SPADES - A SPECIFICATION AND DESIGN SYSTEM AND ITS GRAPHICAL INTERFACE

J. Ludewig, M. Glinz, H. Huser, G. Matheis, H. Matheis, M.F. Schmidt

Brown Boveri Research Center
CH - 5405 Baden (Switzerland)

ABSTRACT

SPADES is a specification system consisting of a method, a language, and a set of tools. These components are based on a set of concepts, which forms its abstract kernel.
SPADES supports the specification of software systems, in particular of real time software. The system to be developed is modelled using the Entity-Relationship-concept. While this seems to be the best way for storing specifications in a computer, it does not automatically lead to representations equally comfortable for humans. This is why SPADES, which has been available for some time, has recently been extended by a graphical interface. This paper gives a brief survey of the system, in particular of its new component.

INTRODUCTION

SPADES, which stands for SPecification And DESign System, is only the last step in a chain of efforts for providing a useful specification system. PCSL[1] was a modified version of the well known PSL/PSA[2], which contributed much more to the evolution of specification systems than any other system. Though very similar in syntax to PSL, PCSL differs in some concepts for specifying real time software; these concepts were partially new, and partially taken from SREM[3] and other systems. A radically new implementation of PCSL allowed for a modern, block oriented syntax of the language, and some new features for the tool. The result was named ESPRESO[4]. While PSL/PSA was implemented in FORTRAN and supported FORTRAN-style specifications, PASCAL took over both positions in ESPRESO. The most recent improvement was achieved by reimplementing ESPRESO under the new name SPADES in MODULA-2. SPADES is currently being used at Brown Boveri in a pilot project. With the feedback from its users, SPADES is continuously being improved and expanded at Brown Boveri Research Center.

Compared to PSL/PSA, SPADES is still a very small system, limited to a certain application area and a specific operating system (VAX/VMS). SPADES-T, its tool, is far less powerful than PSA is. Still, SPADES has a number of advantages for us:

- Since it is fairly small and implemented in a high level language, SPADES can be modified and adapted to the user's needs with comparative ease.
- SPADES was build in a real inside-out manner; therefore, its concepts have not been added for marketing reasons, but formed the building block of the development. While we will sometimes argue about details of the implementation, we have had no reason to change the concepts so far.
- Unlike other specification languages, SPADES-L was rigorously defined by an Attribute-Grammar. As long as there did not exist any proper material for the user[5], this definition was sometimes mistaken for a manual, which it is certainly not. But, as a reference for work on the system, this grammar turned out to be extremely useful.
- Working on a Software Engineering Environment[6], we benefit again and again from our experience with SPADES. Some of the concepts, which were no more than ideas when we made SPADES, are now fully understood, and are being implemented.

COMPONENTS OF SPADES

Some specification systems are referred to as methods, or languages, or tools. We think that none of these is very useful without the others; all must be taken into account. A set of concepts forming the abstract kernel should guarantee that method, language, and tool go well together, and not only syntactically. This logical structure is represented by the "system-triangle" (fig. 1).

In SPADES, the components are named:

- SPADES-M: the method for developing specifications
- SPADES-L: the standard language for entering specifications
- SPADES-T: the set of tools for working on specifications

The following chapters describe concepts, method, language, and tools in this order. This seems to be the most logical ordering, and also corresponds

to the formation of SPADES. However, as the dotted lines in the system triangle indicate, all four depend on each other. That is why it is not always possible to avoid repetitions and forward references.

