Plessow, M.; Simeonov, Plamen L. :

Netlike Schematics and their Structure Description

Zuerst erschienen in:
Workshop on Informatics in Industrial Automation, Berlin, GDR,
Oktober 31 – November 4, 1989, S. 144-163

This book contains papers to the following subjects:

- Control of Production Systems
- Analysis and Modelling Methods
- Computer Aided Design Processes, Including Process Planning and Simulation
- Aspects of CIM and Artificial Intelligence

EDITED BY
G. MENGA V. KEMPE

WORKSHOP ON INFORMATICS IN INDUSTRIAL AUTOMATION

Berlin, GDR — October 31–November 4, 1989

NETLIKE SCHEMATICS AND THEIR STRUCTURE DESCRIPTION

M. Plessow , P. Simeonov
Academy of Sciences of the GDR
Central Institute of Cybernetics
and Information Processes (CICIP)
Kurstr. 33, POB 1298
DDR-1086 Berlin, GDR

Abstract

Engineering charts or schematics are essential for system design. They include not only graphical information (coordinates and spatial location of points, lines and curves), but also describe structural relationships between represented objects.

Netlike schematics, such as electrical drawings, flow charts and diagrams comprise in the main part information about the structure of the device.

In this paper we try to answer the question of how to use and represent structural information in a general system for schematics engineering, a so-called Computer Aided Schematics (CAS) System /1/.

From the schematics viewpoint we introduce a Schematic Structure Description Language (SSDL) and the way to use it in the CAS - System.

INTRODUCTION

Graphical means are an essential part of human communication. It is difficult to imagine the documentation of a complex system without graphics. The design of large scale systems, which are often electronic devices, demands a great deal of graphical support. It is often not possible to understand such a system without the help of charts, flow graphs or diagrams.

A special class of graphical utilities is this one of netlike schematics. They are very useful for the representation of compound objects and processes. Netlike schematics consist of symbols which are inter-connected by lines (nets). The symbols represent elements of the system and the nets illustrate relations between the elements. In this way it is possible to describe both structural and functional contents.

Netlike schematics differ from other "statistical" 2D-graphics such as business graphics (bar charts, pies, etc.) and workshop drawings.

Some typical examples of netlike schematics are electrical diagrams, technological layouts, Petri nets, graphs.

The Computer Aided Schematics (CAS) System is an effective tool for design and implementation of netlike schematics /1/. It is intended to provide at first, the typical graphical support for a CAD-System, namely the graphical representation of the object as well as additional graphical facilities for automatic or interactive layout such as partitioning, placement and routing techniques.

At second, the system should allow the generation, modification and communication of structural information about the object / process considered as well as definition and realization of suitable interfaces.

Our intention is to discuss the problems of structure description and interfacing used in a CAS-System.

1. WHAT ARE NETLIKE SCHEMATICS ?

Let us have an example to illustrate what a netlike schematic is.

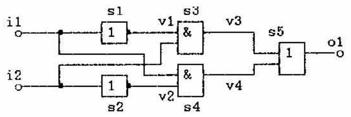

Fig. 1 : A netlike schematic representing the structure of the logical exclusive-or (EXOR) element

Figure 1 depicts a simple electrical drawing of a logical element which realizes the Boolean function exclusive-or. We have 5 items (symbols, components) on that chart. Each of them is regarded as a Boolean function (process) or as a part of an integrated circuit (object).

Every item is denoted by an identifier (symbol name . S1,..,S5) and a type specification (symbol type : '&', '1'). Let us assume that we do not have information about the internal structure of the components. That is to say they have "elementary" types.

Relationships between the symbols are shown by nets (connections). Nets are treelike branched lines. Sometimes they represent a special kind of relation, namely directed nets (for example in a directed graph).

A netlike schematic is said to be build up of symbols and connections.

Every symbol has one or more special points, the so-called pins. Only at these points it is possible to connect a net with a symbol. In the example on figure 1 the left-hand pins of the symbols represent arguments and the right-hand pins represent values of the functions. Because of this it is reasonable to characterize these pins with the attributes Input and Output. It is also useful to have a name or a number as identifier of the symbol pin.

