SERIE RESEARCH MEMORANDA

ON RALLY, AN EVALUATION BASED ON CRITERIA

E. Spoor
E. Houweling
R. Geerts
R. Losekoot
M. Sprengers

Researchmemorandum 1987-33 september '87

VRIJE UNIVERSITEIT
FACULTEIT DER ECONOMISCHE WETENSCHAPPEN
EN ECONOMETRIE
AMSTERDAM

On RALLY, an evaluation based on criteria *

E. Spoor
E. Houweling
R. Geerts
R. Losekoot
M. Sprengers

Free University, Dept of Economics,
August 1987, Amsterdam.

ABSTRACT

The past few years have seen enormous attention directed towards databases and especially towards relational theory. The prominent coverage received by these subjects has until recently somewhat obscured the advances that have been achieved in a closely related field: that of Fourth-Generation-Language (4GL) Application-Development-Systems (ADS's). Has this database-oriented interest led to a gross undervaluation of the ADS phenomenon? No, on the contrary, it now is a hotly debated topic, but the point is: the discussions are rather chaotic. This is mainly due to the fact that a theoretical framework, comparable to that of e.g. relational databases is lacking. The absence of such an academic foundation means that there are almost no clear reference-standards against which the often spectacular claims of manufacturers for their ADS-s can be gauged. This article therefore attempts to investigate a specific ADS (VAX RALLY) [RALL86] within the context of a set of criteria. The research for this publication was carried out in the last quarter of 1986 and the first few months of 1987.

* A version of this article is submitted to DECUS U.S. CHAPTER SIGs NEWSLETTERS.

CHAPTER 1 INTRODUCTION

For many years the development of information-systems has appeared to be a very expensive and time-consuming affair. Many causes are at the root of this fact. In this respect the changing requirements of end-users, that have contributed to the drastically increased complexity of these systems, are of great importance. Besides this the significant shortage of information-specialists is a rather serious problem. Finally the speedy developments concerning hardware and the resulting conversion-perils can be mentioned. Until recently developments in the software-field could not keep up with these advancements.

Next to the production of new information-systems, another issue is relevant: that of application-maintenance. This appears to be of crucial importance, but at the same time it constitutes one of the greatest bottlenecks in many organisations. Responsible for this situation again are the above mentioned causes from the application-development phase. Besides this, other problems of a different nature are prominent, like badly documented programs and poorly structured systems. Particularly to obviate the forenamed difficulties, the so-called "Structured Design Techniques" were introduced during the seventies. These design-methods have brought some alleviation, but certainly have not become an unqualified success. The development and maintenance of software continue to be structural problems.

In recent years efforts have been made to change this situation fundamentally. Various manufacturers are contributing to this by launching "Application Development Systems" (ADS's). One of these ADS's is described and examined against a number of criteria in this article. For that purpose this text is arranged as follows: In Chapter 2 a short explanation of ADS's in general is given. Moreover this chapter includes a paragraph with a discussion of the evaluation-criteria that are used. Chapter 3 gives a system-description of the specifically researched ADS. Finally, Chapter 4, contains the evaluation and the conclusion.

CHAPTER 2 APPLICATION DEVELOPMENT SYSTEMS

2.1 What are application development systems ?

In general, a distinction is made between the third and the fourth generation within the current computer languages. That third generation consists of the well-known higher programming languages like Cobol, Fortran, Basic, etc. These languages present many advantages in relation to their predecessors, but they also have serious drawbacks, the worst of which is their procedural nature. This means that experts are required and that they need to convert all user-specifications into program-procedures. In other words "HOW" the software is to perform must be specified in detail, which is a tedious and technical process. The direct consequence of this usually is a very long and error-prone software-development stage. Moreover the extremely important application-maintenance appears to be a very labour-intensive affair, mainly because of the used languages.
In response to these last disadvantages, in recent years fourth-generation software has been introduced to the marketplace. This software pretends to be as non-procedural as possible. In contrast to the usual state of affairs concerning third-generation lanuages, now the declarative aspect is prominent. This means that the priority now rests with the declaration of the specifications ("WHAT" the software is to do, instead of "HOW" it has to perform), and thus no longer with the technical translation to the program-procedures. The implementation of this conversion now is taken care of automatically by the software.

In practice the term "fourth-generation software" encloses a broad spectrum of programs and tools, with mutually greatly varying feasibilities. Nevertheless some common elements can be distinguished. Particularly :

- DataBases and DataBase Management Systems (DBMS's);
- Program-generators: i.e. advanced software that generates all required procedural programs on the basis of mainly non-procedural specifications and uses the forementioned databases. Such generators should include "Prototyping" facilities, to enable fast development of rough program- frameworks;
- Query-languages: languages that allow the user to easily perform manipulations on data in a database. Of course authorization controls are essential in this respect;
- Screen-generators: tools which can be used for a clear presentation of data by way of a sophisticated lay-out;
- Report-writers: aids to generate hardcopy-printouts.

The confusion concerning the term "fourth-generation software" necessitates the introduction of a new concept: Application Development Systems (ADS's):

An ADS can be described as a system that contains all the above mentioned elements, in such a way that they form a consistent unity, which enables developers to produce applications in a relatively non-procedural manner.

2.2 ADS-Evaluation-criteria

The evaluating character of this article requires the summing up of a number of criteria, against which experiences with specific ADS's can be placed in the context of standards for ADS's in general. The criteria mentioned below are partly derived from Cardenas [CARD82] and in our opinion can serve as a test for the judgement of any arbitrary ADS:

1 Do the ADS-tools form a unity ?
(They should not overlap each other, for then a choice-problem comes into existence; nor should they show mutual gaps, because this causes a penalty in the form of application-limitations);

2 Is the ADS DataBase- and DataCommunication-oriented ?;

3 Are the tools independent of a specific DBMS ?
(That is: can they be connected to DBMS's of different origins ?);

4 Is the ADS interactively oriented ?
And does it in this respect contain:
- high quality data-manipulation facilities ?
- features that enable screen-formatting ?
- report-writing facilities ?;

5 Does the ADS possess language-elements (if possible non-procedural) to describe activities, that cannot be directly deduced from the in-/output-specifications ?;

6 Can an observable productivity-improvement be witnessed in relation to third generation languages ?
(We consider this an essential evaluation-item, because here the motivation for the development of fourth generation software lies: this software should clearly lead to a substantial shortening of the above mentioned application-development stage.);

7 Is there really a higher measure of flexibility as compared to third generation languages ?
(This criterium is closely related to the former one, because obviously the increase in productivity should be evident here too. However flexibility goes one step further: It requires that the application, generated by the ADS, has an easily changeable internal structure. For it is imaginable, that an ADS produces applications with extremely rigid internal build-ups. (Often the application consists of main-parts which can be changed or deleted, only after all of their sub-elements labouriously have been removed one after the other.) If this is the case, changes and/or additions naturally are difficult to realise. By this, application-maintenance is seriously hampered and little or nothing would have been gained in relation to the current practice with third generation languages concerning this extremely important aspect. Concluding: the flexibility-criterium mainly covers the maintenance-friendliness of the software, generated by the ADS.);

8 Does operating the ADS still require the specialized knowledge of a computer-expert, or is it indeed the case that end-users can generate complex applications on their own ?
(This is greatly enhanced if the ADS is as non-procedural as possible, but even then ease of use is still not guaranteed automatically.);

9 Is fast and good "Prototyping" possible ?
(That is: are there facilities within the ADS that enable the generation of a rough concept ?);

10 Does the ADS-software guarantee the integrity of the total application ?
(In this respect a validation-mechanism comes to mind, that (automatically) checks the definition of an application on integrity and gives clear messages about the result. Thus there should not be a tiresome debugging-process.).