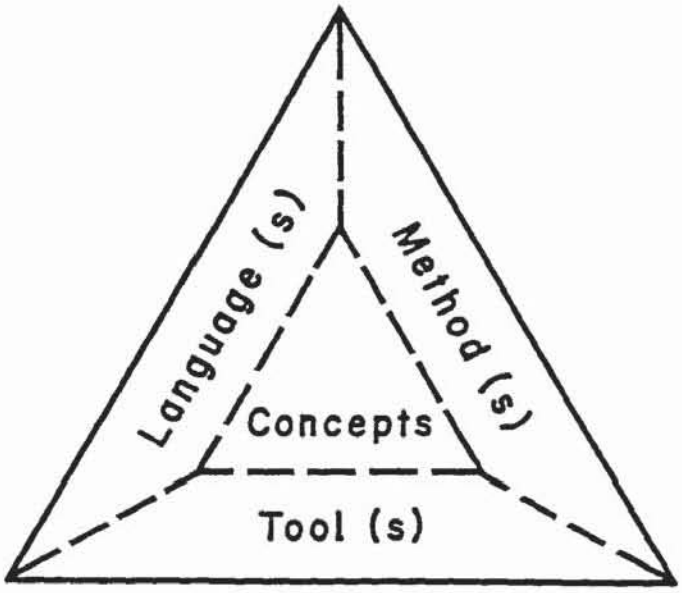

Figure 1: The system triangle

CONCEPTS

The Entity Relationship Concept

Like many other systems, SPADES is based on a modified Entity-Relationship-concept[7], i.e. reality is conceptualized by objects (entities) and links (relationships). Objects and links are the nodes of a bipartite graph, which, as a whole, models the target system and, possibly, parts of its environment.

The set of all objects is subdivided into predefined classes, the kinds (e.g. "MODULE" or "BUFFER"); likewise, every link belongs to one of the given relations (e.g. "COMPRISES" or "CONSUMES FROM").

Most relations require two objects to be linked; each of these so called components of the link plays a specific role. In the comprise-relation, for instance, one component models "the whole", the other one "the part"; since we cannot store 1 to n relations, each subpart needs a special link (see fig. 2).

There are also (pseudo-)relations with only one component, called predicates, and some with three components, one of which is often optional.

Every object is uniquely identified by a user-defined name, while links are not. Instead, a link is distinguished from another link only by the combination of its relation and its components, together with its role (e.g. link of relation "comprises" with ABC as the whole and UV as the part in fig. 2).

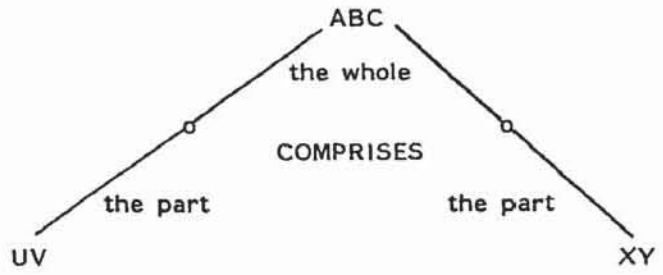

Figure 2: Named objects, links, and roles

An Overview of Kinds and Relations

The kinds of objects in SPADES include

- MODULES for describing the static structure of systems,
- actors (PROCEDURES and BLOCKS) which may be executed,
- parameters (INPAR, OUTPAR, TRANSPAR) related to procedures,
- media (VARIABLES, BUFFERS, TRIGGERS, and RESOURCES) which are used for communication between active components,
- TYPES for defining the structures of variables, buffers, and parameters,
- INTERVALS for specifying dynamic properties like delays, frequencies etc.,
- CONSTANTS for representing system parameters,
- INFORMAL OBJECTS for documentation only.

Relations are provided for describing

- hierarchies and other structures (e.g. nesting of modules, composite types)
- communication between actors and media,
- coordination and timing between actors, media, and intervals,
- execution schedules for actors,
- restrictions about the access to actors and media,
- general references to arbitrary objects.

Other Concepts

SPADES is based on the concept of a smooth transition from a vague idea to a fairly well structured description of the target system. The latter may also be regarded as a high level design; we no longer believe in a strict separation of WHAT and HOW[8,9].

In order to allow for such a smooth transition, constructs offered to the user range from informal texts (kind "INFORMAL") to very specific objects, whose kinds im ply well defined properties (see the paragraph on SPADES-L below).

Certainly an important concept is our intention to make SPADES a simple specification system (but not primitive), in the same way that PASCAL is a simple programming language.

THE METHOD SPADES-M

There should be one and only one specification, which leads to it being stored in a database. This specification should remain in a consistent state at any time; therefore, users must not modify the content of the database directly, but only through well controlled mechanisms. When an analyst is working on the database, he or she should never be in doubt whether the specification (or a part thereof) is currently in the analyst's, or in the database management's custody.

SPADES is based on the idea that all information relevant to the target system should be collected as early as possible. In an early state, however, very little information is formalized, therefore, informal texts play an important role. The user is then solicited to formalize the (already stored) knowledge. Together with the customer or any other person who represents the requirements, the specifier should check the stored specification after every major extension or change. This should be done even when it is far from complete, in order to avoid a costly trip on the wrong track.

