but there need not be any facilities for data entry. Programs are primarily for screen handling.

Figure 2 shows an example of a panel. A panel consists of rectangular fields. Each field has a well-defined size and position. Information in a field is either variable or fixed. In the figure -- a sample menu from a library system -- the time stamp '83-07-19 12:38' is an example of variable information, whereas the text 'LIBRARY SYSTEM ...' is fixed. A field is either an input field or an output field. Input fields accept information from the keyboard, output fields do not; they are protected from updates.

A panel design technique must allow the end-user to develop requirements gradually as the design evolves. To achieve this, a three step procedure may be adopted,

step 1: paint a panel
step 2: format fields
step 3: define names and types

In step one, end-user and designer discuss the appearance of a panel using the screen as a sketch-pad. They fill in test data (fictitious perhaps), experiment with the layout, and produce alternative solutions. They repeat the process until satisfied with the layout.

In step two, they single out the variable information in the panel and demarcate the rectangular areas that hold this information. To the end-user, this is the information that goes in and out of the prototype.

In step three, they classify these fields into input and output fields. They give each field a unique name for reference.

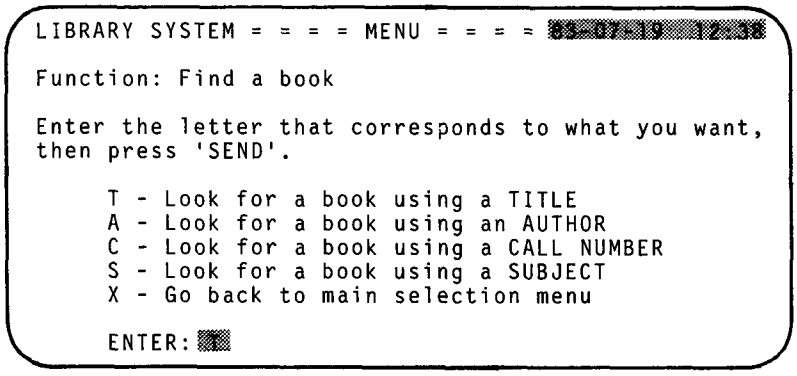

Figure 2. Library system menu.
The bottom line is an input line where the end-user types a letter indicating his choice from the menu.
The shaded areas depict variable information, the rest is fixed information.

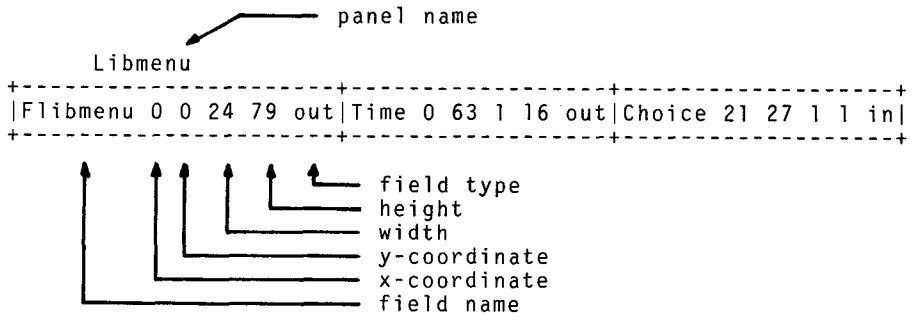

Figure 3. External schema for the library system menu.

A design tool has been devised that supports this technique. It captures necessary pieces of information during the three design steps and sets up an external schema.

Figure 3 shows a schema represented as a nested array, a list of (three) lists. The box diagram is meant as an aid to understand the data structure (presented at APL82, see Schmidt & Jenkins, 1982). The nested array representation (More, 1982) is convenient, since most objects in data processing are collections, and the rectangular arrangement is easy to analyze. In the schema each list defines a field in the panel. The first list defines the fixed information, the two remaining lists in the figure define the variable fields.

The schema gets the field name from design step three, position and size from step two, and field type from step three. Notice how this information can be gathered together into one array, since the array representation allows dissimilar data types within the same array, e.g., characters, numbers, and indivisible phrases.

During design step one, figure 4, the tool displays a blank screen initially. The operator types in text and test data. She is free to move the cursor around over the whole screen by means of cursor control buttons using the terminal as an electronic sketch-pad. She works within a text editor and uses the editor to correct, move, insert, etc.

During step two, the tool initially displays the panel defined in step one (a character matrix at this stage). The operator then types in delimiters, figure 5, ('a' and 'b' in the figure, but any character is valid); in case of one-character fields, only one delimiter is needed. She only delimits variable information, the computer handles the fixed information. She can create, modify, and delete fields by adding, moving, and deleting delimiters. She again has the text editor at disposal for this purpose.

During step three, the tool displays a list of the used delimiter symbols. The operator then defines field names and types (input or output) of each field, figure 6. The defined names become variables internally in the prototype. These variables carry the data that are communicated to and from the panel.

