PRACTICAL METHODS AND TOOLS FOR SPECIFICATION

J. Ludewig
ETH Zürich, Institut für Informatik
CH 8092 Zürich

Contents

Note: Hans Matheis has used an earlier version of this paper for preparing a paper on languages for real-time software specification. Some of his extensions have been integrated here.

Our descriptions of specification systems are based on the material available to us. This information may not be complete, or up to date. Therefore, we are sorry in case some features are not reported correctly. Please refer to the material available from the suppliers (see 6.2).

0. Introduction

This is a course on Specification. Since it is based on experiences in the field of Software Engineering, it applies primarily to Software Specifications. Many observations and reports indicate, however, that, from specification aspects, there is not much difference between information processing systems in general and software in particular. Therefore, most of this course applies also to System Specification.

In the first chapter, some fundamentals are discussed. These include the life cycle model and the distribution of costs over the various activities, some definitions, and a rationale for semi-formal specification. The second chapter provides a general outline of a specification system, whose desirable properties are deduced from the qualities of good specifications. In the third chapter, we present some typical specification systems. The primary goal is to show some typical features of such systems rather than to describe them in detail. The fourth chapter addresses management aspects. In chapter 5, some general conclusions are drawn. The appendix (chapter 6) contains a bibliography on specification, and a list of suppliers.

1. Fundamentals

1.1 Life Cycle Model

Only very small systems can be built in the same way as primitive peoples build houses. As soon as the system is slightly complex, a systematic approach is necessary. The sequence of steps to be taken from the first idea to operation and further on until the system is discarded, is called the System Life Cycle. Though there are many different life cycle models, they are all based on the distinction between certain activities or phases, namely

analysis and specification
design
implementation
integration
operation and maintenance.

Note that the life cycle may be used as a phase model, or a model of activities, or a list of roles. In the sequel, the second meaning is assumed.

Recently, the life cycle concept has been attacked by several authors, not only because it does not reflect the experiences of many projects, but also because alternative ways of building systems (for instance by prototyping) are ignored. See the references in 6.1.6.

1.2 Cost Distribution

About two thirds of the total cost of software are caused by activities which take place when the software is already operational (i.e. during maintenance) (Boehm, 1976). Therefore, every attempt to reduce the high cost of software has to focus on maintenance.
(Note that there is an important difference between maintenance of hardware and of software: while hardware is actually *maintained*, i.e. the original state is conserved or restored, software is corrected, extended, or adapted to new requirements, i.e. it is *modified*. A program is different from its original state after maintenance.)

There are three ways of reducing the need for maintence:
- reduce need for correction
- reduce effort for modification
- reduce total volume (by using standard components or old software)

A good specification contributes to each of these subgoals. Therefore, the overall goal is not to reduce the effort for specification, but rather to invest more in specification in order to save much more during maintenance (and also during design and implementation).

1.3 Terminology

1.3.1 Specification

To date, we have not achieved a stable and well recognized terminology in Software Engineering. In the sequel, we use a simple, pragmatic definition of "specification" (from Kramer et al., 1982):

A description of an object stating its properties of interest. It usually implies that the description should try to be precise, testable, and formal.
It is recommended that "specification" be used with some attribute, e.g. requirement specification.
Specification is frequently used to mean functional specification which contains both requirements and design aspects. This form of use is imprecise.

Many more relevant terms are defined by the IEEE (1983).

Specifications are written and read by many people, like analysts, customers, managers, and programmers. Since these people differ greatly in their background, education, and interest, they have usually not the same idea of what a specification should look like. Tools, which can change the representation of a given information automatically, can help to meet the requirements of more than just one single group.

1.3.2 The System Triangle

When we talk about programming systems, or specification systems, we distinguish three components, or sets of components, namely methods, languages, and tools.

Methods indicate how to proceed, like recipes in a cookbook. Languages restrict the set of possible statements to a particular universe of discourse, and to a certain syntactical representation. Tools check, store, and transform such statements.

All three are strongly interrelated by the abstract concepts of the (specification-) system. Note that the term "methodology" means "science of methods", though it is often misused for "method". Figure 1 exemplifies the system triangle:

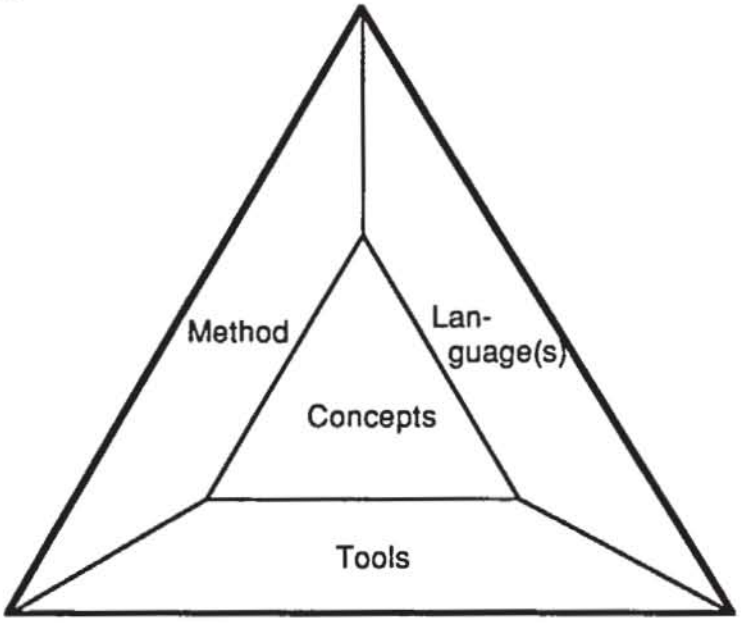

Figure 1: System triangle

1.3.3 Levels of Formality

There are languages of various formality. For our purposes, we distinguish four levels of formality, or styles:

Style	Syntax	Semantics	Examples
informal	not (prec.) defined	not (prec.) defined	natur. languages
formatted	restricted	not (prec.) defined	forms
semi-formal	defined	partially defined	pseudo-code
formal	defined	defined	progr. languages

For coding programs, we use a formal language. (Though the semantics of most programming languages are not precisely defined, if at all, there is always a translator which provides a de-facto-definition.) All other documents are written in informal language, sometimes on forms. Forms impose certain restrictions to the way natural language is used, and require the user to answer all relevant questions. Semi-formal languages are comparatively new; their first application was in program design languages (pseudo code).

1.4 Why semi-formal Specification ?

This paper does not treat formal specification. This does not mean that formal techniques are not important. However, they are not yet in a state that users in industry could really apply them. Semi-formal specification, i.e. an approach which is based on semi-formal specification languages, has (at least for the time being) several advantages:

- The languages can be learned and understood with limited effort by people who did not have extensive training in formal methods
- Documents resemble those written in natural language
- Incomplete and vague information fits better in such a system

On the other hand, semi-formal specification systems are superior to traditional informal specifications because

- many deficiencies which would be buried in plain text become visible
- it can be stored in, and retrieved form, a data base
- automatic tools can be used for checking and for changing the representation.

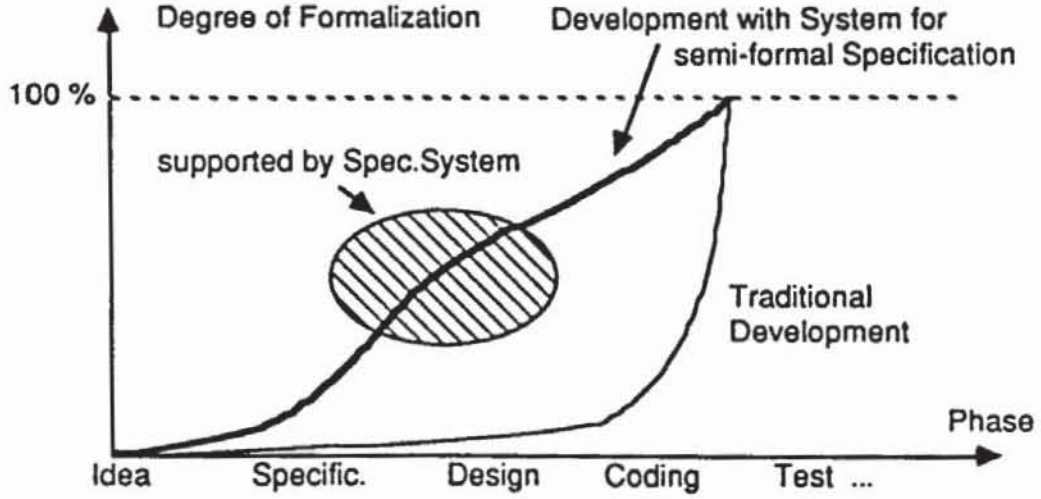

Figure 2: Degree of Formalization during the Software Life Cycle

Figure 2 shows schematically how the software development process is influenced by a system for semi-formal specification. In the traditional approach, there is practically no formalized information until the software is coded. Then, full formalization must be achieved in a single step. This method is, as we all know, error prone, because there are many misunderstandings, inconsistencies, simple errors and other shortages in the specs which are not discovered, because the document produced next, i.e. the code, can only be understood at the level of single instructions. In the modern approach, there are much better chances for detecting deficiencies of the specs, and improving them.

To summarize the message of this paragraph: Specification systems do not shorten the specification phase, but improve the quality of the resulting document.

2. Principles of Specification

2.1 Qualities of Specifications

A specification should be

- correct (i.e. it should reflect the actual requirements)
- complete (i.e. it should comprise all the relevant requirements)
- consistent
- unambiguous
- protected against loss of information and unintended changes
- easily writeable and modifyable
- readable and concise (in order to ease the communication between user and analyst)
- implementable (i.e. it should ease design and implementation)
- verifyable (i.e. there should exist a procedure to check whether or not the product complies with its specs). This quality is also called "testable".
- validateable (i.e. there should be a mechanism to ensure that the specification really reflects the user's specification)
- traceable (i.e. when the specification is changed, it should be easy to identify all statements in other documents affected by that change).

Note that these goals are highly inconsistent. For instance, a formal (e.g. algebraic) specification is verifyable, but not readable for most people, in particular not for the customer. Therefore, it is not validateable.