These methodological goals are reflected in the design of SPADES-L, the language, and SPADES-T, the tool. For instance, an object ABC which has not yet been defined explicitely may nevertheless be referenced from other objects; SPADES-T will evaluate the context in order to reduce the range of possible kinds associated with ABC. When ABC is actually defined later on, or mentioned again in some other context, SPADES-T will "know" whether the occurences are still consistent.

THE LANGUAGE SPADES-L

Many properties of SPADES-L are determined from its purpose, and from the concepts stated above.

At first glance, SPADES-L resembles PASCAL, or rather ALGOL60, because word-symbols start, and may end, with a quote, distinguishing them from user-chosen identifiers.

Every object may be described by an arbitrary number of informal texts. Some objects are never used for any other purpose; these are objects of kind INFORMAL. Like objects of some other kinds, informal objects may be given a hierarchical (i.e. tree) structure.

Each text is identified by the object to which it is attached, and by a so called text-selector. Since text-selectors must be unique within one object only, certain ones may be used for specifying certain aspects (e.g. every object may have a text with selector "deadline"). Texts may contain references, i.e. identifiers of objects preceded by an exclamation mark. Such references are automatically detected and evaluated by SPADES-T. This construct reflects a very common situation at the beginning of program development, when we know vaguely about relationships, but cannot yet formalize them. For instance, we may attach a text to module A expressing that details of A depend on certain properties of module B (reference to B). When we retrieve the specification of B in order to modify it, we come across that reference.

Hierarchical structures can be described in two ways: The natural solution is a likewise hierarchical (nested) description (see the modules in fig. 3). In a specification language, such a description is not always desirable: when information on a certain structure emerges only slowly, the analyst wants to enter it in many small portions. This, altogether, form a flat, less elegant relational structure, in which the hierarchical structure is described by explicit relationships rather than by a corresponding syntactical structure. Therefore, SPADES-L allows for both formats.

```
'MODULE mixer-control-system: "example for ICSE";
(* For every object, a descriptional text should be
   entered. Therefore, SPADES allows for a text
                              with empty selector. *)

'COMPRISES
   'MODULE control  (* short section, header only *)
'AND
   'MODULE mixer:     (* section-header with colon *)
   'COMPRISES            (* begin of section-body *)
      'MODULE mixer-working   (* a nested section *)
   'AND
      'MODULE mixer-supervising
   'END mixer                          (* section-tail *)
'AND
   'MODULE liquid-supply:
   'COMPRISES
      'BLOCK weigh-A:
      'STARTED-BY start-button;
      'TERMINATED-BY stop-button;
      'CONSUMES scales-reading;
      'UPDATES liquid-item-counter;
      'WRITES VA-signal 'WHERE "opened or closed";
      'TEXT yet-to-come "dynamic behaviour";
      'END weigh-A
   'AND
      'BLOCK weigh-B
   'AND
      'BUFFER scales-reading
   'END liquid-supply
'END mixer-control-system.
```

Figure 3: A small sample specification

An object is defined by a section. A section starts with the section-header, and ends with the section-tail. The section-body in between may define an arbitrary number of texts, and of links to other objects (including their definitions). Since SPADES-L is non-procedural, the order of statements is not relevant (and is actually ignored by SPADES-T).

For easy implementation of SPADES-T, the language was given an LL(1)-syntax (which does not matter in most constructs, but makes the language rather clumsy at a few points).

Unlike all other specification languages of its kind, SPADES-L is formally defined by an Attribute-Grammar. As a side benefit, this grammar also covers the effects of entering information to the database, which is the most complicated operation of SPADES-T.

Another interesting part of the definition of SPADES-L is the mapping from SPADES-constructs in a programming language. This mapping provides a default- definition of what is expressed by a specification. (This allows the user to specify deviations from the standard.) When, for instance, an object of kind "TRIGGER" is defined, it has a certain dynamic behaviour (including coordination of competing accesses), if not stated otherwise.

The mapping could possibly be implemented, but that is not its primary purpose; we rather want to show that semi-formal languages can be defined as precisely as formal ones (except for an operational semantics which is excluded by definition).

The example in fig. 3 shows the style of SPADES-L. Note the distinction between texts (" ... ") and comments ((* ... *)). Texts are stored in the database, comments are not.

THE TOOL SPADES-T

SPADES-T is actually not just a tool, but a set of tools. Besides the user interface, which is described below, its components can be classified as follows: there are tools for modifying the specification currently stored (conversion and deconversion for entering and retrieving information), and tools for analyzing it (the report-programs); all these are controlled by an interactive program, the so called SPADES-Monitor (see fig. 4).