There are some net parts (branches), such as i1, i2 and o1 in our example, which are not connected to a symbol pin. They have a special meaning. Such net parts are called "borderpins", since they are located on the schematic border. It is possible to describe the realtionship between a netlike schematic and its surroundings with the help of borderpins. It is also possible to define directions for borderpins.

By changing our viewpoint to the schematic the special characteristic of the borderpins becomes apparent. So far we spoke only of the internal schematic view. In much the same manner it must be possible to obtain a structure as a part of another one. In this instance a new symbol (Fig. 2) should be introduced. It is said that this symbol represents the old structure from an external point of view. Then, instead of old structure we speak of substructure (subschematic) or symboltype. To describe relations inside the new structure we identify the borderpins of the substructure with the symbolpins of the new symbol. In this way it is possible to construct a hierarchy of structures of the process / object described. That means that every substructure represents a different level of hierarchy.

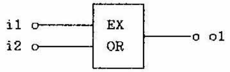

Fig. 2 : A symbolic representation of the logical element EXOR

Resuming, we can say that netlike schematics are defined by symbols and relations between them which on their part are represented by nets. Symbol pins and borderpins are the only elements within a schematic to be connected. By using a symbol for a netlike schematic and identifying borderpins by symbolpins we can get an external view to the schematic. Such an external view may be a symbol in another schematic on a higher level. The symboltype refers to the corresponding internal view (subschematic) of the symbol.

2. THE SCHEMATIC STRUCTURE DESCRIPTION LANGUAGE

2.1 DESCRIPTION

Following problems arise when netlike schematics are used for description of objects or processes:

1. How can be guaranteed that the schematic reflects the particular process or object description?

2. Where does necessary information on schematic construction come from?

3. How can be guaranteed that data in design process remain consistent if structural, i.e. nongeometrical changes in the schematic (resp. in the process) are performed ?

4. How can schematic information be used for otner processes ?

To answer these questions, we developed a language , the Schematic Structure Description Language (SSDL) /6/, which is able to represent structural information of netlike schematics without regard to the particular geometrical realization.

A prototype of the language was the frequently used in the GDR NBS 84 /2/ . Important attributes of the SSDL are said to be its description form variability and simplicity. To fit these requirements, an appropriate description form for each essential part of the netlike schematic was proposed.

Connections (nets), symbols (objects, processes) and types (substructures) are described respectively in terms of net, symbol and type blocks.

Nets are to be declared in a net block. That means an attribute such as Net (default), Input, Output or Unknown can be assigned to each connection. The Bus - attribute allows structuring of connections by collecting nets of the same type in a bus statement. Here, the analogy with busses in electrical diagrams becomes apparent.

Nets, specified as Input, Output or Unknown play a certain part in the structure description of netlike schematics: they represent connections which have a special meaning outside the particular structural level. Such nets are said to have borderpins in the graphical representation. The semantics demands that the enumeration of borderpins take place along with the succession of net declarations.

The symbol block is used for the purpose of symbol declaration. It consists of a sequence of statements assigning appropriate types to the symbols which are identified by their names. This is the way in which schematics hierarchy is addressed. Net and symbol blocks contain information about component relationships at the actual level of resolution.

Type blocks set up another level of resolution. They serve the description of substructures (subschematics) or elementary symbols. Each type consists of one or more net blocks and none or more symbol blocks. Statements about borderpins within a net block virtually "hook" the substructure onto the higher level of representation. This is how low-level references are accomodated to high-level ones. In case that a type block describes an elementary symbol (subschematic), it does not contain symbol and type blocks. That means we can get only an external view of this subschematic. A structural description of an object / process presumes that all type structures (subschematics) be reduced in elementary units at some level of representation.
The number of resolution levels which can be described in this way is virtually unlimited.

SSDL is a visual language /15/ : it was developed for the handling of visual information.

According to the classification given by German and Lieberherr /7/ SSDL belongs to the pure structural class of hardware description languages (HDLs), such as HISDL /8/, BDL /9/ and HIDEL /10/. That is why only structural specifications are allowed. Information about geometry, functional behaviour and data control are not taken into account.