CHAPTER 3 SYSTEM-DESCRIPTION

3.1 An overview of VAX RALLY

After the general description of ADS's and the evaluation-criteria that should be taken into consideration, now the specific ADS-product VAX RALLY of DEC, linked to a RDB/VMS database [RDB/85], can be described. For an introduction of this package, an article has been written by R.T. Bennett [BENN86]. It was published in a former edition of this magazine. In the subsequent text a summery of the most important characteristics of this ADS follows.

VAX RALLY is made up of two main systems:
* The Dialog, consisting of tools to generate an application;
* The Runtime System. This part takes care of the execution of the application. The Runtime System will not receive attention in this article.

Figure 1

The Dialog shows itself as a series of menus and screens that enable the user to create a set of linked objects, which in turn form the application. Some important objects are:
- Forms;
- Reports;
- Menus;
- Help-texts;
- Error-messages;
- Linkages to the database(s).

These objects are laid down in the form of data-structures, which RALLY stores in a special application-file (AFILE). The above-mentioned Runtime System processes the contents of this file during the execution of the application. The Dialog itself also consists of two parts:
* The Builder Tools;
* The Editing Environment.

The Builder Tools.

This is a cluster of 5 sub-tools, that can be used to start building an application. Those sub-tools are:
- The Database Builder.
  This creates and modifies RDB/VMS databases. These databases are located outside the AFILE, as can be seen from figure 1. (see sub-paragraph 3.2.1.);
- The Data-Source-Definition Builder.
  With this, links between the forms/reports from the application and the underlying database(s) can be made. (see 3.2.2.);
- The Form/Report Builder. This part creates the structure of the forms/reports and also builds the screen-layout. (more about that in 3.2.3.);
- The Menu Builder.
  With this tool, the main-menu for the application is constructed. (see sub-paragraph 3.2.4.);
- The Message Builder. Help-texts and error-messages, that can be connected to a form or a menu, are created here. (no separate sub-paragraph has been devoted to this).

The Editing Environment.

The Editing Environment enables the developer to refine and/or change application-objects, that were made with the Builder Tools. Moreover it is possible to create new application-objects (except a database). Besides this, links to e.g. Cobol or Fortran can be established. Finally, the Editing Environment allows the use of VAX RALLY ADL (a procedural language). All these facilities will be explained more thoroughly in paragraph 3.3.

3.2 Further description of the Builder Tools

3.2.1 The Database Builder

The RDB is a relational DBMS and the presentation of data therefore takes place in the form of tables, called relations. A RDB/VMS database can be defined in two ways, namely by means of the Relational Database Operator (RDO; this is no part of RALLY, so that it will not receive attention here) and by means of the VAX RALLY Database Builder.

With that Database Builder a database can be established in the following way:

1. The initial creation of the database.
   This takes place by specifying a database-name. If the naming is completed, two files are created in the home-directory. These files contain the data and allow the database to be queried;
2. The definition of global fields.
   Every field in a relation must be based on a global field. Such global fields allow the user to define a set of standard definitions for all the data in the database. This promotes simplicity and consistency. A global field can thus be viewed as a template, on which specific data-items are based;
3. The creation of relations in the database.
   RALLY takes a relation to be a collection of local fields. These local fields are named on the basis of global fields;
4. The definition of indexes in the database. An index is a data-structure that enables RALLY to find data in a relation and to fetch them, without the need to view all the records in that relation. This method saves time if data have to be accessed frequently and therefore it then is adviseable to create many indexes. However, if a lot of updates occur, it is sensible to use few indexes. In RALLY, the user can determine to what extent indexes should be used.

3.2.2 The Data-Source-Definition Builder

Often a database will not be involved in its totality by a manipulation. Therefore RALLY offers the possibility to redefine the database into new "sources", that only present the required parts of that particular database. This redefinition can be seen as the translation from the conceptual level to the external level [DATE86] and takes place in two steps:

1 The first step concerns the definition of the Data Sources. Such a Data Source establishes a briage between the RDB ana the still-to-be-made form/report. In that Data Source Definition (DSD) the relation(s), which can be usea, are registratea. Besiaes, here the possibility exists to fix aaditional restraints to that use. In a DSD relations can be couplea by way of a "join". However, this join has limitations: upaating, deleting ana inserting are not allowea. In other woras, the "joining" of aifferent relations in a DSD mainly serves report-writing. Furtheron in this text, a methoa (involving another kina of joins) will be shown that aoes allow all manipulations (including upaating, aeleting ana inserting);

2 The seconα step in the redefinition is describea in sub-paragraph 3.2.3 (Form/Report Builder).

Schematically the redefinition can be depicted as follows (Groups will be aiscussea hereafter):

Figure 2

3.2.3 The Form-/Report Builaer

This tool serves to build screens, with which data can be presented to the end-user. Moreover RALLY-objects can be generatea, that facilitate manipulations against the database. In this Form-/Report-Builaer the second step is maae in the reaefinition-proces. To that ena so callea "Data Source Groups" (usually abbreviated to "Groups") are createa within the forms/reports. These "group-aefinitions" aesignate the DSD's that are used by a particular form/report. (In this respect only a single DSD can be aefinea per group; however several groups can refer to the same DSD.) A DSD is independent of any singular group-aefinition.

The functions of the group lie in the formulation of more stringent requirements for the access to the database ana in the establishment of specific characteristics (like the manipulation-mode that is allowea). In other woras a group-aefinition is restrictive towards a DSD ana thus can never signify a widening of competencies. Groups can be "joinea" in a so callea hierarchical form/report. RALLY calls this a "Parent-chila relation". In this case it is possible to carry out all normal aatabase-manipulations (upaate, delete, insert), which are not allowea when DSD's are joinea (Therefore it is evident, that a form/report in which one or more of the groups make use of joinea DSD's, does not allow all these manipulations any more).

3.2.4 The Menu-Builaer

It is absolutely neccessary to build a Main-menu before an application can be run. The Menu-Builaer is only capable of aefining a single menu, which has a simple structure. The reason for this, is that menu-choices can only point to forms/reports. To create one or more submenus ana/or complex structures, the Editing Environment has to be usea.

In the Menu-Builder, the 'Initial Usage Mode' has to be specified for each form/report. This mode indicates which manipulations may be carried out. The options are:

- Browse/Update/Delete (BUD)
  This possibility allows the user to make inquiries and to add, delete or change data;
- Insert
  In case this is specified, the user may add data to the database;
- Query
  This option gives permission to make inquiries to the database;
- Print only
  The specification of this choice results in a hardcopy printout, instead of a data-presentation on the terminal-screen.

3.3 Further description of the Editing Environment

The main differences between the Builder Tools and the Editing Environment are formed by:

- the scope and complexity of the objects;
- the use of extensive default-choices in the Builder Tools;
- the splitting up of objects (whether or not made in the Builder) into subobjects, so that detailed changes can be made to the application.

The Editing Environment does not, like the Builder, consist of neatly demarcated and separately named tools. Only two mechanisms are mentioned as such there:

- the Image Editor, with which the lay-out of forms/reports, menus, legends, help- and error-messages can be edited;
- the Verifier.
  With this the validity of objects, created in the Dialog can be checked. Possible mistakes are mentioned in 'Integrity Reports', together with location of occurrence and likely cause. These tools are not discussed in detail in this article.