Since earlier versions of SPADES did not provide any editing facility, a standard editor must be used. For extending a specification, the analyst generates a new SPADES-L-file and then invokes the conversion program. For modifying an existing specification, affected parts are deconverted, i.e. they are transformed into a SPADES-L-representation and removed from the database. The SPADES-L-file is modified and converted again. When errors are found during conversion, a so called remainder file is automatically generated. This file contains those parts of the specification which do not comply with the context sensitive syntax of SPADES-L. (Note that the stored specification is part of the context.)

The conversion-program of SPADES-T is not directly controlled by the Attribute Grammar of SPADES-L, but by a slightly simplified table. Only some crucial parts unlikely to be changed are fully coded. Still, modifications of the language are very easy.

As mentioned above, the language definition covers the effect of conversion to the database. In the beginning, the database is empty except for some standard information (definition of truth-values and time-units). Then, every conversion contributes to the content of the database. Extensions are accepted if and only if they are consistent with their

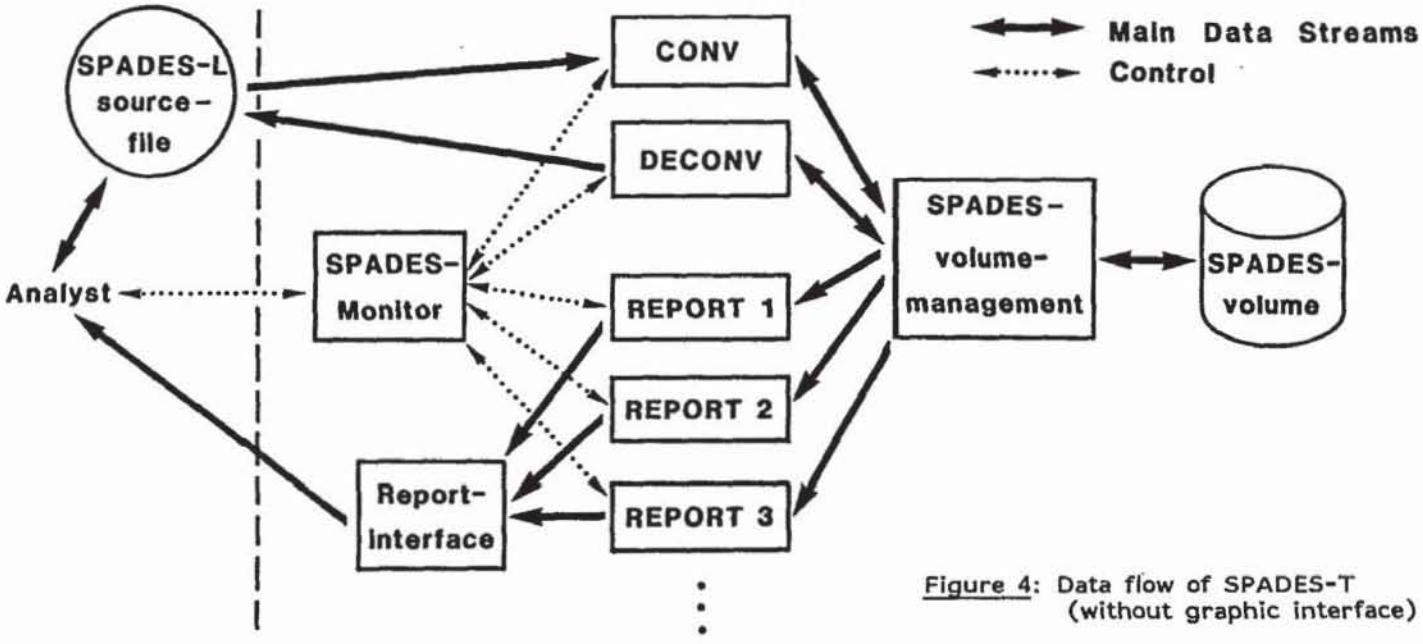

Figure 4: Data flow of SPADES-T (without graphic interface)

(left-)context, i.e. with any previously entered information, whether this information was entered a year before, or was contained in the same file. If information is entered twice, it is consistent, provided it was consistent the first time; therefore, repetition is legal in SPADES-L, and accepted (without effect on the database) by the conversion program.
When the specification is modified, the old version is kept; the analyst can always reset it to its previous state.

