©PTT/Dr. Neher Laboratories, 1987

DASOM - A SDL-TOOL

BY EIRIK VEFSNMO

A.S COMPUTAS
P.O. Box 3765, Granaaslia
N-7001 Trondheim
NORWAY

+ 47 7 917500

ABSTRACT

DASOM is a software engineering tool supporting the SDL-language. This tool support especially the specification- and design phases, but it also have automatic program generation, document treatment, archive facilities, version management and other features giving support to the whole life-cycle of systems. DASOM is now available as a commercial product and is in practical use in several industrial companies and research institutions. This paper will in addition to the description of the tool itself, report some of the experiences gained from real-life use of the tool.

1. DASOM - A DESCRIPTION OF THE TOOL

DASOM is a software system which gives computer support to project teams using SDL in specification, design, implementation, production and maintenance of systems. DASOM assists project workers during several phases in a project and is hence a part of computer aided engineering (CAE) for system development and maintenance.

1.1 DESIGN AND IMPLEMENTATION OF DASOM

The object-oriented concept

In the design of DASOM we have adopted some concepts similiar to those used in object-oriented systems (Smalltalk-80, Simula, etc.). A brief description of the most important concepts will be given in the following.

All components in DASOM are represented as objects. An object may for instance be a system model, a page, a state, a block, etc. An object is manipulated by operations. Operations may for instance be create, delete, enter, leave, etc.

In DASOM a certain set of object-types are predefined. Each object type may have one or several instances. An attribute of an object is either an element with a defined value set or format type, or it represents a new object or object set, i.e. representing a structure of objects. For more details, see reference [5].

Implementation aspects

DASOM is based on support systems like a relational database system (MIMER) and a graphical system (GPGS-F). These support systems are accessed through specialized interface-modules making it easy to eventually replace them. It is futher implemented by the use of an object oriented task scheduling system (see reference [8]). This implementation strategy is making DASOM very flexible with respect to extensions and portability.

Today DASOM is running on all types of VAX-computers from Digital Equipment Corporation (DEC), ranging from MicroVAX-II, via the VAX-11 series to the VAX 8000- series. The operating system VMS is used, and may even be accessed directly from DASOM's user dialogue if you want to perform VMS-calls while you are running DASOM. On the other hand DASOM depends very little on the operating system, it is mainly used for I/O purposes.

DASOM may use a lot of graphical terminals and printers/plotters/laser printers. There exist software drivers for the most common used low cost graphical terminals (e.g. Tektronix, Hewlett-Packard, etc.). However, a resolution of about 800 x 600 pixels seems to be necessary to optain satisfactory graphical pictures.

Figure 1 shows an overview of the different modules in DASOM.

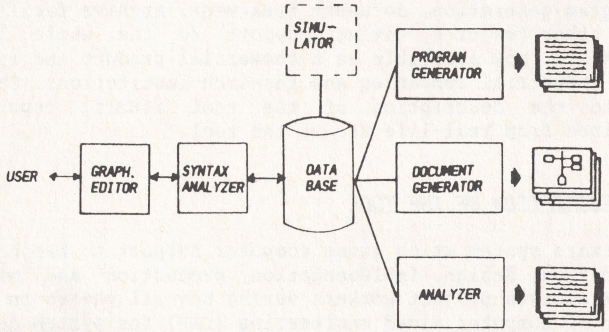

Figure 1: System overview of DASOM

1.2 FUNCTIONS IN DASOM

The functions which DASOM cover are:

* Editing of different diagram types
  (sequence charts, block-interaction- and process diagrams)
* Functions for project organization
* Document generation and treatment
* Model- and document archives
* Syntax- and (consistency) checking
* Version management
* Program generation

These functions are described in more details in the references [5], [6], and [7] and will therefore not be outlined in more details here.

1.3 USER DIALOGUE

Special attention has been paid to make the user-dialogue easy and efficient. It contains both graphical- and alphanumeric parts. It is object-oriented, and gives the user the impression of communicating with objects that represent models and documents. The objects are organized in a hierarchical structure where the attributes of objects are either terminal attributes, which may be given values, or new objects. In this fashion the dialogue is presented as productions in a way much similiar to those used in formal grammars (see references [6] and [7]). The main levels of the object hierarchy are shown on figure 2.