The Editing Environment and its sub-parts can be entered through menu-choices. The 'Defining Application Objects' option of the RALLY-Main-menu provides this entrance. Thereafter a choice can be made from the following submenus:

- defining Menus;
- defining Forms/Reports;
- defining Data Definitions & Data Related Information;
- defining Tasks;
- defining, maintaining and managing applications;
- procedural language (ADL).

Each of these submenus again forms the entrance to a broad variety of further options, which every time are united in tree-structures. It would lead to far to explain all of these possibilities in detail. Only the most significant options are dealt with in the undermentioned.

3.3.1 Defining (& editing) Menus

In the Builder Tools only a simple menu (the Main-menu) can be created, which points exclusively to forms/reports. In the Editing Environment the possibility exists to establish several (sub-)menus and, as a consequence, even complete menu-hierarchies can come into existence. Moreover, the seperate menus can now be equipped with complex structures, which enable them not only to point to forms/reports, but also to other items like menus, tasks, commands and ADL-procedures (as will be explained furtheron). Finally, there are several options that enable changing/improving the lay-out of menu-screens. (Editing is not possible in the Menu Builder.)

3.3.2 Defining (& editing) Forms/Reports

Forms/reports themselves again consist of a number of sub-objects. To mention the most important of these:
- Groups (4 types);
- Fields;
- Text-areas;
- Form/report call-packets.

The Defining Forms/Reports option gives entry to a number of techniques to create or change these sub-objects. As far as the groups are concerned, "Data Source Groups" have been treated (in 3.2.3). No further attention will be paid to the other groups and to fields and text-areas, because this would lead to far. Because they are essential for understanding some fundamental RALLY-concepts, form/report call-packets (also called form/report packets; call-packets or just packets) will now receive a short explanation:

In the Builder Tools standard forms are created and, to enable manipulations against the RDB, linked to the Main-menu. This coupling is established through these form/report packets, which are created implicitly by the Builder. In the Editing Environment these packets can be built explicitly. In a form/report packet, the following is specified:
- which form/report must be called;
- in what "initial usage mode" the form/report stands;
- the before- & after-actions, that are to be deployed (see 3.3.4).

Forms and reports are independent of form/report packets. This means that a form/report can be defined in several different packets (and thus in various modes and with different before- & after-actions). In other words, such a form/report is reuseable.

3.3.3 Defining (& editing) Data Definitions

With the Builder Tools, among other things, DSD's are defined. In the Editing Environment three types of DSD can be created and/or changed:
- the Base DSD;
- the View DSD;
- the Breakup DSD.

The Base-DSD: This one is conform to the (above described) DSD from the Builder and forms the connection between the relations and the application:

Figure 3

The View-DSD: As soon as a "parent-child-relation" (see the above-mentioned) within a hierarchical form/report is used, the Form/Report Builder implicitly (!) creates so called "View-DSD's (one for each child). This is necessary to link required Base-DSD's. In the Editing Environment it is possible to explicitly create View-DSD's. This changes the data-flow to the hierarchical form/report, because the new View-DSD couples different Base-DSD's to each other, than the old one. View-DSD's are built from Base-DSD's and therefore inherit many characteristics from them.

Figure 4

The Breakup-DSD: This third kind of DSD is also used to make hierarchical forms/reports, but each of these forms/reports is (in contrast to those based on View-DSD's) derived from a single Base-DSD and in addition are merely read-only. The Breakup-DSD's can only be created and changed in the Editing Environment. These DSD's too are derived from Base-DSD's and again inherit many characteristics.

Figure 5

3.3.4 Defining Tasks (Application Flow Control)

The management of the flow of an application requires the presence of a number of specific control-processes. In RALLY this control is realized through the execution of so called 'Actions', which are combined into one or more 'Tasks'. On top of that it is possible to switch (back and forth) between Tasks and Actions, through 'Commands'. All these activities are essential in order to properly run an application and therefore are defined within a framework, that is called 'Application Flow Control' (AFC). Within the Editing Environment it is possible to define complex forms of Flow-Control, which consist of many (sub-)objects. The most important components and their relationships will be explained in the following:

- Action: The invocation of certain RALLY-objects is seen as an action. Each action can call one of the following objects:
  . a menu;
  . a form/report packet;
  . an ADL-procedure (see 3.3.5);
  . a parameter-packet (no attention will be paid to this);
  . an external program-link (this too is not discussed);
  . an action-list (will not be attended to either).

- Task: An independent set of mutually related actions. The number of actions within such a set is not limited theoretically. Each task has its own 'Action stack' (see one of the next items). Several tasks can be active at the same time. The distinct tasks can be shown simultaneously and followed in separate windows on the screen. This enables the user to a switch between tasks with one of the commands (e.g. 'next- task'). With a task (among other things) the following can be accomplished:
  . the creation of additional 'Entry Points';
  . running some processes in batch;
  . linking an application-command to a task;
  . the creation of a window on the screen, in which the task-flow can be seen. The first two points will now be illuminated:

- Entry points: If an application comes into existence in the Builder Tools, automatically a single default-task will be generated. This 'Main-Task' serves as an entry (Entry Point) to the application and directly leads to the Main-menu. Only after a choice has been made from this menu, can the user start working. If however several tasks are created, it is possible to define a new Entry Point for each task. Now users to whom an Entry Point has been assigned, can start right away, without the requirement to them to pass through the Main-menu first. This division of the application into tasks and Entry Points also means that protection of parts of the application can be accomplished for the sake of specific end-users:

Figure 6

- Batch-processes: The creation of such a process can be desired in situations in which activities exist, that don't require direct user-input and thus don't have to take place interactively. To be processed batchwise, those activities must be defined as tasks and supplied with Entry Points. Two application-areas are:
  . the execution of a series of printcommands, called 'Print-only-reports' (for example standard periodical sales-reports);
  . the carrying out of Data Maintenance Operations, that demand a considerable amount of capacity (For this, ADL-procedures have to be written).

- Command: With commands various kinds of operations can be executed. One of these operations concerns the navigation among tasks and actions. These navigations can take place interactively or can already be defined during the design of the application. This implicates that these commands have great influence on the Flow Control.

- Action-stack (=execution-stack): A queue, which takes care of the management of a stream of actions that have to be executed when running an application. This queue operates according to the lifo (last in first out) principle. Each active task has its own Action-stack. When an action calls another action, the first action stays in the stack, but the new action is placed on top of it. If this second action is completed, it is removed from the stack and the Flow-Control resumes with the first action, which now is at the top of the stack. If the stack contains only a single action and when that action has been a executed, the task is completed.

- Invocation-type (= call-type; instead of 'type', 'style' is also used). This is the method, with which an action, task or command is activated, because filling the Action-stack happens through these invocation-types. Examples of these invocation-types are:
  . call: the present action stays in the queue; a new action is placed into the queue and executed;
  . execute: the present action is removed and a new action is placed into the queue and executed (to be sure, there are more types, but our investigation has been restricted to these).

- Action-site: A point in the application to call an action, a task or a command. This takes place by mentioning the name of the object and stating an invocation-type. Action-sites are among others:
  . the first action of a task;
  . a menu-choice;
  . a form/report call-packet;
  . a form/report;
  . an ADL-procedure.

- Before- & after-action: This is a RALLY-object, with which actions automatically can initiate other actions, without necessitating user-input (in the form of an explicit menu-choice).