SPADES-T offers several operations for checking and evaluating the specification, each of which generates a report. There are currently six reports available:

- content (a read-only equivalent to deconversion)
- hierarchy of modules and informal objects (prints the tree, or trees)
- call structure (shows from where procedures are called)
- data flow (reports all data whose data flow is not complete, e.g. a variable which is never read)
- range check (checks accesses against scopes)
- completeness (indicates which objects have been referenced, but are not yet explicitly defined)

SPADES-T is implemented in MODULA-2 on VAX/VMS. Since all OS-dependent functions are well separated, implementation on another system should be easy (though it has not yet been tried).

THE USER INTERFACE OF SPADES-T

Drawbacks of specification systems (like tedious work on formal descriptions) are, overall, easily outweighed by their long-sighted benefits (less errors, easier implementation and maintenance). The behaviour of people as individuals working on rather short range goals, however, is not much influenced by those advantages. They will accept and support a tool only if there is an immediate profit. At least those who work in industry must accept this as a basic fact of life. Therefore, we must do our best to make users feel as comfortable as possible. The interface of SPADES, i.e. the style of communication between user and system, is obviously the field where we can win or loose our battle.

For controlling SPADES-T on a VT100-terminal, a simple but clear interface was implemented: Permanent (status) information is shown in reverse video at top of the screen. Commands are either selected from menues (at the right hand side), or entered from keyboard in the bottom line. The left centre area is used for messages from SPADES-T (see fig. 5). When textual information or graphics are displayed this window is temporarily extended.

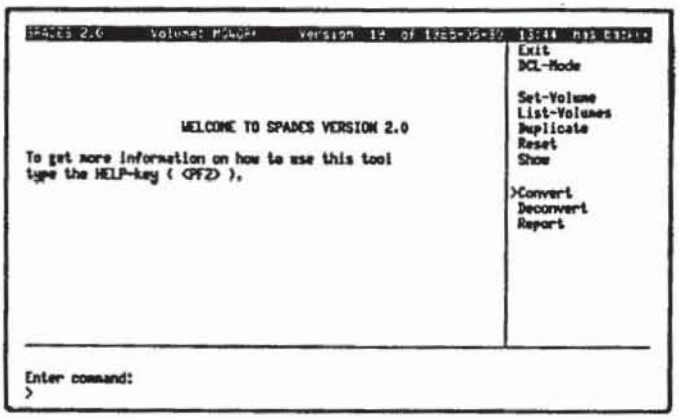

Figure 5: User interface of SPADES-T

Most people prefer graphics to text, provided the pictures are fairly well structured, and not too complex. The Entity Relationship Concept is widely used in specification systems because it reflects well our graphical understanding. Figure 2 is very similar to the drafts we draw when we explain something. As shown above, we can express the same information by a linear representation (SPADES-L), but the most natural style is graphics. Therefore, graphics is the key to improving acceptance. Even though many scientists will correctly object that just another representation of the same information is not an improvement, those who want to provide a tool which is not only good according to their own standards, but also successful, must take this point into account.

Since SPADES is based on the Entity Relationship Concept, any specification in SPADES-L can be easily represented by a graph. In practice, however, it is not that easy. There are three limiting factors:

- User's mental limitations
  The user in not able to handle large amounts of information at one time; this affects both the diversity of elements, i.e. the number of different icons, and the total volume of information.

- Properties of the hardware
  (screen, CPU, and transmission line)
  These limit size, resolution, and picture generation time.

- Implementation effort

With the VT100, graphics is not really possible. Therefore, we chose VT240 and VT241 as hardware for our human-computer-interface[10]. These terminals are compatible with VT100, but offer vector graphics as well (VT241 in color). A workstation with high resolution screen would certainly allow

for better (and much faster) graphics, but we should not develop a tool that cannot be used in other departments of our company. Therefore, VT240 is a cautious choice.

Even with an arbitrarily large screen, the analyst could not make use of very complex graphs; he or she must concentrate on a single point (object), from where links to other objects can be traced. This leads to our so called star-representation: Up to eight objects are arranged around a central object. An object is represented by an icon, which represents its kind, and by its name, written into the icon. Links are drawn as arcs between objects; every relation is represented by a special symbol (arrow, dotted line, etc.). Fig. 6, which is based on fig. 3, shows a star-representation centered at object "weigh-A". Note that (with the current implementation) only binary links are shown, the third component is discarded if present.