If the operator so wishes, she can repeat any of the three steps.

External level communication

Once the external schema is defined, communication with a panel is by means of predefined data manipulation operations. These operations are building-blocks for writing data to and reading data from the panel.

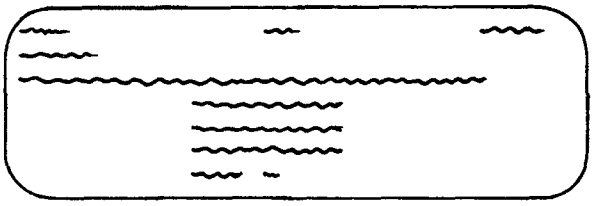

Figure 4. Step one, paint a panel.

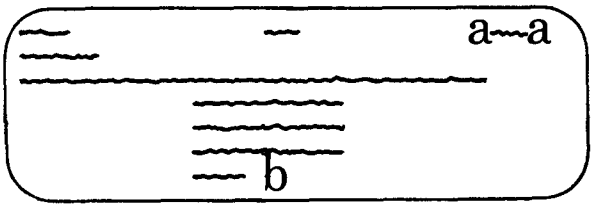

Figure 5. Step two, format fields.

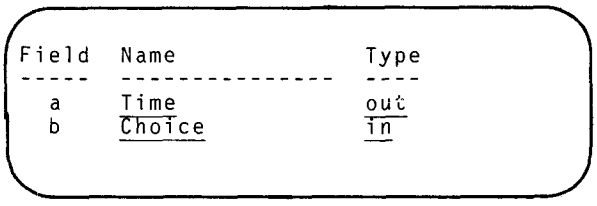

Figure 6. Step three, define names and types. Input is underlined.

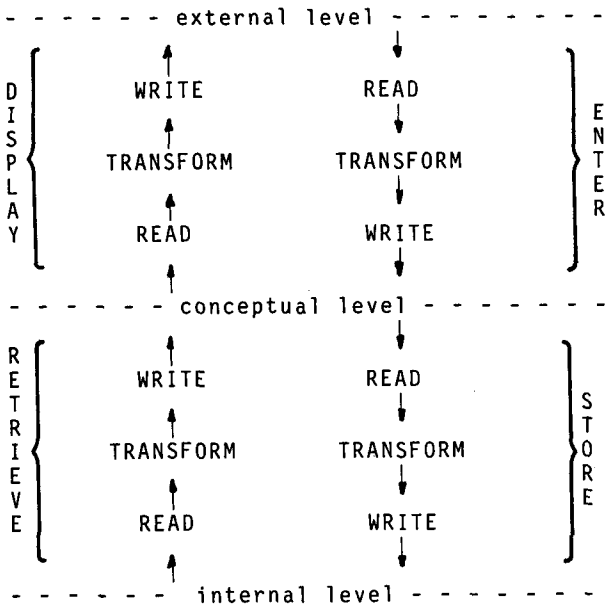

Figure 7. Data manipulation operations of the external-conceptual and the conceptual-internal mapping.

Figure 7 displays a set of operations defined according to the architecture. Notice the symmetry of the figure. Conceptually at least, transferring data from one level to another always involves the three operations: READ, TRANSFORM, and WRITE; no matter whether the purpose is panel display, data entry, storing data in a file, or retrieving data.

The paragraphs below focus on the WRITE and READ operations in connection with the external level. Operations programmed in NIAL (Nested Interactive Array Language, see Jenkins, 1983, for details) will illustrate ideas.

Writing to a panel is expressed in the statement

```
ITERATE writepanel Libmenu ;
```

where the variable Libmenu, the external schema defined previously, is the argument for the operation 'writepanel'. ITERATE, a socalled transformer, makes 'writepanel' apply to each item of Libmenu instead of the array as a whole (equivalent to the 'each' operator in APL). To illustrate, consider the case of the time stamp. The contents of the workspace resident variable named Time, i.e., the string '83-07-19 12:38', is placed in the proper location on the screen, i.e., zero'th row and 63'rd column cf. the schema (index origin is 0).

In that statement a whole panel was handled in one go. The statement

```
writepanel first Libmenu ;
```

writes only the first field of Libmenu to the screen. The operation 'first' ('take' in APL) picks the first item from

```
...                                % Comments:
Time := '83-07-04  12:38' ;        % Assign data
Choice := ' ' ;                    % Assign data
clearscreen ;                      % Blank out screen
ITERATE writepanel Libmenu ;       % Write data to screen
ITERATE readpanel Libmenu ;        % Read end-user input
IF first Choice = 'T' THEN         % Take action according
...                                % to condition
```

Figure 8. NIAL code illustrating how to combine panel manipulating operations.

Libmenu, and passes it on as an argument to 'writepanel'. Comparison of the two statements illustrates how data can be processed in parallel using ITERATE.