3.3.5 Defining, maintaining and managing applications

This menu-choice leads to a miscellaneous set of 'Utilities' and other matters. Of these, the following can be mentioned:
- the creation of additional Entry Points (see the Task-description);
- the creation and changing of Help-, Error- and Legend- messages. This forms an addition to the Message Builder and is of subordinate importance to this description;
- security; this is an important issue and therefore it requires some further explanation:

It is possible to protect (parts of) an application against all unauthorized users. The protection can be accomplished in four different ways, but only the last two methods are part of RALLY itself (so called Dialog Security Mechanisms):
- on the file-level, through 'VMS file protection'. A feature, that controls entry to the AFILE and the RDB/VMS database files. This control is carried out by the VMS Operating System, which can assign different authorizations to different users;
- through 'RDB/VMS Security'. A mechanism, that takes care of specific relations in a database (through RDO);
- on the level of distinct RALLY-objects. This happens by way of 'RALLY Security Items', which (among other things) contain passwords. Each password is linked to a specific object, like a (Main-) menu, form/report packet, ADL-procedures, etc. Through such a security-item it is also possible to couple a password to an AFILE and thus protect an entire application against modification-attempts;
- through the creation of additional Entry Points (see the foregoing). Only users, who know the name of such an Entry Point have access to the associated task. Therefore those users only have permission to use a particular part of the total application.

3.3.6 Procedural Language (Application Development Language)

As part of VAX RALLY, the Application Development Language (ADL) can be used. This is a special language, which should be applied for the definition of necessary elements within the application and/or the creation of advanced supplements. Examples are:
- the development of field-validation procedures, which cannot be produced with the form/report validation method;
- the formulation of complex formulas to determine the value of 'computed fields';
- the execution of sophisticated data-manipulations, without direct user-input (see the task-description). Therefore ADL includes built-in DML-statements;
- the settlement of error-conditions.

ADL is the only procedural part of RALLY. DIGITAL (DEC) claims that this language is very easy to work with. Two reasons are its powerfull structuredness (in 'blocks') and its similarity with e.g. Pascal. As recorded, RALLY takes ADL-procedures to be actions and that is why they can be invocated from any arbitrary action-site.

3.3.7 Integration of defined RALLY-objects

To conclude this chapter a scheme will be shown, to explain how the various objects in an application fit together, and are related to each other through the Flow-Control:

Figure 7

From the figure it appears, that the task-level is the highest level. The start of an application or part of an application always begins with the invocation of a task. In this case, the first action of the first task is the invocation of the Main-menu. A Main-menu in turn can initiate several actions, varying from a menu-set to a new task. In this respect it applies that every menu-option is an action-site. In each one of these action-sites, invocation-types are used to call RALLY-objects (e.g. form/report packets). These invocation-activities can be monitored by means of the associated tasks and particularly their action-stacks.
Only the left side of the figure is worked out in detail. A Main-menu (in this example) consists of a number of sub-menus. Each sub-menu in turn can initiate several form/report packets. As said, in these packets the forms/reports, that are to be invoked, are mentioned. After the execution of these packets, another action (e.g. a print- command) can be carried out, without the requirement of any user-action. This then is accomplished through the after-action. The data of the database that have a bearing on a form/report are determined by the groups, which are defined in the form/report-definition. The group-definition relies on the Data-Source-Definition, which in turn contains one or more RDB-relations.

CHAPTER 4 EVALUATION AND ASSESSMENT

4.1 Practical experiences and criticisms

The following remarks are derived from our specific experiences and they encompass positive as well as negative elements:

- The totality of the Builder Tools is goal-oriented and transparent: In this context goal-oriented means, that the user can confine himself to the declaration of his wishes without having to convert all the details into a programming language. Transparent refers to the fact, that there is not a great stream of options, but always only a limited number. In order to know what everything means, a somewhat tedious learning-process needs to be absolved, but once this has been accomplished, it is rather easy to build a simple application with the Builder. The Editing Environment is much less clear, because of the enormous amount of options. All in all, RALLY comes across as massive and therefore relatively unclear to the uninitiated. This means that a long span of experience with this ADS is required, to handle it well.

- We question ourselves as whether there has to be a distinction between the Builder Tools and the Editing Environment, for in our opinion the Builder does not allow genuine "Prototyping". While there is only one menu; the objects can not be subdivided; etc., all this means that the possibilities for arriving at such a rough design are really too limited. Moreover, the Editing Environment has to be studied anyhow, to build a full-blown application. So, why not e.g. one single tool, that contains all the options ? Of course, this would neccessitate a thorough tackle of the transparency problem.

- ADL pretends to be a simple programming language, with statements, procedures, functions and syntax resembling that of Pascal. In other words it is procedural. If an application gets a little complex, extensive validation-procedures are necessary to guarantee the integrity-surveillance of the database. Then the use of ADL is inescapable. This means that some procedural programming turns up again. As a consequence specialistic know-how is still required and so many of the RALLY-advantages are neutralized again!

- The possibility to verify applications has saved much time and effort during the development of programs. However, the error-messages that were displayed when something went wrong were not always crisp and clear.

- The "Trace-log" command (in VAX/VMS) is extremely important. Without this option, we never could have succeeded in developing a properly working ADL-procedure. Besides, this command enabled the tracking down of the reasons why some other parts of the application initially did not live up to our expectations. For this trace-log option precisely denotes which form/reports are called, what ADL-procedures are run through, what happens to the database, etc.

- Nowhere can it be discovered, what the underlying base-language is that RALLY uses. This implicates that no thorough investigations are possible in the case of persistent errors and that "tuning" (by specialists) cannot be undertaken.

- The application in its totality can be well protected with, among other things, Entry-points and/or Passwords. In this way several end-users can use the same application, without them having to see the total or each others views.

- The integrity-protection of the singular objects within an application, by way of mutual "linking", is sometimes carried too far. Therefore, repairing mistakes and performing changes and additions is a tiresome process, that takes quite some time. This is because many mutual references in the application have to be unlinked first (they can remain in existence), before any part of that application can be altered or deleted.

- Guaranteeing the integrity of the database [CODD85]. First, the entity-integrity. In RALLY the use of a primary key is optional, for the user does not have to define this. Furthermore, the use of indexes with unique values is optional too. Alltogether, this integrity rule is thus not supported automatically. Secondly, the referential-integrity. This rule is not supported automatically either. To enforce it, relatively complex ADL-procedures must be written. Thirdly, the user-defined integrity. This is optional too. The specification of e.g. maximal values is not required. Concluding: Guaranteeing the integrity of the database is totally up to the user.

- Concerning user-friendliness, the following aspects can be distinguished (N.B. in our opinion it sometimes is necessary to discern between application-developers and end-users in the undermentioned):
  . The bulk of this ADS package has been experienced to be reasonably user-friendly, mainly because of the good menu-guidance;
  . We have found the extensive on-line help to be very handy, especially at the beginning of our research. Unfortunately this help information is not available when formulating ADL-procedures;
  . Notwithstanding the on-line help, manuals (in particular those concerning ADL) are still neccessary. In our opinion, the manual that deals with ADL is too restricted and gives little clarification;
  . During the development of an application, the developer is kept well informed by a "status-line" at the bottom of the screen;
  . If a certain item is required by an object of the application, this is clearly indicated;
  . The list-of-values (lov) features help to avoid unneccessary typing, especially as far as the developer is concerned. (A lov is a legenda, that automatically appears on the screen whenever a choice has to be made.);
  . The function-keys are very usefull for application developers, but in our opinion the same does not hold for end-users. Certainly to incidental users, the handling of the many function-keys can be quite frightening;
  . The system is fast as far as processing-speed is concerned and when it has to generate new applications;
  . The manipulation of the database can be accomplished in a relatively simple manner through a language that resembles QBE [ZLOO75].