If there are more than eight objects linked to the star-object, seven of them are shown, and the fact that more objects are logically present is indicated by a message saying that the analyst may scroll.

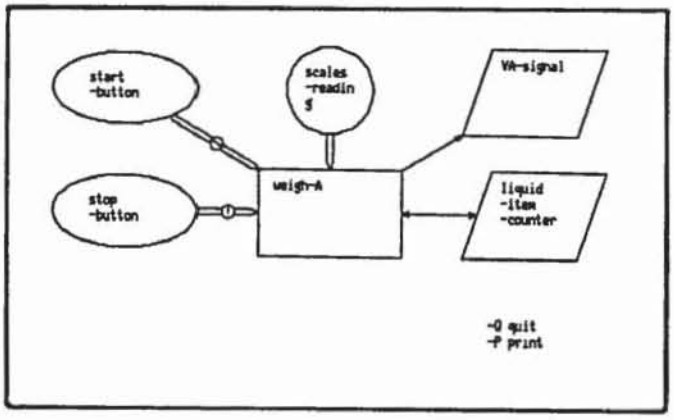

Figure 6: Object weigh-A of fig. 3 in star-representation

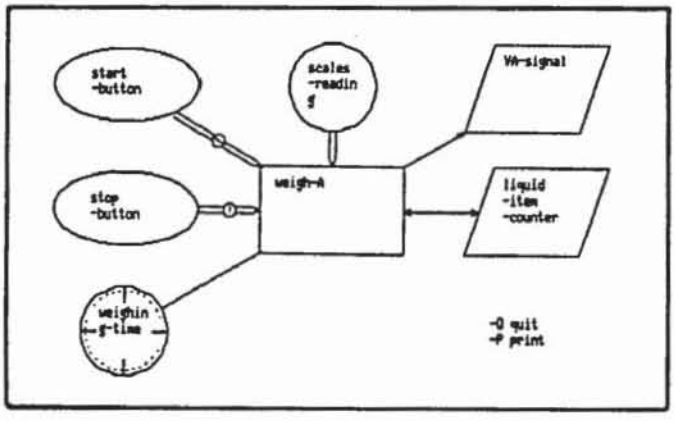

Figure 7: Extended star-representation from fig. 6

With the graphical interface, we will be able to allow for interactive editing instead of editing an excerpt in SPADES-L. Fig. 7 shows an extension of fig. 6, fig. 8 shows the equivalent fragment in SPADES-L.

```
'BLOCK weigh-A:
'TAKES-OF weighing-time;
'END weigh-A.
```

Figure 8: SPADES-L-equivalent to the extension in fig. 7

The star-type graphics shown above is well suited for displaying data flow and interaction, but not for hierarchical structure, as expressed by the COMPRISE-relation. For these, the tree-representation was developed. In a tree-type graphics, the root is the point of interest. It is displayed at top of the page. In the middle level, up to three off-springs of the root object are shown; nodes displayed in the bottom level are the offsprings of the central object (see fig. 9, which is based on the specification in fig. 3).

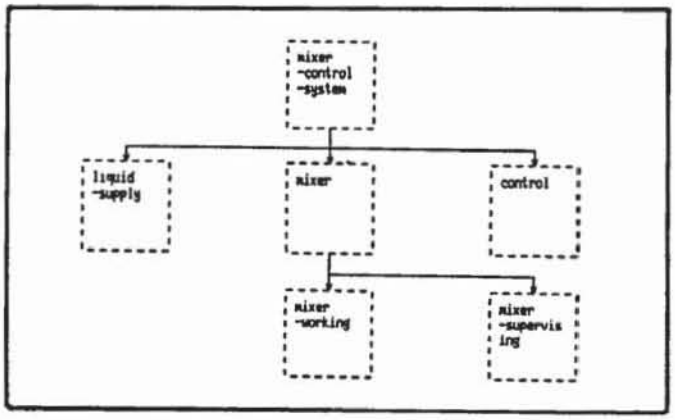

Figure 9: Tree-representation of module mixer-control-system

The principle of scrolling applies to the tree-graphics as well; when the first-generation subtree is scrolled, the lower line follows automatically.

Both in the star- and in the tree-graphics, browsing through the logical structure is achieved by selecting a new object of interest, and redrawing the picture accordingly. Selection would ideally be done with a pointing device (a mouse for instance), but we have none, so we use the cursor instead. The cursor is positioned with keys, moving from one object to another with every keystroke. See fig. 10 showing the result of moving the point of interest in fig. 9 from mixer-control-system to liquid-supply.