SSDL is a "non-executive" language. It supports neither conditional nor executive statements (except for one), nor parametrically defined classes of design objects.

By these features SSDL could be considered as a formal graph representation language rather than as a typical HDL. It is required, that the designer plainly "list" structure items along with their relations, namely SYmbols (nodes), NEts (connections) and symbol TYpes (substructures) which describe essentially in much the same manner another level of structural hierarchy. However, once a particular description is completed, it can be involved into another one : the INclude-statement provides an additional facility for extending the structural hierarchy (nested descriptions).

SSDL might be considered as a 2D-network-representation language which is related to a special class of engineering charts, sketches, diagrams, expression trees, flow graphs and tables.

In the following section we shall discuss some characteristic language aspects.

2.2. A DESIGN EXAMPLE

Our objective was to design a comfortable well-structured description language. Its modularity becomes apparent in terms of itemized blocks appearing in an arbitrary order. It was our intent to enable the user to partition his or her description as he or she likes. The only thing that should not be forgotten is keeping track of completeness and hierarchy references. That is why we emphasized on semantics in SSDL

Let us consider the following net block description (Fig.3):

```
NE
   a, b, c :              = I ;
   d       : s1#3, s2#1     ;
   e         s2#3         = O ;
EN ;
```

It is essential, that object declarations be associated with their attributes. In our case net declarations are followed by optional symbol pin and net type assignments.

Existing descriptions may be extended within the current or any other itemized block at the same structural level just by adding new statements.

For that reason information about data objects collected by the early phase of the Structure Description (SD) Compiler (Fig. 5) is continuously verified and updated.

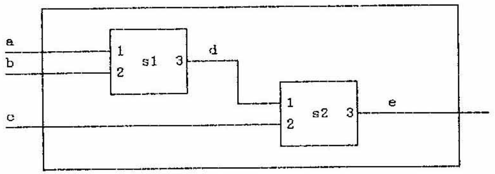

Fig. 3 : A Network Example

In our example net 'd' has the default type Net (if not otherwise specified) and connects pin #3 of symbol 's1' with pin #1 of symbol 's2'. 'e' is specified as an output net connected to pin #3 of symbol 's2'.

In order to complete our net description three additional specifications, namely for the nets 'a', 'b' and 'c', are required. Which node are they linked to ? The user has the choice between following two options :

```
1.    a : s1#1 ;                ( within the current or any
      b : s1#2 ;                  other net block at the
      c : s2#2 ;                  same structural level    ;

                     and

2.    s1 : a - b - d = type_1;   ( within  a symbol block
      s2 : d - c - e = type_2;     at the same structural
                                   level                 )
```

The net sequence in a single symbol statement (choice 2) denotes the implicite assignments to the symbol pins :

```
          net 'a' --> symbol s1, pin #1
          net 'b' --> symbol s1, pin #2
          net 'd' --> symbol s1, pin #3      etc.
```

This sequence may be omitted in case that equivalent assignments within a net block already exist.

Note, that some information redundancy is not excluded : in choice 2. net 'd' is defined for the second time. A skilled designer would replace 'd' with an asteriks, indicating by this that the actual net has been described somewhere else, namely in the previous net block. However, multiple definitions should not cause any troubles, unless they are ambiguous. This consideration is reflected also in the SSDL semantics.
Our structure is completely described, provided that there are no hierarchy references (compound TYPEs) within the symbol block.

Here is the entire structure description consistent with the example in Fig. 3 :

```
          MS : example_v2

             SY
                s1 : a - b - * = exor ;
                s2 : * - c - e = exor ;
             EN ;

             NE
                a, b, c                = I ;
                d        : s1#3, s2#1     ;
                e        :             = O ;
             EN ;

          EN.
```