The first four of the qualities listed above (correctness, completeness, consistency, and unambiguity) do not have the same meaning to all people: vendors of tools for specification, for instance, often claim that their system can guarantee correctness. This does, of course, not imply that the *content* of the specs is correct with respect to the intentions of the customer, but only that certain formal requirements are met. The reason for this is that there is no reference (except the user's brain) to prove specifications correct or complete, in contrast to programs being provably correct with respect to the underlying specification.

2.2 Useful Properties of Specifications

In order to achieve the qualities listed above, certain properties are obviously useful:

- The specifications must be recorded on some permanent medium (e.g. paper, magnetic tape).

- They should be as formal as possible, and as informal as necessary. Also, they should support the processing of information which is vague, incomplete, or not yet well defined (i.e. providing a fill-in that indicates the lack of information).

- Specs should exist only in one single copy ("single source concept").

- There should be tools for automatic checks and transformations between different representations.

- Specs must be available in representations appropriate for those who have to use them (e.g. graphical representations which naturally mirror human's way of thinking).

2.3 Specification Systems Requirements

From the useful properties stated above, we can derive the requirements of specification systems; such a system should provide

- a data base system as the central information repository,

- a semi-formal specification language and several representations, including a graphical one,

- tools for all clerical tasks (storing, retrieval, checking, transformation).

Since software systems are developed by several people, and usually exist in several versions and variants at the same time, the specification system should also provide

- multi user operation of tools,

- automatic management of versions and variants.

2.4 General Structure of a Specification System

As mentioned above, an ideal specification system consists of a method, a language, and a set of tools, which are all based on a common set of concepts. The list following below summarizes the most desirable features.

Abstract concepts
- Life cycle model
- Stepwise completion
- Permanent validation

Methods supported by the system
- Enter every information immediately
- Allow for informal texts
- Check early for correctness, completeness, consistency, unambiguity
- Concentrate on information necessary for specification.

Languages
- Semi-formal specification language
- Several syntactical representations of a specification (e.g. graphics, tables etc.).

Tools
- Multi-user data base system
- Tools for checking, retrieval and selection.

In reality, however, most systems are incomplete. They are usually based on either of the components, and do never cover the full scope. Some activities started from a particular method (e.g. Structured Analysis, see 3.2), or from a certain representation (e.g. SADT, see 3.1), or from a set of tools (e.g. PRADOS, see 3.6). In the next chapter, some specification systems are presented. Our goal is to give an idea of their dominant feature; we certainly do not attempt to provide complete information. Please refer to the references (6.1), or contact the vendors listed in 6.2.

3. Specification Systems: Some Examples

In this chapter, we present some examples of specifications in various languages. Additionally, we briefly describe their underlying methods. The purpose is to show some typical styles rather than to describe systems in detail. These are the examples chosen for this paper:

SADT is probably the best known graphical language for expressing specifications;
Structured Analysis (SA) is similar to SADT, but has a wider range (towards design). We present it together with **proMod**, a tool which supports SA.
PSL/PSA is the classical tool-based specification system.
SREM is a very powerful system for describing, and simulating, real time software.
EPOS, another tool dedicated to the development of real time systems, is fairly successful in Germany and central Europe.
PRADOS was chosen as an example of those systems which do not really form a monolithic specification system, but rather a tool kit.

Our list covers only a part of those systems that we know, which in turn are certainly only a small fraction of those which exist. Therefore, our choice should not be interpreted as a judgement, or recommendation !

3.1 SADT (Structured Analysis and Design Technique)

SADT was developed by SofTech between 1972 and 1975. It covers the requirements analysis, the design and the documentation of specifications, aiming at improved communication between analysts, developers, and users.

3.1.1 The Method

The method SADT focuses on data flow and implies a stepwise refinement of so called SADT-diagrams which are hierarchically ordered. In its original definition (Ross, 1977), there is a duality between so called actigrams and datagrams modelling the data flow in two different ways representing different views of the system:

- actigrams identify functions as central elements of the description and data providing e.g. input or output for the functions

- datagrams identify data as central elements of the description and functions providing e.g. input or output for the data.

The redundancy makes it possible to prove consistency, i.e. one can check whether every function in an actigram is comprised in some datagram, and vice versa.

3.1.2 The Language

SADT is a graphical specification language allowing the user to describe the system in terms of activities and data. As outlined above, on the one hand there are actigrams consisting of activities and data. Activities are represented by boxes and data by arrows. On the other hand there are datagrams, where boxes stand for data, while arrows represent activities. Practical experience, however, indicates that most users tend to use only actigrams. For the reason of complexity the language restricts the number of boxes per SADT-diagram to seven.

Figure 3 shows an SADT-box with its typical components:

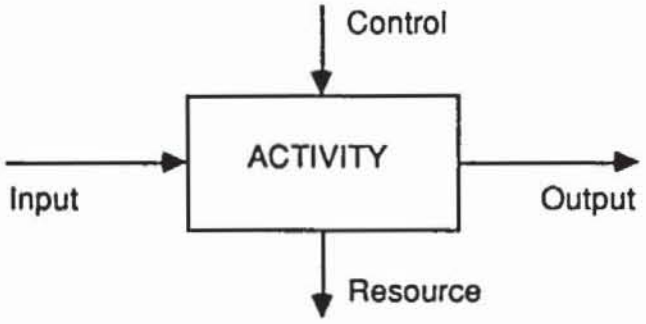

Figure 3: SADT-box (Actigram)

The three actigrams on the following pages (figures 4, 5, 6) show an activity ("ASSIST SADT USERS") at three different levels of refinement. Note that the last actigram (fig. 6) refines an activity ("CREATE KITS") of the second diagram (fig. 5). (Source: Lissandre et al., 1984, from IGL, Paris)

3.1.3 Tools

SADT is still a paper and pencil method. And there is no problem in draawing all the diagrams once. However, when there are changes (and the need for change is the only property of software that does never change), diagrams must be redrawn again and again. This is very annoying. Therefore, there have been several activities for providing tool support; the examples in figures 4, 5, 6 were produced by such a tool. Their capabilities range from simple graphics (i.e. they are used as an automatic drawing machine) to fairly sophisticated programs which do some sematic checking and analysis. According to D.T. Ross, who invented SADT, "none (of the tools) is fully successful in implementing SADT" (Ross, 1985 b).