The user may navigate in the hierarchy of objects by using the functional keypad of the terminal. Operations on the objects are initiated by using the same functional keypad. The functional keys are assigned different operations which may be performed on the objects. Object values are given via the alphanumeric keyboard.

The graphical part uses to some extent automatic layout, but for the most cases the users are free to place the objects where they want on the screen (by the aid of coordinates or cross-bar).

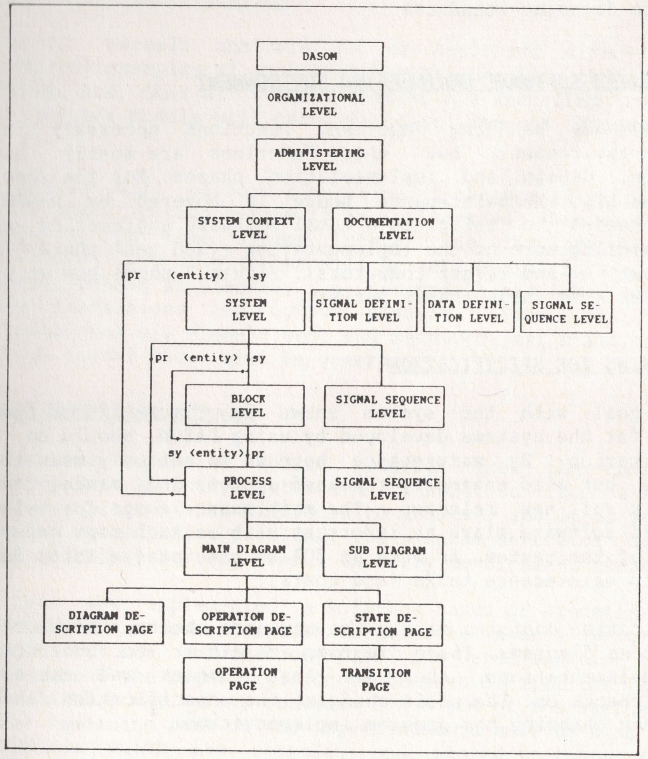

Figure 2: Different levels in the object hierarchy.

1.4 DASOM's COMPLIANCE WITH SDL

DASOM supports the main -, and most commonly used aspects of the SDL-language as defined in SDL '84 (reference [2]).

However, DASOM contains restrictions in the use of SDL (as defined in reference [2], i.e. SDL '84). For the process diagrams there are a few symbols not yet implemented, but these symbols are planned to be included. The pictorial elements option is not included. In the block interaction diagrams we do not allow subprocesses, and it is only allowed with one process per block. The block-tree diagrams are not used. The channels are extended to allow for bidirectionality. SDL/PR are not used or supported at all, nor is the data in SDL (recommendation Z.104 in reference [2]).

DASOM also contains extensions compared to SDL. The extensions are mainly in the block interaction diagrams where DASOM offers more flexibility in the use of channels. Futher the blocks may be assigned different roles which give them different shapes on the diagrams. DASOM has very good facilities for supporting sequence charts, which are very much used during early stages of a development.

Other aspects concerned with methodology, not mentioned at all in SDL, but still necessary for all practical purposes, are supported by DASOM. Examples of such aspects are project organization (project workers, responsibility areas) and document treatment (archives, generation and distribution). These parts of DASOM may be adopted to fullfill existing standards within a company (for instance document standars).

2. AN INTEGRATED SOFTWARE ENGINEERING ENVIRONMENT

DASOM is already handling important functions necessary in a software engineering environment, but these functions are mostly used for the specification, design and implementation phases. For the specification and documentation also the maintenance phase is covered by DASOM today. The extensions needed to integrate a total software engineering environment are functions covering more of the implementation - and test phases (e.g. testing at both host - and target computers). Figure 3 shows how we plan to extend DASOM to cover also these phases better.

2.1 MAINTAINING THE SPECIFICATION

The basic goal with the system shown in figure 2 is that most of the maintenance for the systems developed by using DASOM, should be performed on the specification. By maintenance here we do not only mean the activity to correct bugs, but also enhancing released systems and making changes in the functionality of new releases. The maintenance costs for telecommunication systems where software plays an important part is much more expensive than the development of the system. As much as 80% of the total efforts in a system may be related to maintenance tasks (and costs).