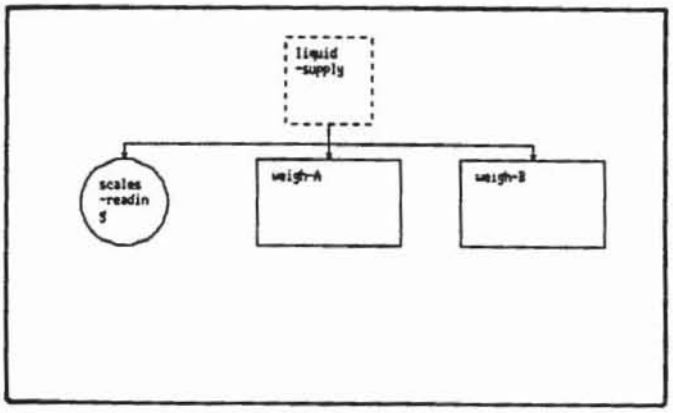

Figure 10: Owner-hierarchy of object liquid-supply

While working with the graphical interface, the analyst may access a help-subsystem, which provides simple hints on possible commands.

CURRENT STATE, PROBLEMS, CONCLUSIONS

Graphical display of specifications and browsing is now available, editing is currently being implemented. Since this implies modifications of the stored specification, all checks for maintaining consistency must be performed, which is not trivial. In order to emphasize particular aspects of specifications (for instance their dynamic structure), future extensions will allow for subsetting the kinds and relations to be displayed.

SPADES is currently being used in a pilot-project, i.e. outside the protection of its creators. Despite many difficulties in details, it seems to prove its value. The hardest problem, however, is acceptance. By providing a graphical interface, we hope to improve its quality slightly, but we expect to improve its acceptance dramatically.

It is one thing to write down a list of features which seem useful or even necessary, and another thing to implement them. Many extensions to SPADES are highly desirable, but are not, or not yet, realized due to obvious limitations of manpower. A list of such extensions, some of which have been investigated in some detail, would at least include

- a real multi-user interface (i.e. protection and locking mechanisms at the level of single objects and relationships)
- a facility for language extensions, particularly for languages dedicated to dialogue and protocol specification;
- a sophisticated version and variant control system;
- a simulation system for performance analysis (as available in SREM[3]);
- a transformation system for generating program-skeletons from specifications.

ACKNOWLEDGEMENTS

ESPRESO, the predecessor of SPADES, was developed at Nuclear Research Center in Karlsruhe, Federal Republic of Germany. Implementation of the graphical interface and proofreading of this paper was supported by Mark Garrett from Hatfield Polytechnic. Three anonymous referees contributed many helpful comments.

REFERENCES

[1] J. Ludewig, Process control software specification in PCSL. in V. Haase (ed.), IFAC/IFIP Workshop on Real Time Programming, Pergamon Press, Oxford etc., 1980, pp.103-108.

[2] D. Teichroew, E.A. Hershey III, PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems. IEEE Transactions on Software Engineering, SE-3 (1977), 41-48.

[3] M. Alford, SREM at the age of eigth: the distributed computing design system. IEEE COMPUTER, April 1985, 36-46.

[4] J. Ludewig, ESPRESO - a system for process control software specification. IEEE Transactions on Software Engineering, SE-9 (1983), 427-436.

[5] J. Ludewig, H. Matheis, M.F. Schmidt, SPADES-Manual (Language Reference Manual, Operator's Manual, Guide to the practical application of SPADES). Brown Boveri Research Center, Internal Report, 1985.

[6] M. Glinz, H.J. Huser, J. Ludewig, SEED - A database system for software engineering environments. in Blaser, Pistor (Hrsg.): Datenbanksysteme für Büro, Technik und Wissenschaft, Informatik-FB 94, Springer, 1985, pp.121-126.

[7] P.P.-S. Chen, The Entity-Relationship Model - toward a unified view of data. ACM Transactions on Data Base Systems, 1 (1976), 1, 9-36.

[8] J. Ludewig, Computer aided specification of process control software. IEEE COMPUTER, May 1982, 12-20.

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

[10] G. Matheis, Konzeption und Realisierung der graphischen Ausgabe von Spezifikationen. Master Thesis, University of Kaiserslautern, 1985.