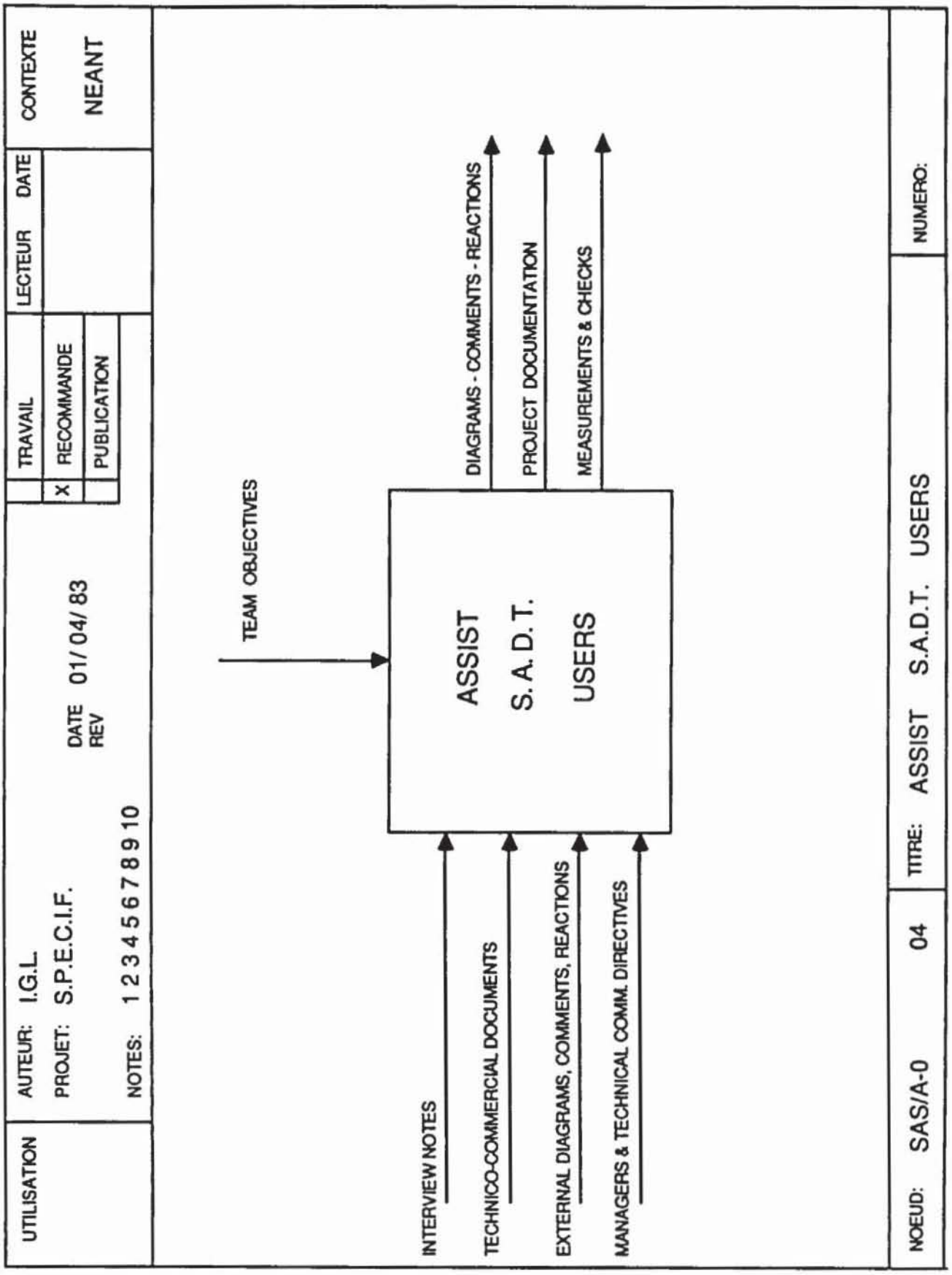

Figure 4: SADT-Diagram, top-level

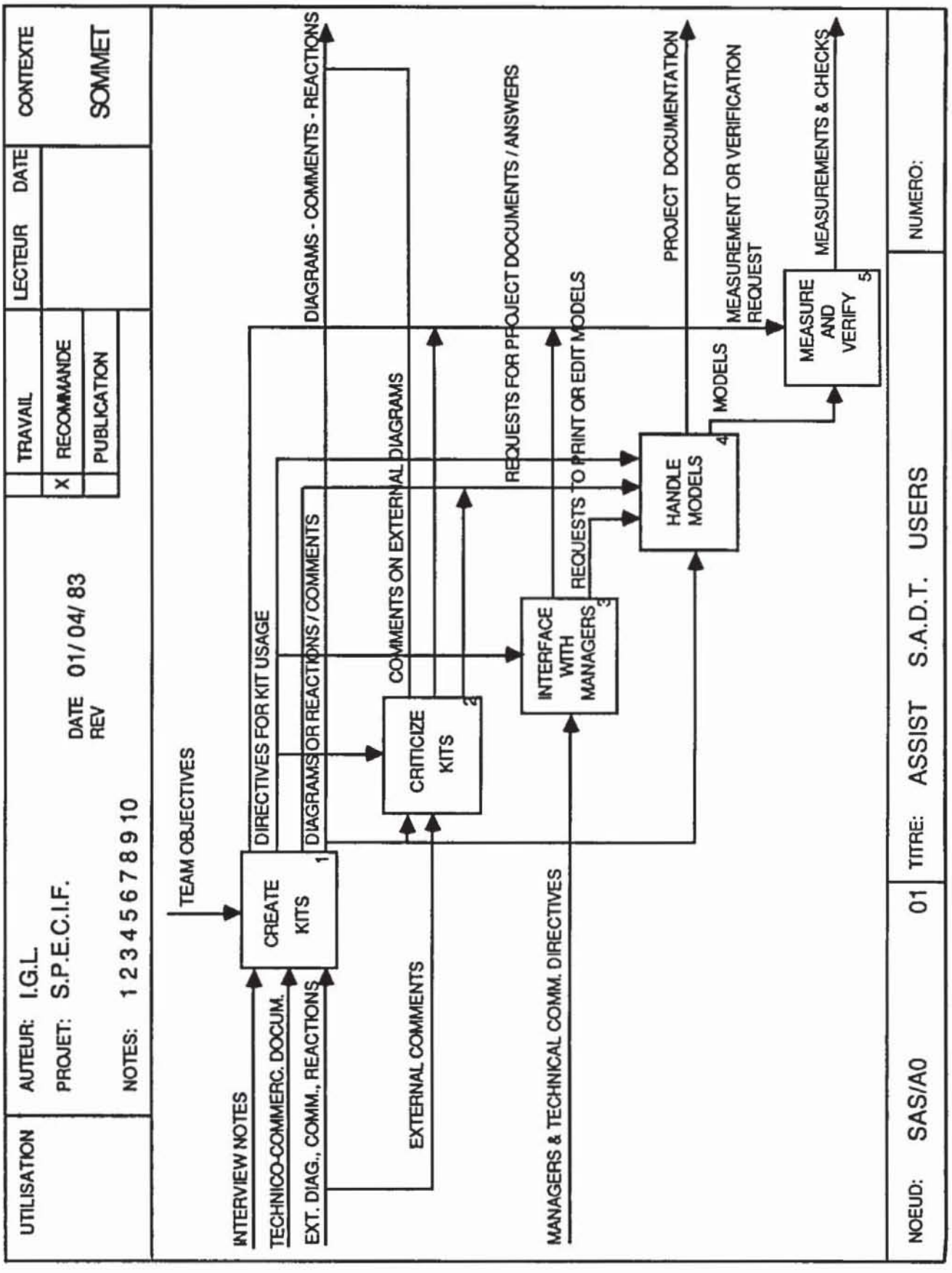

Figure 5: SADT-Diagram, first level below top

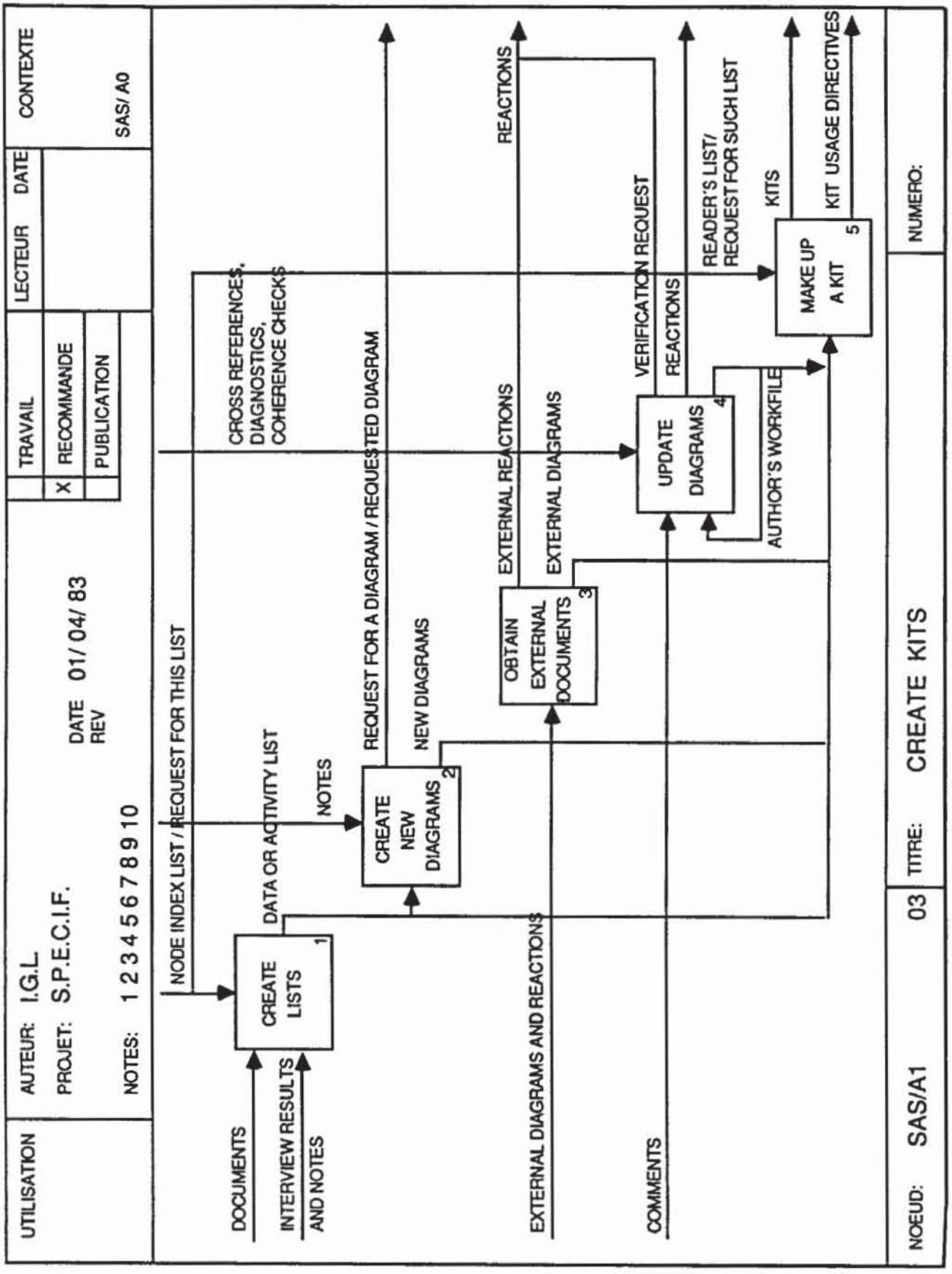

Figure 6: SADT-Diagram, second level

3.2 Structured Analysis (SA)

SA was developed by Yourdon and others (see Yourdan, Constantine, 1979). Although, the name is very similar to SADT, only the data flow as the central principle is common to both. It is used for analysis and both coarse and detailed design.

3.2.1 The Method

The method allows the user to model a system with data-flow diagrams (DFDs) consisting of data, and processes transforming the data. In other words, DFDs describe the flow of data through the system by denoting sources and sinks for data flows, the data flows itself, and processes. So called minispecs are used to describe processes in more detail. For refining the structure of data, a data dictionary (DD) is used. SA proposes a stepwise decomposition of DFDs so that each process in the parent DFD is broken down into several child DFDs. Consequently, several levels of DFDs emerge.

SA proposes two major steps. The first one is to develop a so called context diagram, which shows how the system is connected to its environment. Hereby, the user defines the interface in terms of sources and sinks of the environment, processes, data flows, and files. Note that the data flow consists of both the data and the direction of flow.

In the second step, the user partitions and refines the system "as long as possible", i.e. each process of a DFD is described in more and more detail until the level of atomic processes is reached. Then the user writes minispecs demonstrating the algorithmic structure of these atomic processes. Also, a data dictionary is created containing the structure of the data. SA also gives naming conventions for processes, dataflows, files, which can help the user to express his understanding most clearly.

3.2.2 The Language

The sources and sinks belonging to the environment of the target system are shown as boxes on a data-flow diagram. Other symbols are circles representing processes, arrows representing data flows, and bars representing files. Please note that the first time a file is referenced in a DFD two bars are used (see fig. 7a, file "Bit Map") while further references to this file (in other DFDs) are denoted by a single bar (see fig. 8a, file "Bit Map").
The minispecs are written in pseudo-code, the data described in the data dictionary is written in a BNF-like notation.

The examples given below were taken from a paper on the Tektronix-tool (Bell, 1985). They show data-flow diagrams, together with minispecs and information stored in the data dictionary.