4.2 Evaluation

Following now is a judgement, based on the criteria as described in Chapter 2:

1 Do the ADS-tools form a unity ?
Yes, they certainly do. There is a well-built structure, especially with regard to the Builder Tools. All those tools are neatly grouped into menus and can be invoked easily. But: If the Builder Tools are compared with the Editing Envirorment a marked duplication of tools emerges. With the exception of building and editing a database, exactly the same (and of course many additional) things can be done by the Editing Environment as by the Builder. We therefore consider this duplication to be unneccessary. (see point 9 and the conclusion too) We did not encounter essential mutual gaps between the tools.

2 Is the ADS DataBase- and DataCommunication-oriented ?
It clearly is DB-oriented. The DB even is indispenseable and the first thing that has to be created (apart from of an AFILE) in the Builder is the database. Whether RALLY is DC- oriented could not be ascertained by us, though it is quite likely because of the linkage-capabilities of the VMS Operating System (with among other things DB's on so called "remote nodes"), that are mentioned in the documentation. We only had a "stand-alone system" at our disposal, so we could not investigate if (and when so, to what extent) network- facilities can be implemented.

3 Are the tools independent of a specific DBMS ?
Probably not. We cannot make definite statements concerning this item, for we only could work with VAX/RDB. In all DIGITAL manuals just this one type of DB (and related DBMS) is mentioned and discussed.

4 Is the ADS interactively oriented and does it possess good data-manipulation features, screen-formatting and report-writing facillities ? The part of this question concerning interactive orientation can be answered affirmatively, though this is not the case as far as ADL is concerned. The data-manipulation facilities have been arranged in an excellent manner, by way of a QBE-like language in form/reports. With respect to report-writing: the generation of reports is simple through the "print-only-mode" specification in menus and/or packets. Finally, screen-formatting is excellent, through adjustments of the so called "default-layouts".

5 Does the ADS posses language-elements (if possible non-procedural) to describe activities, that cannot be directly deduced from the in-/output-specifications ?
Yes, RALLY does, in the form of ADL (mainly needed for the creation of validation-procedures). However, this language is highly procedural. (N.B.: the great majority of the activities that have to be included can be described in the in-/output-specifications of the non-ADL part of RALLY).

6 Can an observable productivity-improvement be witnessed in relation to third generation languages ?
Yes, for in a very short time a working new application can be built. However, we have not quantified the advantage.

7 Is there really a higher measure of flexibility as compared to third generation languages ?
Partly there is, because implementing improvements or additions is done faster and easier than with third generation languages. This too of course is a subjective judgement, that can be subjugated to further quantification. On the other hand, flexibility is still a long way from being optimal, due to the rigid internal structure, with very extensive mutual"linkings". This means that all sub-objects of a certain object have to be decoupled or destroyed, before any changes or additions can be made to the object itself and this in turn is certainly not maintenance-friendly.

8 Does the ADS still require specialistic knowledge ?
Yes it does, for early on in the design-process, procedural language-facilities have to be called in. Therefore an end-user probably will not be able to build complex applications on his own.

9 Is fast and good "Prototyping" possible ?
Indeed it is possible to create such a rough design in the Builder Tools in a short time. However, we consider the result to be too simplistic. For example, it is not possible to create more than one menu in the Builder and so if more (sub-)menus are needed it is neccessary to use the Editing Environment. The same procedure has to be followed for many other objects.

10 Does the ADS-software (automatically) guarantee the integrity of the (total) application ?
Yes, there is a "Verifier" that undertakes a check of every single RALLY-object, whenever the definition-phase of such an object is completed. Furthermore, it is possible to exercise control on the validity of the entire application through the "Verify application" command. Finally, a check is made at the beginning of any runtime-session.

4.3 Conclusion

Taking everything together, we can conclude that RALLY displays a rather positive image. It certainly is a great improvement in relation to Cobol and other third-generation languages.
Nevertheless, some points are still susceptible to improvement:

- It is adviseable to:
  - . either merge the Builder Tools and the Editing Environment, in order to eliminate duplication in tools,
  - . or either further develop the Builder in such a way, as to enable full-blown "Prototyping".
- The ADL is a stumbling block on the road to virtually complete non-procedurality. By paying much attention to this weak spot, the total picture could be improved considerably. In our opinion this can be accomplished by deminishing the role of ADL in RALLY and moreover by reducing the procedural nature of ADL. Deminishing the role of ADL could partly be attained by the inclusion of the creation of (advanced) validations in the non-procedural part of RALLY.
- Linkage-possibilities with other databases and DBMS's are mentioned nowhere. These facilities should be present and they should be described explicitly in the documentation.

- The internal structure of the generated applications should be less rigid, as far as the mutual "linkage" of the objects is concerned. Through the increased flexibility which would be established in that way, the maintenance-friendliness is bound to be greatly enhanced.

Literature

[BENN86]: Bennett, R. T.; An Introduction to RALLY, DECUS SIGs Newsletters, December 1986, Volume 2, Number 4.

[CARD82]: Cardenas, A. F. and W. P. Grafton; Challenges and Requirements for New Application Generators. AFIPS, Proceedings of the 1982 National Computer Conference, June 7-10, 1982, Houston, Texas.

[CODD85]: Codd, E. F.; "Is Your Relational Database Management System Really Relational? An evaluation scheme.", Computerworld, October 1985.

[DATE86]: Date, C. J.; Introduction to Database Systems, Fourth Edition, Addison-Wesley, 1986.

[RALL86]: Introduction to VAX RALLY, April 1986, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX RALLY Dialog User's Guide, April 1986, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX RALLY Dialog Reference Manual, April 1986, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX RALLY Command Reference Manual, April 1986, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX RALLY ADL User's Guide, April 1986, Digital Equipment Corporation, Maynard, Massachusetts.

[RDB/85]: VAX Rdb/VMS Guide to Database Design and Definition, December 1985, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX Rdb/VMS Guide to Database Administration and Maintenance, December 1985, Digital Equipment Corporation, Maynard, Massachusetts.

: VAX Rdb/VMS Reference Manual, December 1985, Digital Equipment Corporation, Maynard, Massachusetts.

[ZLOO75]: Zloof, M. M.; Query By Example, Proc. NCC 44, May 1975.