The specification contains mainly the sequence charts, the block interaction- and the process diagrams. These diagrams are easier to understand than the program implementations, they are less complex and easier to perform consistency checks on. Therefore changing the specification should generate less bugs than changing the program implementation.

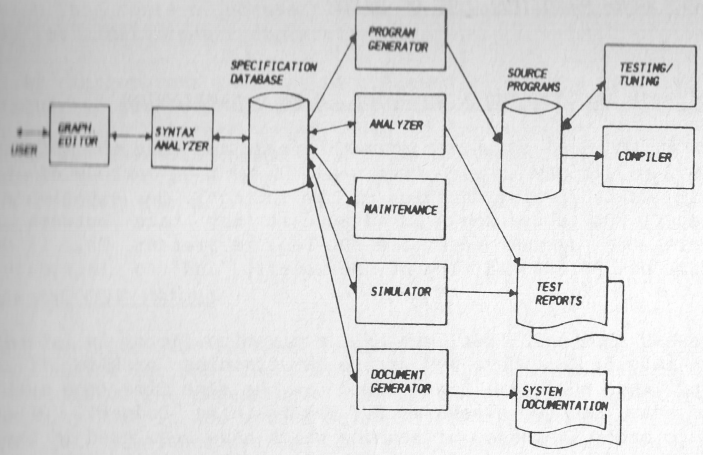

Figure 3: Future DASOM software engineering environment.

2.2 ANALYZE/SIMULATE THE SPECIFICATION

DASOM will allow certain analysis to be performed directly on the specification. Typical examples of such analysis are checking the completeness of process diagrams, i.e. that all states referred are specified, no deadlocks exist, etc. The analyzer module will generate test reports documenting the analysis.

The simulator module will perform simulation of the specification. It will actually "run" the specification before it is implemented. This is possible to perform by using a run time system interpreting the information in the specification database. Such simulations will be very useful in verifying that a certain process behave as planned, i.e. respond to the input signals with the right state transitions and output signals. From the simulator documentation containing all signals sent and received, all state transitions performed per input signal etc., will be generated.

2.3 AUTOMATED IMPLEMENTATION SUPPORT

The program generator generates source programs from the data in the specification database. A fully automated translation from the process diagrams to source programs is difficult to achieve, and it is perhaps not wanted at all due to optimization problems. The program generator allows for rapid prototyping which may be very useful in early development phases.

In DASOM, we have now implemented a solution based on predefined program skeletons which the program generator uses to generate source programs. The program generator fetches information from the specification database, interpret it, and choose the right skeleton to be used to fill the translated source code into. In this way we may generate 50% - 60% of the source programs automatically. The remaining part of the implementation must be done manually by coding procedures which are compiled and stored in libraries. During linking, these procedures are fetched from the libraries. See reference [9] for a complete description of the program generation facilities in DASOM.

3. EXPERIENCES FROM PRACTICAL USE OF DASOM

3.1 PROBLEMS WITH INTRODUCING A SDL-TOOL IN AN ORGANIZATION

Introducing a SDL-tool in an organization requires that the SDL-language is known and perhaps already in use. Otherwise SDL must be introduced before or in parallell with the introduction of the SDL-tool. Our experiences in this field is that if SDL is not known in advance it may take between 6 to 12 months before the actual need of a SDL-tool is present. This is due to the education-time needed for training of personell, and to introduce SDL in projects.

Futher it seems that a SDL-tool may be a motivation factor in introducing the SDL-language itself. But there may arise a training problem if both the language and the tool are introduced at the same time (too much delay in existing projects, and too expensive for the training budget). Often people relate SDL to other languages or methods which have been used in the company. They must therefore be convinced that SDL is better than already existing languages/methods.

3.2 USER ASPECTS

New tool users are very interested in trying the tool, consequently it is easy to motivate them to use it. It should, however, be a requirement that each user has access to the tool from his/her own terminal; and the situation with a few workstations should be avoided.