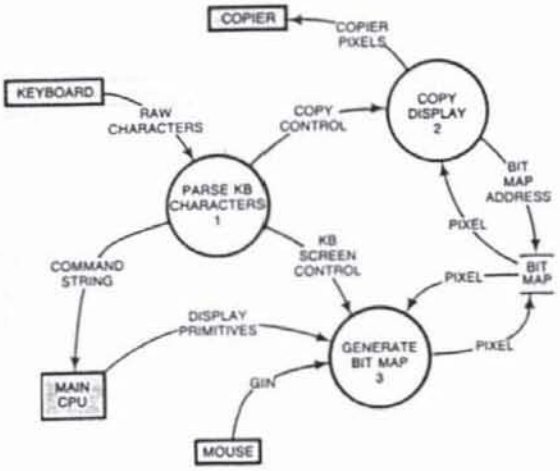

Figure 7a: DFD for a display controller

```
COPIER_PIXEL = PIXEL
BIT_MAP_ADDRESS = INTEGER
PIXEL = LOGICAL
TEXT = ASCII_CHAR
GRAPHICS = [POLYLINE | POLYMARKER | AREA FILL | GDP]
SCREEN_CONTROL = [SCROLL | ERASE | REVERSE | HORIZ_SCROLL]
KB_SCREEN_CONTROL = SCREEN_CONTROL
BIT_MAP = {PIXEL}
BIT_MAP_PIXEL = PIXEL
TEXT_PIXEL = PIXEL
GRAPHICS_PIXEL = PIXEL
COMMAND_STRING = {ASCII_CHAR} + DELIMITER
DISPLAY_PRIMITIVES = GRAPHICS + TEXT + SCREEN_CONTROL
GIN = X_POSITION + Y_POSITION
X_POSITION, Y_POSITION = INTEGER
```

Figure 7b: DD for 7 a

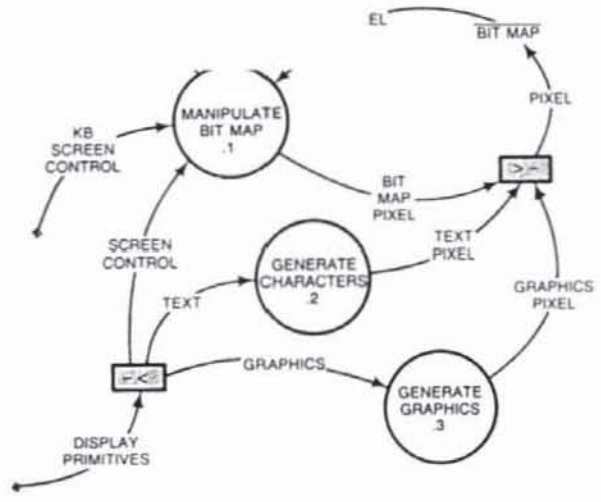

Figure 8a: DFD for Generate Bit Map from figure 7.a

```
CHARACTER_GENERATION_MAP_LOCATION = ASCII_CHAR
FOR I = 1 TO 12 DO
    CHAR_GEN_MAP_INDEX = I
        FOR J = 1 TO 9 DO
            IF CHARACTER_GEN_MAP_CONTENTS (J) = TRUE
                SEND 1 TO BIT MAP
            ELSE
                SEND 0 TO BIT MAP.
        END
    END
```

Figure 8b: Minispec for 8a

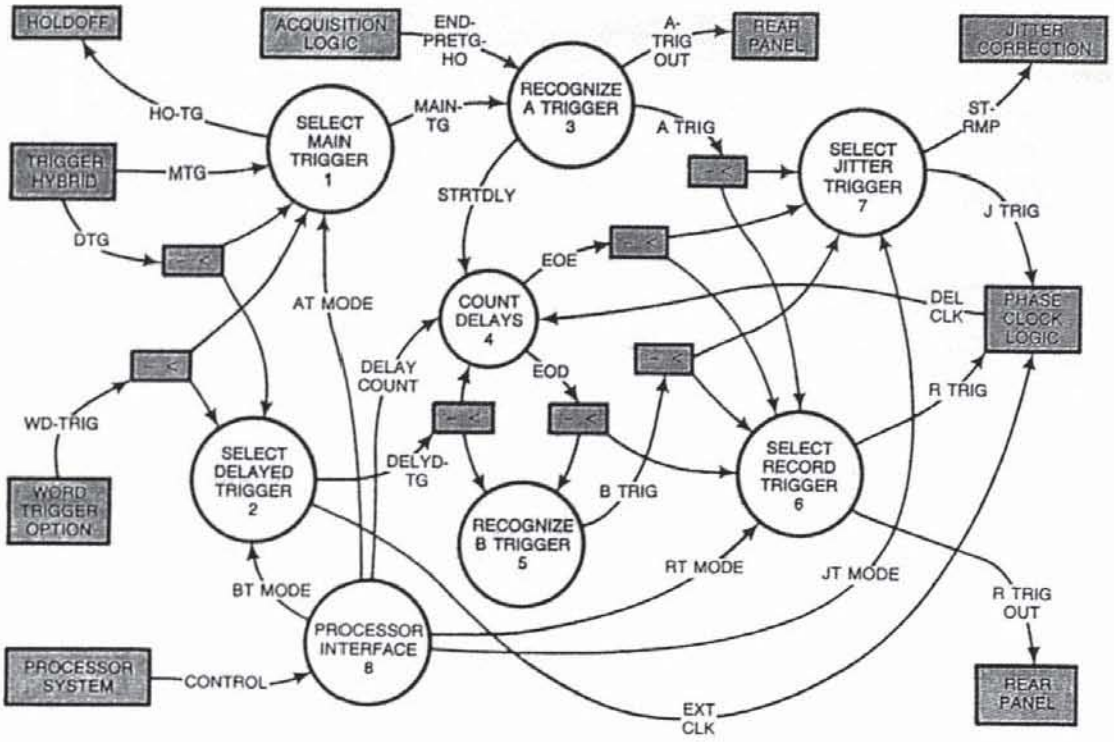

Figure 9: Top level DFD of a trigger gate array

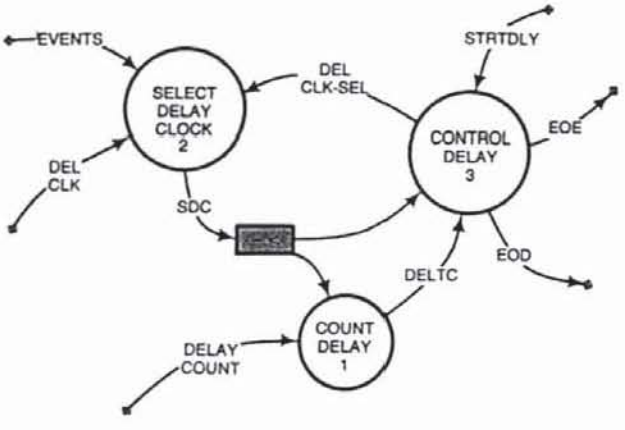

Figure 10: DFD of Count Delays (from figure 9)

MINISPEC 4.3

CIRCUIT ELEMENTS: 2FF2, 2FF3, 2FF4, 2G4, 2G5

OVERVIEW: THIS CIRCUIT IS A 3-FLIP-FLOP STATE MACHINE. 2FF2 CONTROLS THE START OF COUNTING DELAY, 2FF3 SETS AT THE END OF EVENTS COUNT, AND 2FF4 SETS AT THE END OF THE TIME-DELAY COUNT. SPECIAL-CASE COUNTS OF NO EVENTS AND 1 EVENT ARE CONTROLLED BY LEVEL INPUTS SET BY THE PROCESSOR. THE INITIAL STATE OCCURS WHEN THE PROCESSOR STROBES RSTACQ. THIS CLEARS 2FF2, WHOSE QBAR OUTPUT CLEARS 2FF4. 2FF3 IS CLEARED BY THE A TRIGGER FLIP-FLOP 1FF1. THE FIRST DCLK AFTER A TRIGGER WILL SET 2FF2 TO ENABLE THE DELAY COUNTER. IF ONEVNT = 1, 2FF3 WILL ALSO SET AT THIS TIME. DCLKS WILL BE COUNTED UNTIL DELTC = 1, CAUSING 2FF3 TO SET. WHEN EOE = 1, THE SELECT DELAY CLOCK LOGIC SWITCHES TO COUNTING DELAY BY TIME. THIS WILL CONTINUE UNTIL THE NEXT OCCURRENCE OF DELTC = 1, WHEN EOD = 1 WILL OCCUR. THE STATE MACHINE REMAINS IN THIS STATE UNTIL THE NEXT RSTACQ.

LOGIC:

ALL FLIP-FLOPS ARE RESET ASYNCHRONOUSLY BY PROCESSOR ACTION·

SET STRTDEL = 0 WHEN RSTACQ = 1
SET EOD = 0 WHEN STRTDELB = 1
SET EOE = 0 WHEN ATB = 1

ALL FLIP-FLOPS WILL SET ON CONDITION ON THE RISING EDGE OF DCLK

SET STRTDEL = 1 WHEN AT = 1 (RESETS ARE NOW REMOVED FROM 2FF3, 2FF4)
SET EOE(N+1) = ONEVNT + DELTC + EOE + NOEVNTS
SET EOD(N+1) = (EVDN + EOD)*(DELTC + EOD) = EVDN*DELTC + EOD

Figure 11: Minispec of Control Delay (from figure 10)

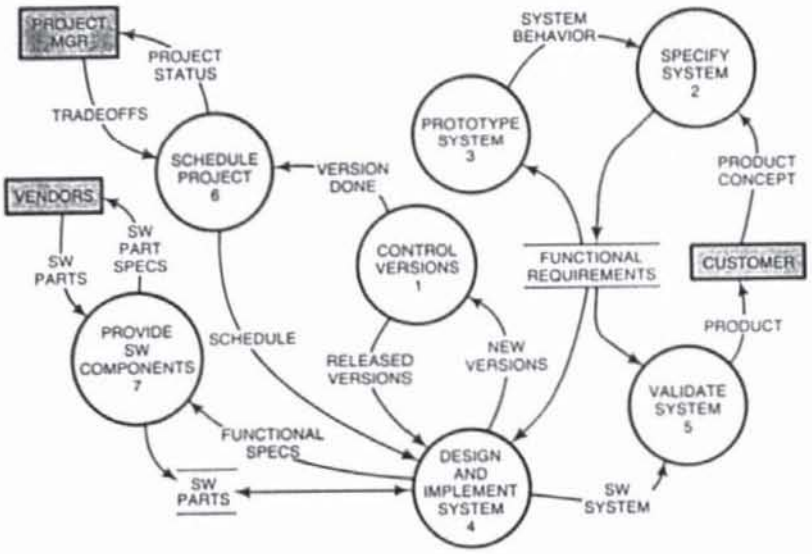

Figure 12: DFD of a product development

3.2.3 Tools for SA: proMod (Projektmodell)

proMod was developed bei GEI, Aachen, FRG. It is based on the Structured Analysis/Structured Design-concept.