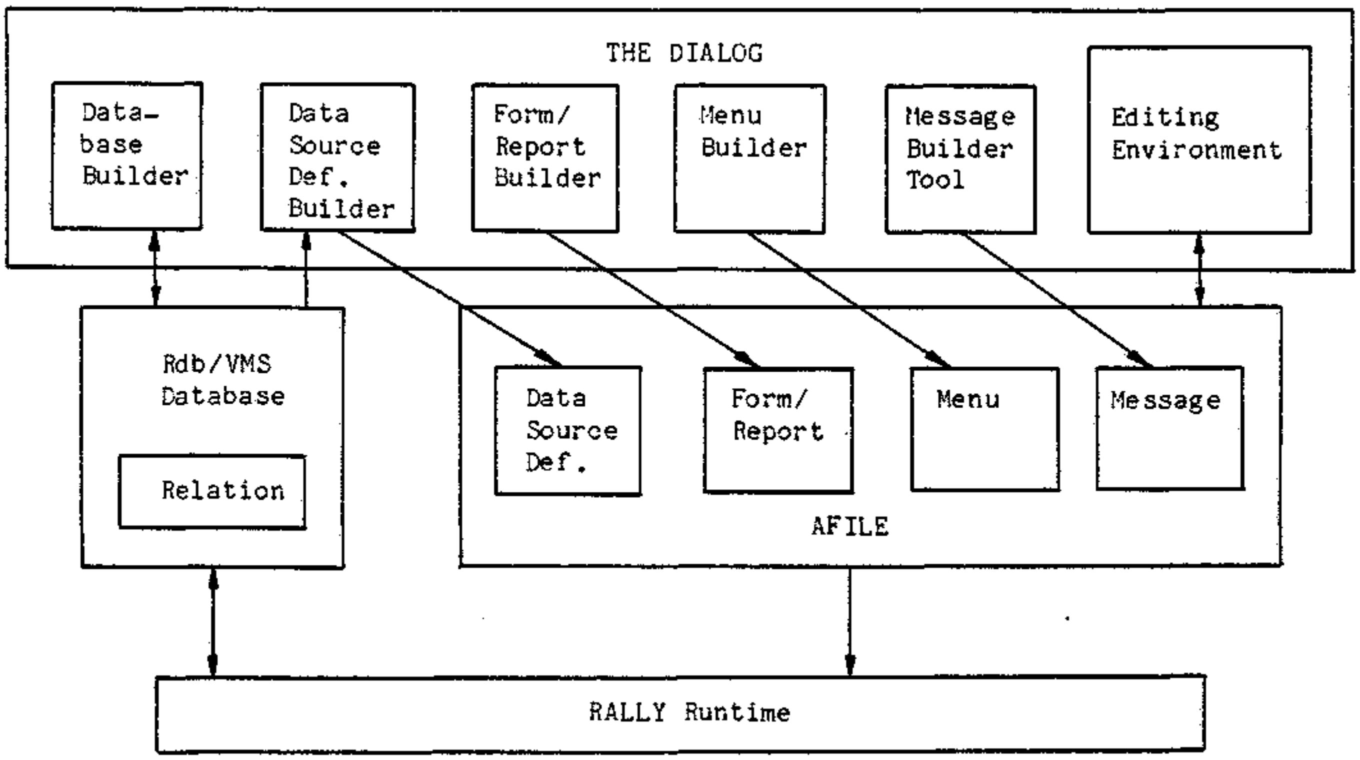
THE DIALOG
Data-base Builder
Data Source Def. Builder
Form/ Report Builder
Menu Builder
Message Builder Tool
Editing Environment
Rdb/VMS Database
Relation
Data Source Def.
Form/ Report
Menu
Message
AFILE
RALLY Runtime

Figure 1

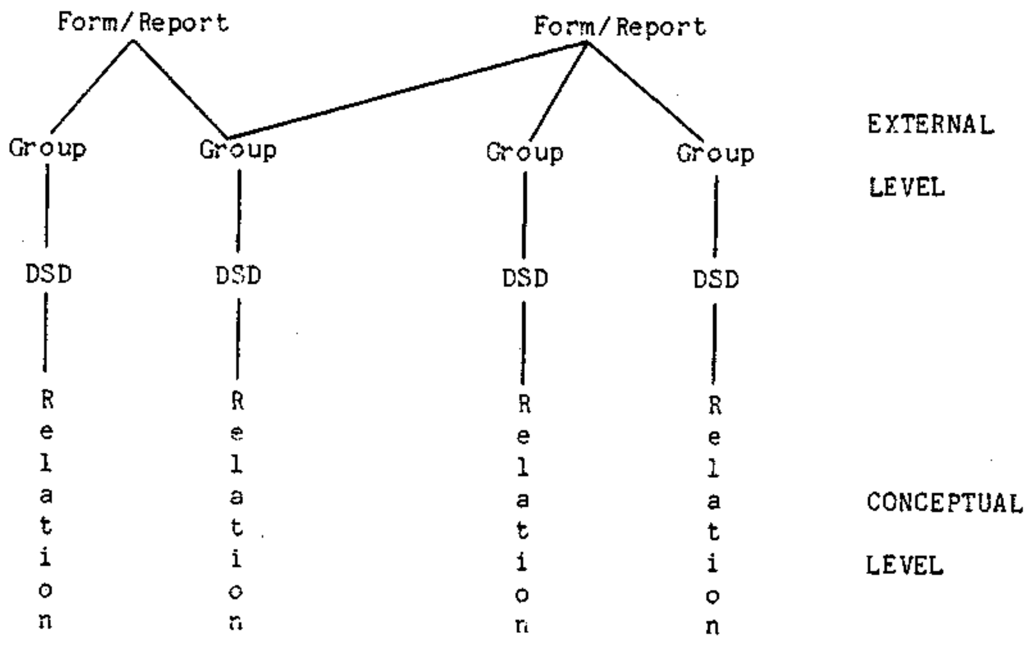
Form/Report
Form/Report
Group
Group
Group
Group
DSD
DSD
DSD
DSD
Relation
Relation
Relation
Relation
EXTERNAL LEVEL
CONCEPTUAL LEVEL

Figure 2

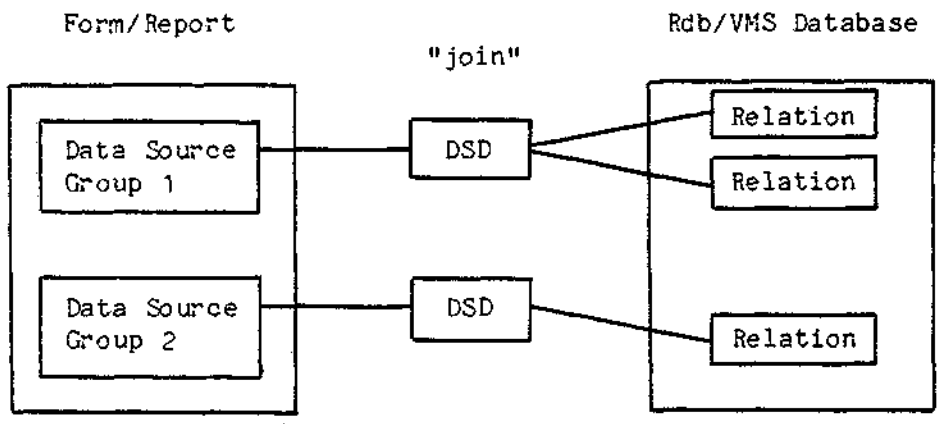
Form/Report
"join"
Rdb/VMS Database
Data Source Group 1
DSD
Relation
Relation
Data Source Group 2
DSD
Relation