Reading data from a panel is expressed in the dual statement

```
ITERATE readpanel Libmenu ;
```

where the operation 'readpanel' is applied to each field of the panel. After checking whether a field is defined as an input field, it positions the cursor, reads what the end-user types, and assigns the data to a variable in the workspace.

Table I summarizes a sufficient set of building-blocks for reading from and writing to a panel.

These building-blocks can be put together in a manner that complies to the previously mentioned mappings. Consider the piece of code in figure 8. The first four lines constitute TRANSFORM and WRITE of the DISPLAY operation, and the last two lines constitute READ and TRANSFORM of the ENTER operation.

Conclusion

The strategy for formulating a systematic and operational prototyping approach rests on three aspects

- a suitable prototype architecture
- suitable tools and techniques
- a suitable host language

The three-levelled ANSI/SPARC architecture applies to prototypes and provides a framework on which prototypes can be described and built. This architecture is in agreement with the ISO recommendations. The architecture has inherent symmetries, and it is believed that these symmetries can aid the designer in structuring the programs of the prototype. The architecture also aids in identifying suitable PROTOKIT building-blocks.

The objective of the PROTOKIT is to build modular, transparent prototypes in a stepwise, consistent, and economic way. Suitable tools and techniques should evolve from existing tools and techniques. The tools must be modular and easy to comprehend, enabling simple and expedient design of prototypes.

It is recommended to choose an interac-

Table I. Panel manipulation operations in NIAL.

Class	Example	Explanation
WRITE	writepanel Fielddef	write a field to a panel
	ITERATE writepanel Panel	write whole panel
	setcursor X Y *)	position cursor in X,Y
	clearscreen *)	blank out screen
	write S *)	write string S to screen
	writescreen S *)	same, S must be char.string
	writechars S *)	same, no linefeed
READ	readpanel Fielddef	read a single field
	ITERATE readpanel Panel	read a whole panel
	readscreen P *)	reads string; P is prompt
	read P *)	evaluated readscreen

Legend: *) - primitive NIAL operations

tive, nested array language as a host language, since most objects in data processing are collections. From a designer's viewpoint, the nested array representation is convenient since

- it handles variable length data
- dissimilar data types can be mixed
- data can be processed in parallel
- table formatting is easy

It is concluded, that APL and NIAL are prototyping languages.

References

1. Gomaa,H. & Scott,D. "An APL Prototype of a Management and Control System for a Semiconductor Fabrication Facility". I: I.P.Sharp Associates (Eds.), PROC APL User's Meeting. Toronto: I.P.Sharp, 1980, 73-83.

2. Hansen,P.R. & Schmidt,F. "A Systems Planning Laboratory: Computerized Tools and Procedures. I: O.Bjorke & O.I.Franksen (Eds.), Structures and Operations in Engineering and Management Systems. Trondheim: Tapir, 1981, 267-284.

3. Hendren,L. "RIPO: An Automated Tool for Producing Prototypes from Rosters. I: PROC APL84, APL Quote Quad, 1984 (these proceedings).

4. ISO, International Organisation for Standardization. Concepts and Terminology for the Conceptual Schema and the Information Base (Publ.no. ISO/TC97/SC5 - N 695). New York: American National Standards Institute, 1982.

5. Jantzen,J. Prototyping for the End-User: An Experimental Approach Using APL. Technical University of Denmark: Elec. Power Engineering Dept., 1982, Ph.D. thesis.

6. Jenkins,M.A. The Q'Nial Reference Manual. Queen's University, Canada: Dept. Computing and Information Science, 1983.

7. Mason,R.E.A. & Carey,T.T. "Prototyping Interactive Information Systems". COM ACM, 1983, 26(5), 347-354.

8. More,T. "Rectangularly Arranged Collections of Collections". I: W.H. Janko & W. Stucky (Eds.), PROC APL82, APL Quote Quad, 1982, 13(1), 219-228.

9. Schmidt,F. Baseoperations in Factory Management Systems: Interactive Prototyping Using APL. Technical University of Denmark: Elec. Power Engineering Dept., 1982, Ph.D. thesis.

10. Schmidt,F. & Jenkins,M.A. "Array Diagrams and the Nial Approach". I: W.H.Janko & W.Stucky (Eds.), PROC APL82, APL Quote Quad, 1982, 13(1), 315-319

11. Schmidt,F. & Jenkins,M.A. Data Systems Design: The Nial Approach. Queen's University, Canada: Dept. Computing and Information Science, 1982.

12. Schmidt,F. & Theilgaard,N.B. "Baseoperations within Relational Data Models". I: PROC APL80. Amsterdam: North Holland, 1980.

13. Sorensen,U. & Hagensen,R. VS APL Prototyping. Copenhagen: IBM, 1979. General documentation: IBM order number GB21-3078.