While the project proceeds, all information is accumulated in the proMod project library.

Tools for Structured Analysis are:

DFD-processor	editing and processing of data flow diagrams
DD-processor	data dictionary system
TD-proccessor	minispec - processor
AAD-analyzer	cross checking between DFDs, DD and minispecs.

Toools for Structured Design are:

Translator	from SA - to SD - System
MS - processor	for module specifications
FS - processor	for functional specifications
DD - processor	data dictionary system
SE - analyzer	cross-checking at SD - level

Below the level of SD, proMod provides PDL and DARTS, two pseudo-code-systems. Other tools generate code-frames in several languages (PASCAL, FORTRAN, COBOL).
proMod is available on VAX/VMS, and IBM-PC (XT, AT) / PC-DOS.

3.3 Problem Statement Language/Problem Statement Analyzer (PSL/PSA)

PSL was developed at the University of Michigan by the ISDOS-project (Information System Design and Optimization System) in the seventies. PSL primarily supports requirements analysis and documentation.

Like some other systems developed at a university, PLS/PSA is now supported, improved, and commercially distributed by a private company (META-systems).

3.3.1 The Method

PSL/PSA was the very first tool-based system for semi-formal specification which was actually useful - and commercially successful. All other such systems are copies of PSL/PSA, at least in part.

PSL/PSA emerged since 1970 in a very organic manner, and Daniel Teichroew and his co-workers did never put too much effort in writing down the method they had in mind. Still, there is a method behind PSL: It is the one sketched in 2.4.

3.3.2 The Language

PSL is based on the entity-relationship approach first described by Chen (Chen, 1976), but applied long before. The entity-relationship model was originally used as a database model splitting the world to be described into entities, and relationships between these entities. The dominant feature of this approach is the similar treatment of entities and relationships.

Different from SADT and SA, PSL is a linear (textual) language. PSL provides some 30 entity-classes and 75 relations to the user. The most important ones are:

Entity-classes:

REAL WORLD ENTITY	objects outside the target system
PROCESS	activities
INPUT	input data
SET	set of data elements

Relations:

GENERATES	e.g. <process> GENERATES <data>
RECEIVES	e.g. <process> RECEIVES <data>
UPDATES	e.g. <process> UPDATES <data>
CONSISTS	describes data structures; e.g. colour CONSISTS yellow, red, green, blue

```
PSA Version A5.2R2M                          Jul 23, 1983  20:05:19
                          PSL/PSA - ISDOS - VM/CMS

                          IPSL Input Source Listing

 Parameters:   DB=VESSEL.DBF  INPUT=VESSEL.PSL  SOURCE-LISTING  NOCROSS-REFERENCE
   UPDATE  DATABASE-REFERENCE  NOWARN-NEW-OBJECTS  NOSTATEMENT-NUMBERS
   DBNBUF=200  WIDTH=84  LINES=60  INDENT=0  HEADING  PARAMETERS  PAGE-CC=ON
   NOEXPLANATION

 LINE  S T M T

    1 >/*  This is a set of PSL statements to define user views */
    2 >
    3 >/*  Here is the global users' view  */
    4 >
    5 >DEF ENTITY         Userviews;
    6 >   TKEY            'Global';
    7 >   SUBPARTS ARE    User-View-1,
    8 >                   User-View-2,
    9 >                   User-View-3,
   10 >                   User-View-4,
   11 >                   User-View-5,
   12 >                   User-View-6,
   13 >                   User-View-7;
   14 >   DESC;
   15 >This is a global view of a ship company.;
   16 >
   17 >
   18 >/*  ELEMENTs are declared */
   19 >
   20 >DEF ELE            Vessel,Cargo-Volume,Details,Port,Date-of-Arrival,
   21 >                   Date-of-Departure,Consignee,Container#,Size,
   22 >                   Shipping-Agent,Waybill#,
   23 >                   Delivery-Date,Contents,
   24 >                   Handling-Instructions;
   25 >
   26 >
   27 >/*  Here is the local users' view  */
   28 >
   29 >DEF ENTITY         User-View-1;
   30 >   TKEY            'V1';
   31 >   CSTS OF         View1-Ship;
   32 >   ATTR ARE        FREQUENCY-IS          100,
   33 >                   TIMING-REQUIREMENT     25;
   34 >   RPD IS          'E. Basar';
   35 >   DESC;
   36 >Information is stored about each ship, including
   37 >the volume of its cargo storage capacity.;
   38 >
   39 >
   40 >DEF ENTITY         User-View-2;
   41 >   TKEY            'V2';
   42 >   CSTS OF         View2-Ship,
   43 >                   View2-Ship-Port,
   44 >                   View2-Port;
   45 >   ATTR ARE        FREQUENCY-IS          100,
   46 >                   TIMING-REQUIREMENT     50;
   47 >   RPD IS          'E. Basar';
   48 >   DESC;
```

Figure 13: PSL source listing (incomplete)

```
PSA Version A5.2R2M                              Jul 23, 1983  20:05:19
                              PSL/PSA - ISDOS - VM/CMS

                              IPSL Input Source Listing

  LINE  S T M T

    49 >A ship stops at many ports and it is necessary to
    50 >print out its itinerary.;
    51 >
    52 >
    53 >DEF ENTITY          User-View-3;
    54 >   TKEY             'V3';
    55 >   CSTS OF          View3-Consignee,
    56 >                    View3-Port,
    57 >                    View3-Ship,
    58 >                    View3-Container;
    59 >   ATTR ARE         FREQUENCY-IS          25,
    60 >                    TIMING-REQUIREMENT     7;
    61 >   RPD IS           'E. Basar';
    62 >   DESC;
    63 >Persons who ship goods are referred to as consignees.
    64 >Their goods must be crated or stored in shipping containers.
    65 >These are given a container identification number. A list
    66 >can be obtained, when requested, of what containers have
    67 >been sent by a consignee.;
    68 >
    69 >
    70 >DEF ENTITY          User-View-4;
    71 >   TKEY             'V4';
    72 >   CSTS OF          View4-Agent,
    73 >                    View4-Port,
    74 >                    View4-Container;
    75 >   ATTR ARE         FREQUENCY-IS         110,
    76 >                    TIMING-REQUIREMENT    75;
    77 >   RPD IS           'Chiang Wan';
    78 >   DESC;
    79 >The shipments are all handled by shipping agents. A
    80 >shipping-agent report must be generated, listing all
    81 >the containers that a given agent is handling and giving
    82 >their waybill numbers.;
    83 >
    84 >
    85 >DEF ENTITY          User-View-5;
    86 >   TKEY             'V5';
    87 >   CSTS OF          View5-Waybill,
    88 >                    View5-Port,
    89 >                    View5-Ship,
    90 >                    View5-Container;
    91 >   ATTR ARE         FREQUENCY-IS         100,
    92 >                    TIMING-REQUIREMENT    50;
    93 >   DESC;
    94 >A waybill related to a shipment of goods between two
    95 >ports on a specified vessel. The shipment may consist
    96 >of one or more containers.;
    97 >
    98 >
    99 >DEF ENTITY          User-View-6;
   100 >   TKEY             'V6';
   101 >   CSTS OF          View6-Ship,
```

Figure 13: incomplete PSL source listing (continued)

Figure 13 shows a fragment of a PSL-input source listing; the specification describes cargo-vessels and their organizational environment.
(Source of all examples in 3.3: Papers from ISDOS, 1983)

3.3.3 The Tools

PSA, the tool, is actually the heart of the whole system. It is built upon a CODASYL-database system, and offers a large selection of services and report functions. PSA is a huge FORTRAN-program consisting of some 60 000 loc. It is available on almost any hardware and operating system; implementation on PCs was announced some time ago.

Two reports follow below; the second one (figure 15) shows a tree-structure (the hierarchical content-relation) by indentation. The first one (figure 14) shows part of the the same information in a table. These examples represent the traditional position of the ISDOS-project, where all output had to be line-printer oriented. Therefore, pseudo-graphics was the best representation available. But the system has now been extended by new tools, which support also high-resolution diagrams (not shown here).