Figure 3

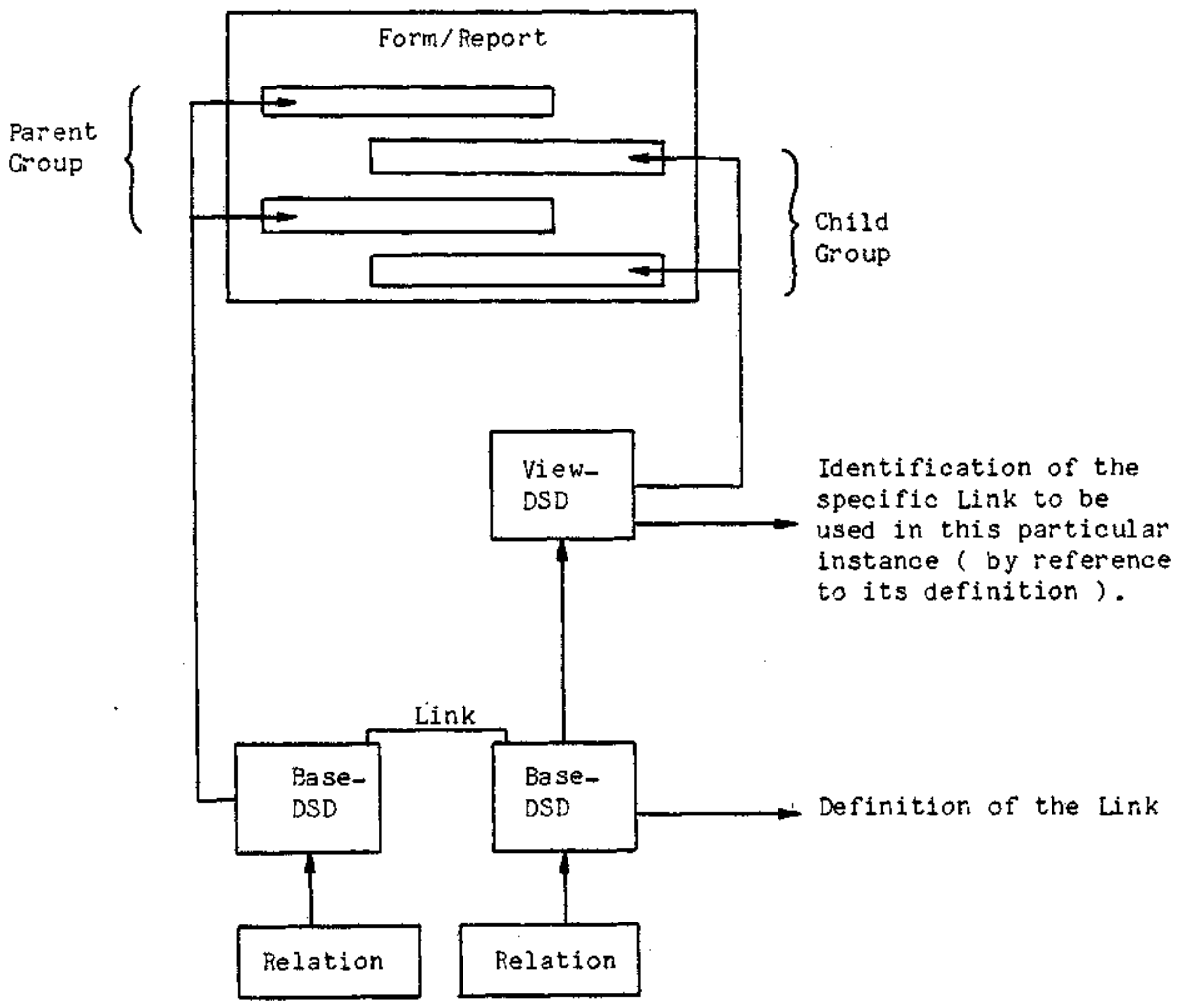
Form/Report
Parent Group
Child Group
View-DSD
Identification of the specific Link to be used in this particular instance (by reference to its definition).
Link
Base-DSD
Base-DSD
Definition of the Link
Relation
Relation

Figure 4

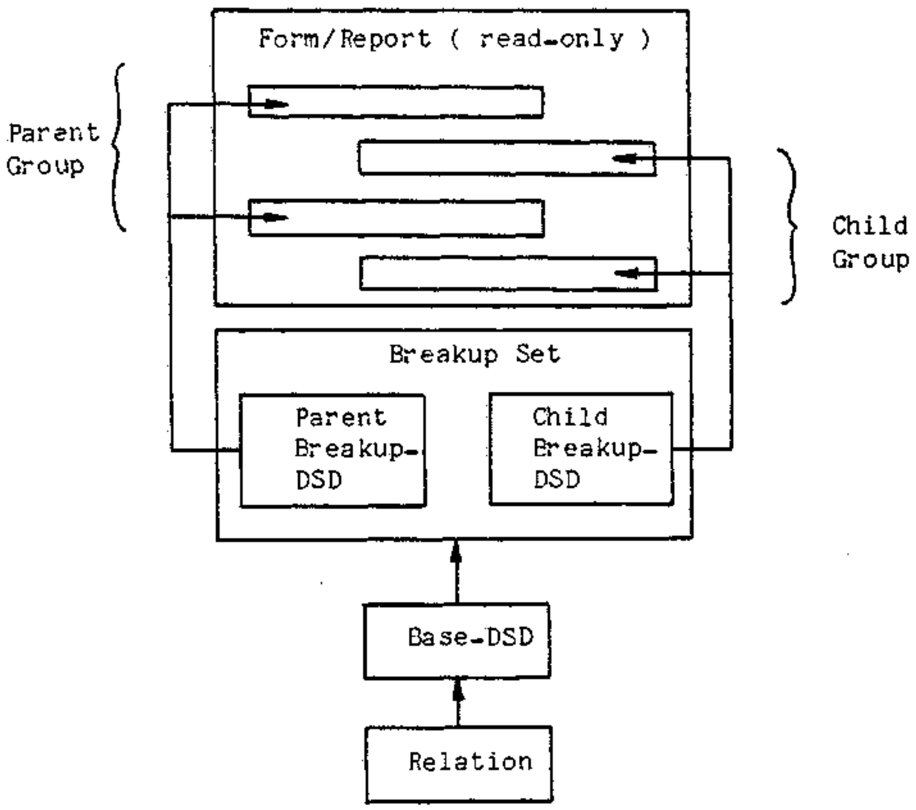
Form/Report (read-only)
Parent Group
Child Group
Breakup Set
Parent Breakup-DSD
Child Breakup-DSD
Base-DSD
Relation

Figure 5

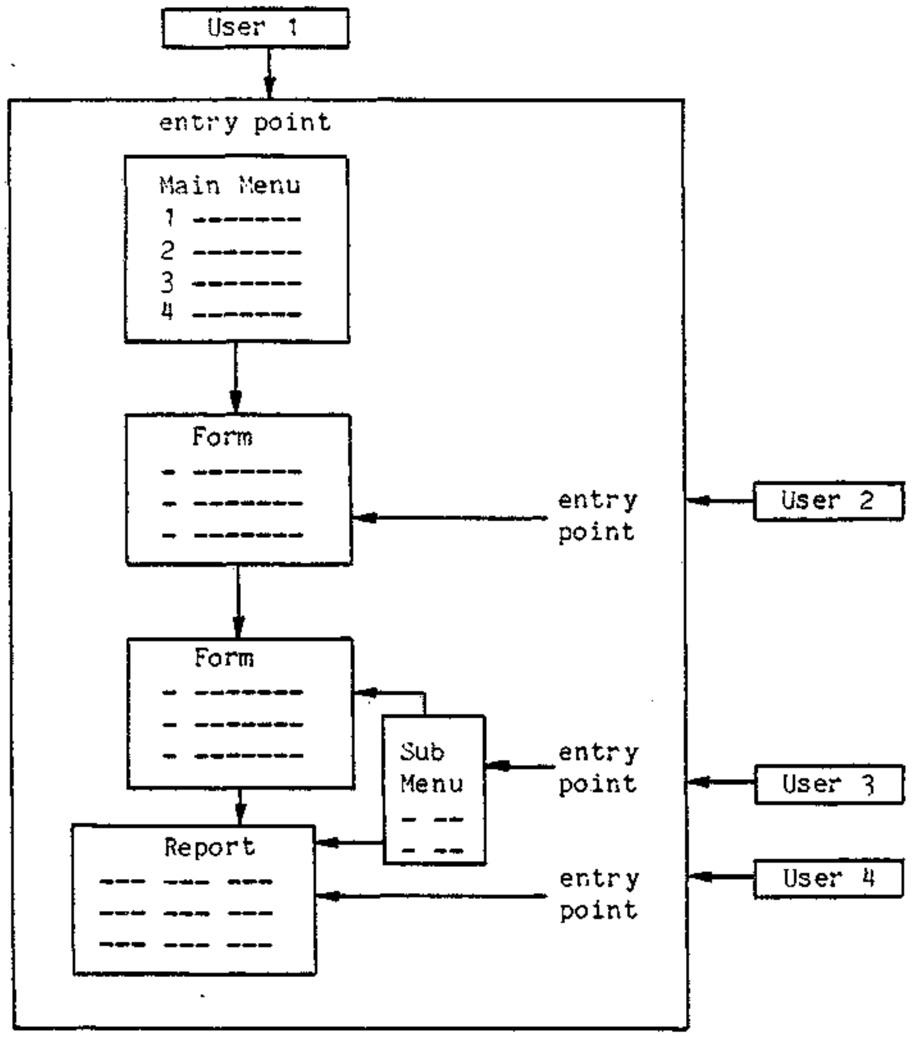
User 1
entry point
Main Menu
1
2
3
4
Form
entry point
User 2
Form
Sub Menu
entry point
User 3
Report
entry point
User 4