Let us assume now that the symbol type identifier "exor" specifies a substructure. Then, we are supposed to provide a type block describing the internal structure of the "exor" element (Fig.1). Here is the contents of the required type block according to figure 1 :

```
        TY: exor,                { COMMENT :  description located }
                                 { in external file "exor.dat"    }
            NE: i1,i2:=I;
                o1:=O;
            EN;

            SY
               s1:i1-v1    =neg;
               s2:i2-v2    =neg;
            EN;

            SY
               s3:v1-i2-v3=and;
               s4:i1-v2-v4=and;
               s5:v3-v4-o1=or;
            EN;

            TY: neg;
                NE in:=I; out:=O EN;
            EN;

            TY: and;
                NE in1,in2:=I;out:=O EN;
            EN;

            TY: or;
                NE in1,in2:=I;out:=O EN;
            EN;

        EN .
```

At this level of representation elementary types are "neg", "and" and "or". As we can see, these types belong to the last level of structural resolution : they do not include symbol and type blocks

A type block may be tacked on to the actual level of representation or saved in an external file, provided that an INclude-statement is placed somewhere within that level.

3. THE SSDL PLACE WITHIN THE CAS - SYSTEM

The SSD-Language was developed as a user-friendly interface. Its primary purpose was to afford the opportunity of describing network contents by means of a general abstraction formalism at the logical design stage without constraints both on scale (single level representation) and depth (hierarchy). A SSDL - description can be generated for example by a simulation / verification program or by the designer himself.

With a formal SSDL-description of an object / process being provided, an appropriate schematic can be generated by filling in the graphical data which are required.

For this reason, the description mentioned should be converted into CASI-DS, the Computer Aided Schematic Interchange Data Structure. This is realized by a geometrical-structural interface in the CAS-System, the CASI-File.

The CASI-File form represents a copy of the data structure used in the CASI-DS. It is a LISP-like list of network items along with their attributes. Every level of structural hierarchy in the CASI-F form is represented by a sequence of statements enclosed in paranthesis.

The CAS-philosophy presumes that the CASI Data Structure set up the communication kernel of the system intended to conduct the data flow between the separate internal processes and the outer world. A CASI schematics description is both input and target representation. That is why CASI-DS combines pure structural description and geometrical information /6/

Both CASI-File and SSDL are entry points of the CAS-system (Fig. 4).

An important feature of the CASI-File is that the data acquisition for the schematic description can be performed at any time during the schematic processing. Depending on design level the missing geometrical information is gradually completed or modified. This can be done automatically or interactively

Finally, a complete geometrical and structural description of the schematics is available. It can be visualized by the graphical components integrated into the CAS-System.

On the other hand, if structural changes are performed by the user of the CAS-System, they should be reflected in the corresponding new versions of the CASI-DS and CASI-F forms. These structural modifications must supply a new version of the SSDL-description of the object / process.

Hereby arises the necessity to develop a compiler / decompiler able to translate structural descriptions in both directions, namely SSDL -> CASI-F and CASI-F -> SSDL.

In this way the consistency between structural description of an object / process and its graphical representation as a netlike schematic can be guaranteed. The use of structural information with a view of other applications is also possible.

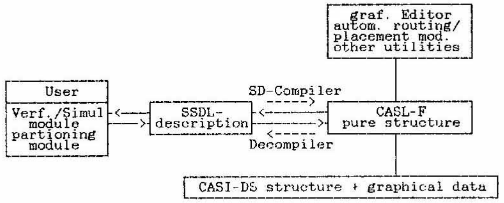

Fig. 4 : The SSDL - CASI functional organisation

SDC, the Structural Description Compiler, takes as input an SSDL-description and provides as output an equivalent CASI-File, which is the base for the CASI-DS.

The SD-Decompiler extracts the structural information from a CASI-File and generates its SSDL formal description.

And now let us pay some attention to the compiler structure.

4. THE SD-COMPILER

The network contents is supplied to the CAS-system in the form of an SSDL-description. An equivalent 'spread out' CASI-File is to be generated by the SD-Compiler. Fig. 5 shows the single phases of the compilation process.