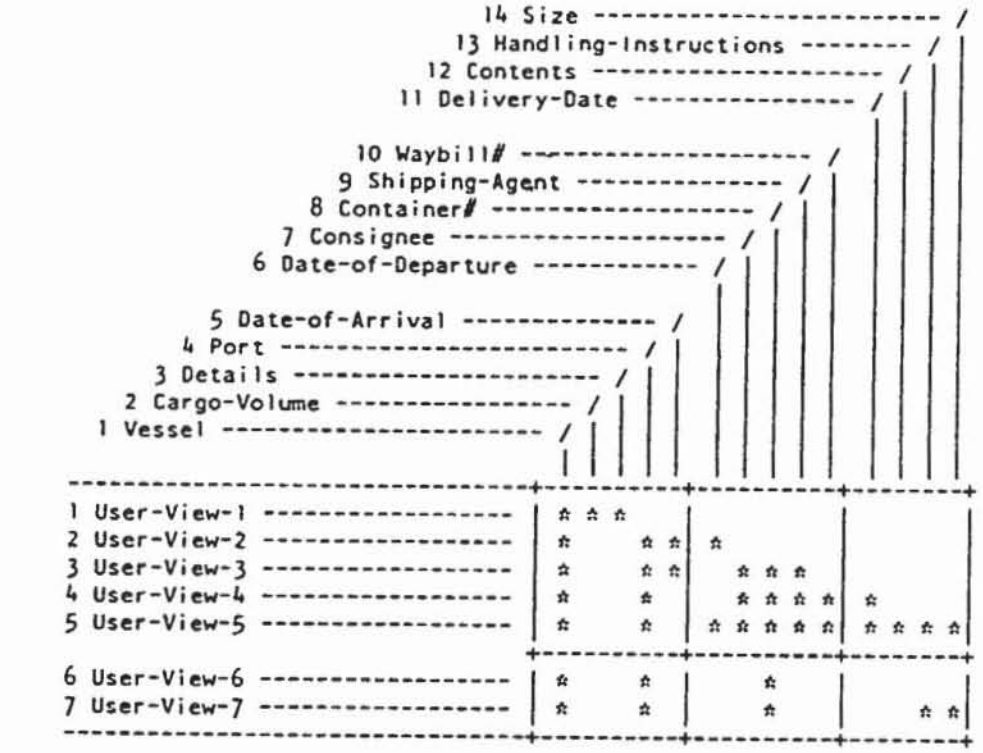

Figure 14: A PSA-report (Basic Content Matrix)

```
PSA Version A5.2R2M                          Jul 23, 1983  20:05:19
                          PSL/PSA - ISDOS - VM/CMS

                              Contents Report

 Parameters:   DB=VESSEL.DBF  FILE=PSANAMES.PSATEMP  NOCOMPLETENESS-CHECK
   NOINDEX  NOPUNCHED-NAMES  LEVELS=ALL  LINE-NUMBERS  LEVEL-NUMBERS
   OBJECT-TYPES  PRINT  NONEW-PAGE  DBNBUF=200  WIDTH=84  LINES=60  INDENT=0
   HEADING  PARAMETERS  PAGE-CC=ON  NOEXPLANATION

   1* (ENTITY)      1 User-View-1
    1 (GROUP)        2   View1-Ship
    2 (ELEMENT)      3     Vessel
    3 (ELEMENT)      3     Cargo-Volume
    4 (ELEMENT)      3     Details
   2* (ENTITY)      1 User-View-2
    1 (GROUP)        2   View2-Ship
    2 (ELEMENT)      3     Vessel
    3 (GROUP)        2   View2-Ship-Port
    4 (ELEMENT)      3     Port
    5 (ELEMENT)      3     Vessel
    6 (ELEMENT)      3     Date-of-Arrival
    7 (ELEMENT)      3     Date-of-Departure
    8 (GROUP)        3     View2-Ship (M-1)
    9 (ELEMENT)       4       Vessel
   10 (GROUP)        3     View2-Port (M-1)
   11 (ELEMENT)       4       Port
   12 (GROUP)        2   View2-Port
   13 (ELEMENT)      3     Port
   3* (ENTITY)      1 User-View-3
    1 (GROUP)        2   View3-Consignee
    2 (ELEMENT)      3     Consignee
    3 (GROUP)        3     View3-Container (M)
    4 (ELEMENT)       4       Container#
    5 (ELEMENT)       4       Date-of-Arrival
    6 (ELEMENT)       4       Shipping-Agent
    7 (GROUP)         4       View3-Port (1)
    8 (ELEMENT)        5         Port
    9 (GROUP)         4       View3-Ship (M-1)
   10 (ELEMENT)        5         Vessel
   11 (GROUP)        2   View3-Port
   12 (ELEMENT)      3     Port
   13 (GROUP)        2   View3-Ship
   14 (ELEMENT)      3     Vessel
   15 (GROUP)        2   View3-Container
   16 (ELEMENT)      3     Container#
   17 (ELEMENT)      3     Date-of-Arrival
   18 (ELEMENT)      3     Shipping-Agent
   19 (GROUP)        3     View3-Port (1)
   20 (ELEMENT)       4       Port
   21 (GROUP)        3     View3-Ship (M-1)
   22 (ELEMENT)       4       Vessel
   4* (ENTITY)      1 User-View-4
    1 (GROUP)        2   View4-Agent
    2 (ELEMENT)      3     Shipping-Agent
    3 (GROUP)        3     View4-Container (M)
    4 (ELEMENT)       4       Container#
    5 (ELEMENT)       4       Waybill#
    6 (ELEMENT)       4       Consignee
    7 (ELEMENT)       4       Vessel
```

Figure 15: A PSA report (Contents Report)

3.4 Software Requirements Engineering Methodology (SREM)

SREM is directly based on PSL/PSA; it was developed by TRW since about 1975. It supports the early phases (analysis, definition, verification, and validation of requirements) of the software development process. It is especially tailored for the development of large, embedded, real-time systems; the U.S. Air Force was the contractor of that project. For more information on SREM, see papers by M. Alford (references in 6.1.4).

3.4.1 The Method

SREM possesses two important features missing from most other methods or languages for specification. Firstly, it allows the stepwise development of specifications beginning with informal descriptions, from which an increasingly formal specification is developed. Secondly, data on performance (estimated or required) of the target system can be formally included in the specification. Since there is a tool for simulating specs, software designers can check early whether or not they will be able to meet response time requirements.

The method (SREM) is applied in seven steps:

1. Define kernel: identify the interface between the system and the environment and describe the data flows and the data-processing units inside the system.
2. Establish baseline: outline the very first description of the system using either the graphical R-Net formalism (R-Net means requirements-net, a stimulus-response network) or the linear language RSL (requirements statement language).
3. Define data: define data input to, and output from, each so called ALPHA (active component); complete, and improve the RSL-specification developed so far; implement Pascal-procedures for ALPHAs.
4. Add project information, and establish traceability: add management informations, e.g. deadlines, milestones, needed tools etc.
5. Simulate functionality: prove syntactical correctness and simulate dynamic behaviour
6. Identify performance requirements: define traceable, testable performance requirements; each path should be constrained by response time and accuracy
7. Demonstrate feasibility: prove that the current design is useful as a basis for a technical realization by means of a analytical feasibility study

3.4.2 The Language

SREM offers the user two means of description:
- a graphical language (**R-Nets**) and
- a textual language (**RSL**).

R-Nets are stimulus-response networks describing reactions in a system evoked by events. An R-Net consists of nodes (ALPHAs and SUBNETs) and arcs connecting the nodes. While ALPHAs are functional specifications of processes, SUBNETs are specifications of processes at a lower level of hierarchy. The flow of control is described by some single entry - single exit constructs (AND for parallel execution, OR for a multiway branch, FOR EACH for a loop). Additionally, validation-points can be inserted in order to express performance requirements.
See figure 16 for a list of all symbols used in R-Nets.

RSL is a textual specification language based on the following concepts:

Elements
are standard types defining features of each object of such a standard type. For example, MESSAGE, DATA, and FILE are standard types used to describe data; e.g. ALPHAs stand for processes. Elements represent nouns in the language.

Relationships
express logical links between Elements, e.g. <data> INPUT TO <alpha>. They represent verbs in the language.

Attributes
are used to complete the description of Elements, e.g. <data> INITIAL VALUE <value>. They represent adjectives in the language.

Structures
are used to define the sequences of processing steps and represent R-Nets, SUBNETs, and VALIDATION-PATHs in terms of RSL-statements.

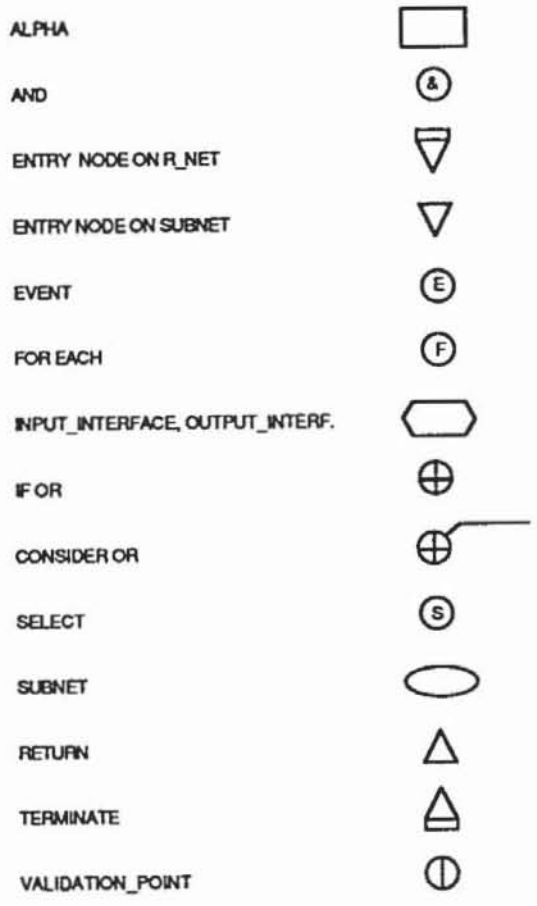

Figure 16: Symbols in SREM

RSL is also used to enter the R-Nets, which are then automatically drawn.

A few examples are given below. Figure 17 shows a schematic R-Net. In figures 18a and b both the RSL-representation and the flow graph representation of a sample R-Net are exhibited.
(Source: M. W. Alford, Proceedings of the COMPSAC Conferences 1978, 1980).