The user interface of the tool is very important. The user-dialogue must be easy to understand and at the same time be efficient in order to encourage the users.

The user's project situation does also play an important role in the introduction of the tool. If the users can benefit from the tool in their daily work, i.e. in the project they are engaged in, this will make the use of the tool more meaningful and at the same time they gain experience in practical use.

3.3 IMPROVEMENTS BY USING DASOM

The industrial use of DASOM so far has been in the development of intercom systems, queue- and distribution exchanges and aiport communication systems. These are all well suited application areas for SDL, and the use of DASOM should therefore give immediate benefits.

We have not made any scientific measurements, but we have reports (and own experiences) from the use of DASOM showing considerably productivity increase compared to manual use of SDL. The following estimates on reductions in the efforts by using DASOM have been experienced:

- Reduction in diagram production : 10 - 20 %
- Reduction in maintaining diagrams : 30 - 50 %
- Reduction in variant production : 60 - 70 %

Reuse of models/diagrams has not yet been evaluated, but is of course considerably. The above mentioned values are in good accordance with what was expected by DASOM (in advance).

In addition the costs of document production has been significantly reduced, especially for final product documentation.

The quality improvments that may be achieved by using a tool like DASOM, are very difficult to quantify. We have no figures for these improvements, though we know it exist. Perhaps this kind of improvements first of all will be achieved through the maintenance of the systems specified and design by the aid of DASOM. This because less errors will appear in the operation phase when errors are extremly expensive to correct, because the system is actually in use already.

4. SUMMARY AND CONCLUSIONS

This paper describes the software engineering tool DASOM supporting SDL and a methodology suited for SDL. The design and implementation aspects of DASOM are briefly described. The experiences from practical use of DASOM shows increase in productivity both in new system developments, in variant production and for maintenance of developed systems. The resulting products have higher quality (i.e. have less bugs and are more reliable) than those developed without DASOM.

DASOM is now under futher development in order to cover all aspects of a total software engineering environment where SDL will be a central component. This development process is outlined in the paper.

In order to make SDL more available, and to make it more attractive to use, we need computer aided tools like DASOM to support it. We believe that the tool must be available for all project participants in order to obtain full benefits of SDL in different phases of the system development. Our experiences so far shows that the expected improvments will be achieved. But introducing a SDL-tool in an organization may take time and should be well planned and organized. Necessary training resources are also essential for a successful introduction and use of the tool (DASOM) and the language (SDL). A certain application methodology should be introduced together with SDL in order to take care of practical matters in the projects.

REFERENCES

[1] R. Bræk, O. Helle, F. Sandvik: "SOM - A SDL Compatible Specification and Design Methodology", 4th International Conference on Software Engineering for Telecom. Switching Systems, Coventry, 20-24 July 1981

[2] CCITT Red Book, Geneva 1985: Recommendations Z.100 - Z.104.

[3] A. Rockstrøm, R. Saracco: "SDL - CCITT Specification and Description Language", IEEE Transaction on Communication, June 1982.

[4] E. Vefsnmo: "DASOM - A Step Towards Higher Quality at lower cost", NT-Symposium on Languages and Methods for Telecom. Applications, Turku, Finland 6-8 March 1984.

[5] E. Vefsnmo: "DASOM - An Advanced Computer Aided Tool Supporting A SDL-Oriented Methodology.", Second SDL Users and Implementors Forum, Helsinki, Finland 11-15 March 1985.

[6] E. Vefsnmo: "DASOM - A Software Engineering Tool for Communication Application Increasing Productivity and Software Quality.", 8th International Conference on Software Engineering, London, England 28-30 August 1985.

[7] E. Vefsnmo: "DASOM - An Integrated Software Engineering Tool for Telecommunication Software Systems". 6th International Conference on Software Engineering for Telecommunication Switchins Systems, Eindhoven, The Netherlands, 14-18 April, 1986.

[8] U. Johansen: "RUNSYS - An Object Oriented Task Scheduling System", ELAB-report STF44F87006, January 1987.

[9] U. Johansen, E. Vefsnmo: "Automatic Program Generation of SDL-Specifications: Principles ans Solutions", Third SDL Forum, The Hague, The Netherlands, 6-10 April 1987.