Figure 6

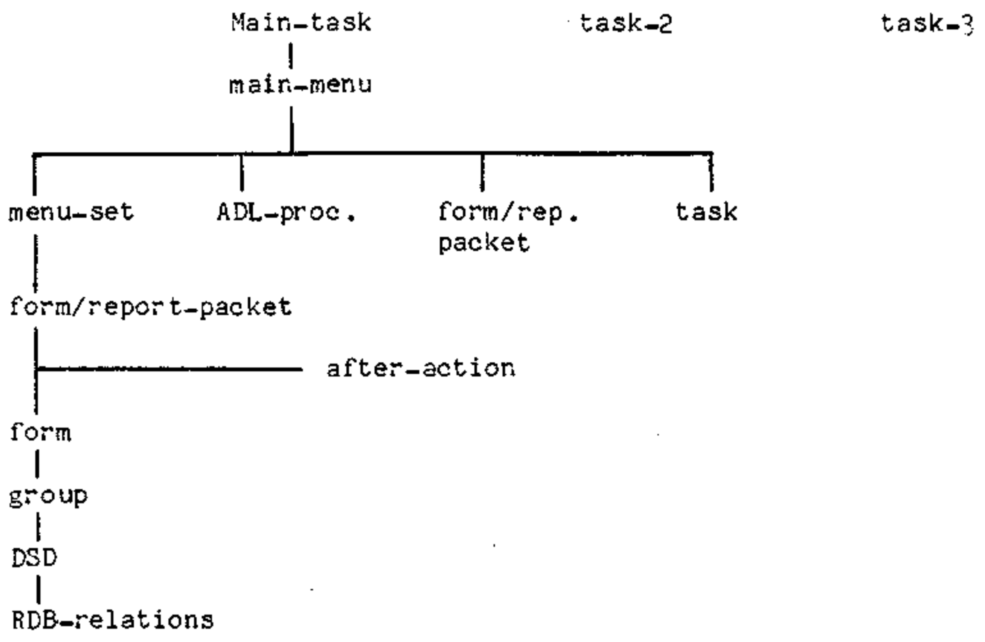
Main-task
task-2
task-3
main-menu
menu-set
ADL-proc.
form/rep. packet
task
form/report-packet
after-action
form
group
DSD
RDB-relations

Figure 7

1981-1	E. Vogelvang	A quarterly econometrie model for the Price Formation of Coffee on the World Market
1981-2	H.P. Smit	Demand and Supply of Natural Rubber, Part I
1981-3	R. Vos	The political Economy of the Republic of Korea; A proposal for a model framework of an open economy in the ESCAP-region, with emphasis on the Role of the State
1981-4	F.C. Palm	Structural Econometric Modeling and Time Series Analysis - Towards an Integrated Approach
1981-5	P. Nijkamp in co-op. with H. v. Handenhoven and R. Janssen	Urban Impact Analysis in a Spatial Context: Methodologie and Case Study
1981-6	R. Ruben	Primaire exporten en ekonomiese ontwikkeling
1981-7	D.A. Kodde	Het genereren en evalueren van voorspellingen van omzet en netto winst: een toegepast kwantitatieve benadering
1981-8	B. Out	Financiële vraagstukken onder onzekerheid
1981-9	P. van Dijck and H. Verbruggen	A Constant-Market-Shares Analysis of ASEAN Manufactured Exports to the European Community
1981-10	P. Nijkamp, H. de Graaff and E. Sigar	A Multidimensional Analysis of Regional Infrastructure and Economic Development
1981-11	P. Nijkamp	International Conflict Analysis
1981-12	A.J. Mathot	L'Utilisation du Crédit lors de l'Achat d'une Voiture
1981-13	S.P. van Duin en P.A. Cornelis	Onderzoek naar levensomstandigheden en opvattingen over arbeid bij mensen zonder werk, deel I
1981-14	W. van Lierop and P. Nijkamp	Disaggregate Models of Choise in a Spatial Context
1981-15	Hidde P. Smit	The World Vehicle Market
1981-16	F.C. Palm	Structural Econometric Modeling and Time Series Analysis: An Integrated Approach
1981-17	F.C. Palm and Th.E. Nijman	Linear Regression Using Both Temporally Aggregated and Temporally Disaggregated Data
1981-18	F.C. Palm and J.M. Sneek	Some econometric Applications of the exact Distribution of the ratio of Two Quadratic Forms in Normal Variates
1981-19	P. Nijkamp and P. Rietveld	Soft Econometrics as a Tool for Regional Discrepancy Analysis
1981-20	H. Blommestein and P. Nijkamp	Soft Spatial Econometric Causality Models
1981-21	P. Nijkamp and P. Rietveld	Ordinal Econometrics in Regional and Urban Modeling
1981-22	F. Brouwer and P. Nijkamp	Categorical Spatial Data Analysis
1981-23	A. Kleinknecht	Prosperity, Crises and Innovation Patterns: Some more Observations on neo-Schumpeterian Hypotheses
1981-24	Hidde P. Smit	World Tire and Rubber Demand

1982-1	Peter Nijkamp	Long waves or catastrophes in Regional Development
1982-2	J.M. Sneek	Some Approximations to the Exact Distribution of Sample Autocorrelations for Autoregressive-moving Average Models
1982-3	F.E. Schippers	Empirisme en empirische toetsing in de wetenschapsfilosofie en in de ekonomische wetenschap
1982-4	Piet van Helsdingen maart 1982	Mantelprojekt 'Management & Politiek': Produktiebeleid en overheid; een onderzoek naar de invloed van de overheid op het produktbeleid in de verpakkingsmiddelenindustrie
1982-5	Peter Nijkamp Jaap Spronk	Integrated Policy Analysis by means of Interactive Learning Models
1982-6	Ruerd Ruben (ed.)	The Transition Strategy of Nicaragua
1982-7	H.W.M. Jansen mei 1982	Een alternatieve modellering van het gedrag van een besluitvormer: 'satisficing' i.p.v. 'maximizing'
1982-8 mei	J. Klaassen and H. Schreuder	Confidential revenue and Profit Forecasts by management and financial analysts: some first results
1982-9	F. Brouwer and P. Nijkamp	Multiple Rank Correlation Analysis