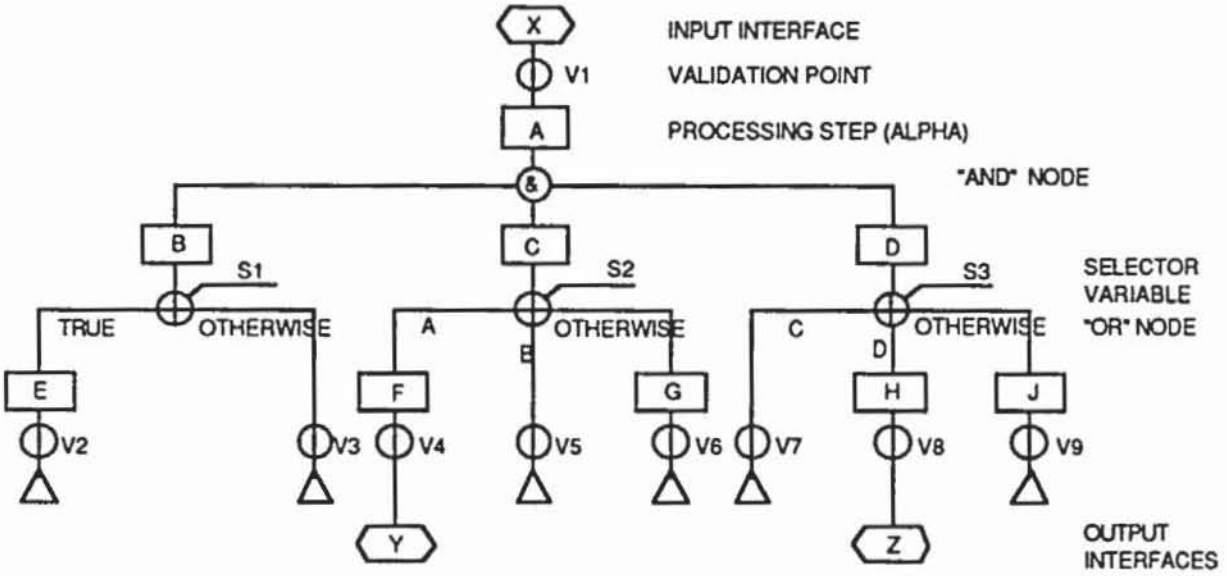

Figure 17: A schematic R-Net

3.4.3 The Tool

Like PSL/PSA, SREM is based on a large tool, called REVS (Requirements Engineering Validation System). Beyond the abilities of other tools, REVS allows for project dependent extensions of the specification language, and for simulation of the specs. Maybe that REVS is currently the most powerful tool for specification; but prospective customers in Europe cannot buy it because its distribution is still limited to the U.S.

```
R_NET:  PROCESS_RADAR_RETURN.
    STRUCTURE:
        INPUT_INTERFACE RADAR_RETURN_BUFFER
        EXTRACT MEASUREMENT
        DO (STATUS = VALID_RETURN)
            DO UPDATE_STATE AND KALMAN_FILTER END
            DETERMINE_ELEVATION
            DETERMINE_IF_REDUNDANT
            TERMINATE
        OTHERWISE
            DETERMINE_IF_OUTPUT_NEEDED
            DO  DETERMINE_IF_REDUNDANT
                DETERMINE_ELEVATION
                TERMINATE
            AND DETERMINE_IF_GHOST
                TERMINATE
            END
        END
    END.
```

Figure 18a: A sample R-Net, textual representation

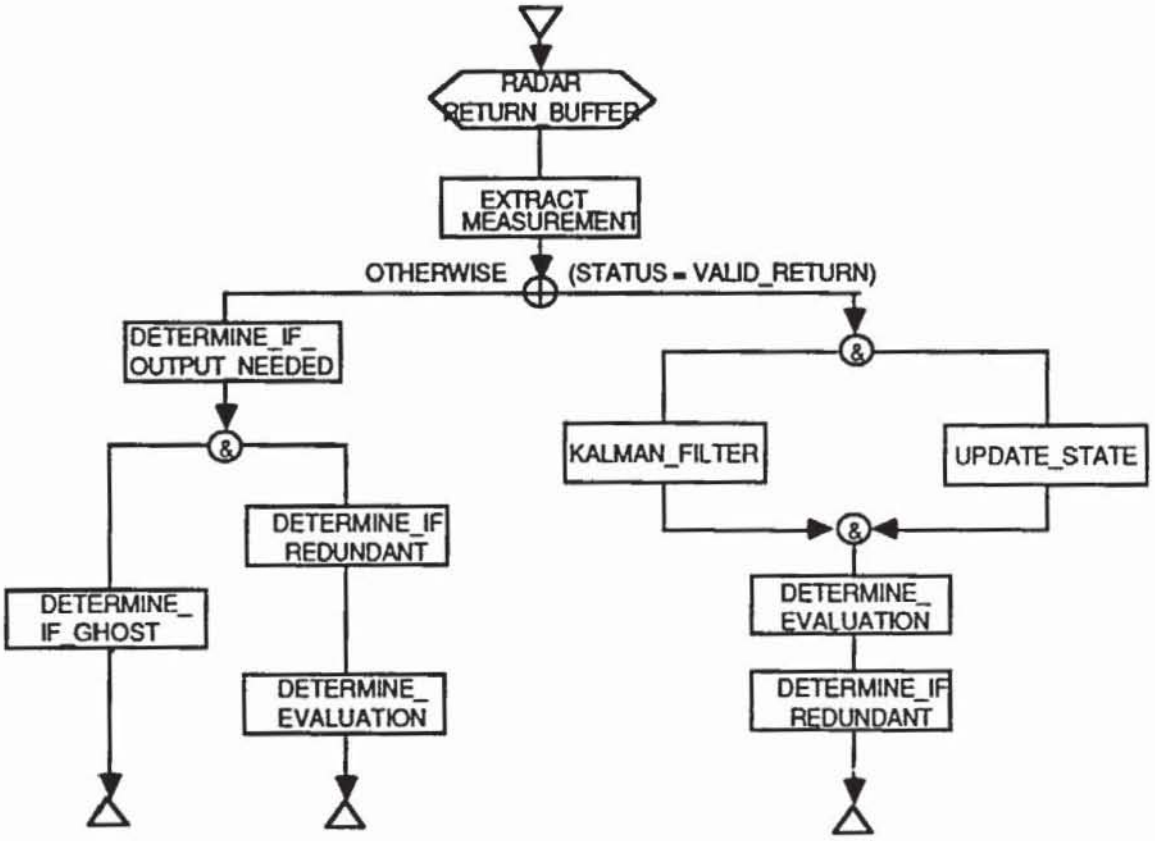

Figure 18b: A sample R-Net, flow graph representation

3.5 EPOS

(Entwicklungs- und Projektmanagement orientiertes Spezifikationssystem)

EPOS was developed at TU Stuttgart by R. Lauber and co-workers since 1978 (Biewald et al., 1979). The product is now sold and supported by GPP (see 6.2).

3.5.1 The Method

EPOS is one of the systems which do explicitely not support a particular method (though they do refer to the general principles of SADT).

3.5.2 Languages

In EPOS, there is no clear distinction between languages and tools, i.e. the same name is used both for the language and for the program which is used for processing that language. Therefore, the following list may be inconsistent with other papers on EPOS.

There are three languages used for input:

EPOS-R	language for requirements definition (formatted)
EPOS-S	language for system design (semi-formal)
EPOS-P	language for project management information (semi-formal)

Several graphical representations can be generated by the tools.

3.5.3 The Tools

The tools of EPOS are separated in four groups:

EPOS-M	Tools for project management
EPOS-A	Analyzer and report generator (for all levels)
EPOS-D	Generator for documentation (e.g. Petri-Nets, Nassi-Shneiderman-Diagrams)
EPOS-C	Human-computer interface of EPOS

Epos-A stores all information in a (non-standard) data base, which is accessed by all tools.

EPOS is available on PDP 11/RSX 11-M, VAX/VMS, IBM 370/VM/CMS, Intel 8086, 80286/iRMX and several other machines.

3.6 PRADOS (Projekt-Abwicklungs- und Dokumentations-System)

PRADOS was (and is still being) developed by SCS, Hamburg and Munich, FRG. It is tailored to UNIX and uses components of the UNIX-System.

3.6.1 The Method

While EPOS is based on a vague idea of a method, which has not been made explicit, PRADOS does not even have such an idea, because it is a collection of tools many of which existed before PRADOS was developed.

3.6.2 The Languages

The languages of PRADOS are the languages of its tools. When SADT is available, there will be at least one genuine specification language in the system.

3.6.3 The Tools

Three general components are:

UNIFY	relational database
XED	text processing system
IFE-GRAPH	business graphics system

Upon these, all other components are built:

PV, PM	project management tool
PB	project library
TE	text processing
MB	methods library
SADT	SADT Generator (announced)
DSA	data description based on entity-relationship-model
ESS	design specification language processor
PS	pseudo code pre-compiler
ST	Nassi-Shneiderman-chart-generator
BT & TR	test monitor

4. Management Aspects

There are two important management aspects in the topic of this paper:

First, the decision to use a specification system, and the choice of a particular product, require a commitment of the management. Introduction of a specification system is very expensive. The cost of the system itself and,

possibly, of new hardware is often high, but it is usually negligible compared to the cost of training (or the failures due to insuffient training). The step to using a specification system is of similar importance like the step to using a computer; if you are not prepared to do it right, don't do it at all! Problems are inevitable, and there will be a situation when an important project seems to be late, because it is done with a specification system. If the management is not prepared to show a bold front against the breakers, they will not succeed.

Second, the specification system may improve quality assurance and project control. Most vendors advertise some management tools as part of their products. To date, these are not very powerful. The real improvement stems from the disciplin and standardization implied by the application of a specification system. This side effect is in fact the main advantage of a specification system!

5. Conclusions

- There are many specification systems commercially available. Everybody who uses any of the more common machines, and operating systems, will find a specification system, if he or she wants to.

- It is obviously still possible to produce software (and systems) without a specification system. Special problems, like developing user interfaces, are actually better done by other approaches, e.g. prototyping.

- A specification system causes large expenses, mainly for training, but can improve quality and productivity significantly. Therefore, it should be regarded as a (medium- or long-range) investment.

- A specification system improves standardization in the way that every member of a project uses the same method, the same language, and the same tool. Moreover, the documents itself have standardized features. This implies a discipline which is the real benefit of a specification system!

- Vendors say little about the methods, which are most important for the customers.

- Maintenance of specifications is not yet supported by the tools. Therefore, the responsibility to change all documents, when one is modified, rests with the user. If he or she fails to do so (what is the normal situation), the specification becomes obsolete.

- Implementing one's own specification system is hardly feasible, because it takes at least ten person years to develop just a prototype.

6. References and Addresses of Suppliers

6.1 References

6.1.1 Textbooks on Software Engineering

Boehm, B. W. (1980): **Software Engineering Economics.**
Prentice Hall, Englewood Cliffs, N.J.

Fairley, R. (1985): **Software Engineering Concepts.**
McGraw-Hill Book Company, New York usw.

Sommerville, I. (1985): **Software Engineering**. Addison-Wesley Publishing Company, London etc., 2nd ed.

6.1.2 The Life Cycle

Lehman, M.M. (1980): Programs, life cycles, and laws of software evolution.
Proc. of the IEEE, 68, 9, 1060-1076.

Ludewig, J. (1982): Computer aided specification of process control software.
IEEE COMPUTER, 15, 5, 12-20.

Swartout, W., R. Balzer (1982): On the inevitable intertwining of specification and implementation. **Commun. ACM, 25**, 7, 438-440.

6.1.3 Fundamentals and Principles of Specification

Balzer, R., N. Goldman (1979): Principles of good software specification and their implications for specification languages.
in **Proceedings of Specification of Reliable Software (SRS),**
IEEE Cat. No. 79 CH 1402-9C, pp.58-67.

Boehm, B.W. (1976): Software Engineering.
IEEE Transactions on Computers, C-25, pp.1226-1241.

Brooks, W.D. (1981): Software Technology Payoff: Some statistical evidence.
Journal of Systems and Software, 2, 3-9.

IEEE (1983): Standard glossary of software engineering terminology.
IEEE Std 729-1983.

Kramer, J. (ed.) (1982): Glossary of terms.
TC on Application Oriented Specification.
Jeffrey Kramer, Imperial College, 180 Queen's Gate, GB - London SW7 2BZ.

Parnas, D.L. (1977): The use of precise specifications in the development of software. in Gilchrist, B. (ed.): **Information Processing 77**. North Holland Publishing Company, Amsterdam, New York, Oxford, pp.861-867.

Timm, M. (1982): **Grundlagen von Anforderungs- und Entwurfsspezifikationen im Prozeß der Software-Entwicklung**. GMD-Studien, Nr. 66, 82 S.

6.1.4 Surveys (Articles and Books)

Balzert, H. (1982): **Die Entwicklung von Software-Systemen**. Reihe Informatik/34, Bibliographisches Institut, Mannheim.

Hommel, G. (Hrsg.) (1980): **Vergleich verschiedener Spezifikationsverfahren, am Beispiel einer Paketverteilanlage**. KfK-PDV 186, Teile 1 und 2, Kernforschungszentrum Karlsruhe, BRD.

Cheng, L.L. (1978): Program design languages - an introduction. Report No. ESD-TR-77-324, Electronic Systems Division, Hanscom Air Force Base, MA 01731.

COMPUTER (1982): Special issue on application oriented specification. **IEEE COMPUTER 15**, 5 (May 1982), 10-59.

COMPUTER (1985): Special issue on requirements engineering environments. **IEEE COMPUTER 18**, 4 (April 1985), 9-91.

IEEE-SE (1977): Special collection on requirements analysis. **IEEE Trans. Software Eng.**, SE-3, 2-84.

Ludewig, J., W. Streng (1978): Methods and tools for software specification and design - a survey. **EWICS TC on Safety and Security**, Paper No. 149, 23 Seiten.

Ludewig, J. (Hrsg.) (1983): **Spezifikation von Realzeit-Systemen - Konzepte, Lösungen, Erfahrungen**. 54. Tagung der Schweizerischen Gesellschaft für Automatik (SGA-ASSPA), Baden/Aargau, 1983- 3-21.

Ohno, Y. (ed.) (1982): **Requirements Engineering Environments**. Proceedings of the International Symposium on current issues of Requirements Engineering Environments; Kyoto, Japan, September 20-21, 1982. NHPC, Amsterdam usw.
(some of the papers have also been published in COMPUTER, 1985).

Prentice, D. (1981): An analysis of software development environments. **ACM SIGSOFT Software Engineering Notes**, 6, No.5, 19-27.

Ramamoorthy, C.V., H.H. So (1977): Survey of principles and techniques of software requirements and specifications. in **Software Engineering Techniques**, Vol.2, Infotech Intern. Ltd., Nicholson House, Maidenhead, Berkshire, England, pp.265-318.

6.1.5 Particular Specification Methods and Systems

Alford, M. (1977): A requirements engineering methodology for real time processing requirements. **IEEE Transactions on Software Engineering**, SE-3, 60-69. (on SREM)

Alford, M. (1985): SREM at the age of eight: The distributed computing design system. **IEEE COMPUTER 18**, 4, 36-46.

Balzert, H. (1985): **Moderne Software-Entwicklungssysteme und Werkzeuge**. Reihe Informatik/44, Bibliographisches Institut, Mannheim. (contains material on proMod, PRADOS and other systems; in German)

Bell, R. (1985): Structured analysis aids in micro-computer system design. **EDN**, March 21, 1985, 251-257. (on Structured Analysis)

Biewald, J., P. Göhner, R. Lauber, H. Schelling (1979): EPOS - a specification and design technique for computer controlled real-time automation systems. **4th Intern. Conf. on Software Engineering**, München, 1979, IEEE Cat. No. 79 CH 1479 - 9C, pp.245-250.

Hamilton, M., S. Zeldin (1976): Higher Order Software - a methodology for defining software. **IEEE Transactions on Software Engineering, SE-2**, 9-32. (on HOS)

Lissandre, M., P. Lagier, A. Skalli, H. Massié (1984): SPECIF - A specification assistance system. Institut de Génie Logiciel, Paris, France. (SADT-tool)

Ludewig, J., M. Glinz, H.J. Huser, G. Matheis, H. Matheis, M.F. Schmidt (1985): SPADES - A Specification and Design System and its Graphical Interface. **8th Intern. Conf. on Software Engineering**, IEEE CH2139-4/85/0000/0083, 83-89.

Ross, D.T. (1977): Structured analysis (SA): A language for communicating ideas. **IEEE Trans. on Software Engineering, SE-3**, 16-34. (on SADT)

Ross, D.T. (1985 a): Applications and extensions of SADT.
IEEE COMPUTER 18, 4, 25-34.

Ross, D.T. (1985 b): Douglas Ross talks about Structured Analysis.
IEEE COMPUTER 18, 7, 80-88. (on SADT)

Teichroew, D., E.A. Hershey III (1977): PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems. **IEEE Trans. Software Eng., SE-3**, 41-48.

Yourdon, E., L.L. Constantine (1979): **Structured Design: Fundamentals of a disciplin of computer programs and systems design**.
Prentice Hall Inc., Englewood Cliffs.

6.1.6 Use of Programming Languages for Specification, Prototyping

Goldsack, S.J. (ed.) (1985): **Ada for specification: Possibilities and limitations**. Cambridge University Press (for the Commission of the EC).

Boehm, B.W., T.E. Gray, Th. Seewald (1984): Prototyping versus Specifying: A multi-project experiment. **7th ICSE**, Orlando, FL., March 1984, 473-484; also in **IEEE Trans. on SE, SE-10**, 290-303.

Budde, R. K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (eds.) (1984): **Approaches to Prototyping**. Springer-Verlag, Berlin etc.

6.1.7 Software Engineering Environments

Howden, W. (1982): Contemporary software development environments.
Comm. ACM, 25, 5, 318-329.

Hünke, H. (ed.) (1981): **Software Engineering environments**.
Proc. of the Symposium held at Lahnstein, June 16 - 20, 1980.
North Holland Publishing Company, Amsterdam, New York, Oxford.

Osterweil, L. (1981): Software environment research: directions for the next five years. **IEEE COMPUTER**, April 1981, 35-43.

6.2 Adresses of Vendors

Please note that the following list is rather arbitrary, and far from complete, and it does not imply a judgement or recommendation !

EPOS (Entwurfsunterstützendes Prozeß-Orientiertes Spezifikationssystem)
GPP, Kolpingring 18a, D 8024 Oberhaching. Tel. D 089 - 61 10 42 18

HOS (Higher Order Software) and **USE.IT**
Higher Order Software, Inc., 2067 Massachusetts Avenue
Cambridge, Massachusetts 02140, USA, Tel. USA 617-661-8900

MASCOT (Modular Approach to Software Construction, Operation, and Test)
MASCOT Suppliers Association
c/o Computing Standards Section, Room L303, Royal Signals and Radar Establishment, St. Andrews Rd., Malvern, Worcestershire, WR14 3PS, GB

Perspective
Software Technology Centre, System Designers Ltd.,
Systems House, 1 Pembroke Broadway,
Camberley, Surrey GU15 3XH Great Britain, Tel. GB (0276) 62244

PET (Programm-Entwicklungs-Terminalsystem)
PHILIPS AG, Data Syst., Allmendstr. 140, 8027 Zürich, Tel. CH 01 432211

PRADOS (Projekt-Abwicklungs- und Dokumentationssystem)
SCS Techn. Autom. und Systeme GmbH,
Hörselbergstr.3, D 8000 München 80, Tel. D 089 - 41 27 - 0

proMod (Projektmodell)
GEI, Albert Einstein Str. 61, D-5100 Aachen, Tel. D 02408 - 130

PSL/PSA (Problem Statement Language/Analyzer) and related systems
META-systems, 315 E. Eisenhower Pkwy., Suite 200,
Ann Arbor, MI 48104, USA; Tel. USA (313) 663-6027

Softool (Softw. Management, Developm., Maintenance, and Conversion Tools)
via mbp, Semerteichstr. 47, D 4600 Dortmund 1, Tel. D 0049 231 43480
or directly from SOFTOOL Co., 340 S. Kellogg Av., Goleta, CA 93117

SPECIF (Specification System) for SADT-Diagrams
Institut de Génie Logiciel (IGL), 39 rue de la Chausée d'Antin,
F-75009 Paris, France; Tel. F (33) 1 281 41 33

Tektronix SA Tools (Structured Analysis)
Tektronix Inc, P.O.B. 500, Beaverton, OR 97077, Tel. USA (503) 627-7111

TOPAS-N (alias **NET**, an editor and simulator for extended Petri-Nets)
TOPAS-B (formerly **BOIE**, a tree-oriented development tool)
PSI GmbH, Heilbronner Straße 10, D 1000 Berlin 31, Tel. 0049 30 890090