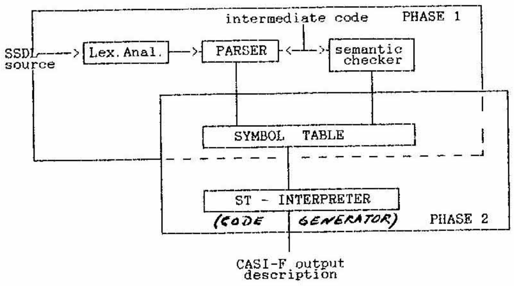

Fig. 5 : Structure of the SD-Compiler/Decompiler

It is composed of two phases : first, every single SSDL-statement within a block structure is evaluated and booked down as one or more entries into the symbol table every time a new identifier or new information about an existing name is encountered, and second, the symbol table contents is interpreted into a CASI-File.

Two features are distinctive for the SD-Compiler : the Symbol Table, set up as a network of pointer-referenced linear lists and binary trees keeping the actual information about the names in the source description (Fig. 6), and the ST-interpreter which scans the symbol table structural representation into a CASI-F description.

The SD-Compiler/Decompiler is written in C /13/ using a PC-implementation of the standard UNIX tools LEX /11/ and

5. SYSTEM OUTLOOKS

It seems quite appropriate to integrate a lexical and syntax corrector module into the SD-compiler structure in Fig. 5. An equivalent EDIF-representation /4/, /5/ for the purpose of communication with other independent CAD-system components might be also created during the second phase of compilation.
Another suggestion is to generate a graphical representation of the SSDL-description in order to verify what has been entered into the system /14/.

A further extension or even redefinition of the SSD-Language in view of gaining a more comprehensive notation of the scheme topology is not excluded.

Following features should be taken into consideration :

- explicite description of borderpins
  (the implicite one using special net types does not seem to be notably expedient)

- convention about notation of symbols and connections within symbol types (substructures) when used by other symbols

- an appropriate notation for simplified description of regular (resp. repetitive) structures

A further development of the SSDL as an object-oriented language seems to be quite reasonable /16/.

6. SUMMARY

The CAS-System and the SSD-Language are appropriate tools for the design of complex systems which can be represented by netlike schematics. They allow a detailed structure description of the process / object to design. This structure description may be considered from different points of view to the object. It can be used for the purpose of documentation, or , as we discuss in this issue, as a design utility. In this instance a new interpretation of the objects described, namely symbols and nets, is needed.
A typical application is the structure description of an electronic device. During the design process, an SSDL-description comprising the necessary basic information about the circuit structure or other important features (e.g. for the purpose of simulation) is built up.
In this sense the use of the SSDL guarantees the consistency of the design data and increases the effectivity of interaction.

7. REFERENCES

/1/ May,M.;Doering,S.;Thiede,F.;Vigerske,W.:
Schematics Engineering In Design Automation
CAA 88 Conf.,Torino,Nov.2-4, 1988

/2/ Netzbeschreibungssprache NBS - 84.
AdW der DDR,Karl-Weierstrass-Institut fuer Mathematik,
Berlin,1985.

/3/ Rueckle,G.: CBDL/1 - Eine Beschreibungssprache fuer Flachleiterplatten. Forschungsbericht Nr. 107 der Technischen Hochschule Darmstadt,Darmstadt,1981.

/4/ EDIF Electronic Design Interchange Format, V110.
EDIF Steering Committee, 1985

/5/ Ungerer,M.M. (ed.): CAD-Schnittstellen und Datentransferformate im Elektronik-Bereich.
Springer Verlag,1987.

/6/ Entwicklung eines Computer-Aided Schematics System.
Forschungsbericht , AdW der DDR, ZKI,
Berlin 14.11.1988.

/7/ German,S.; Lieberherr,K.: Zeus : A Language for Expressing Algorithms in Hardware. Computer, Vol. 18, No.2, Feb. 1985, 55 - 65.

/8/ Lim,W. Y-P.: HISDL - A Structure Description Language.
Comm. ACM, Vol. 25, No.11, Nov. 1982, 823-830.

/9/ Slutz,E.; Okita,G.; Wiseman,J.: Block Description Language (BDL) : A Structural Description Language. Proc. 21st ACM/IEEE Design Automation Conference, 1984, 81-85.

/10/ Ancona,M.; Clematis,A.; de Floriani,L.; Puppo,E.: A Hardware Description Language Based on a Hierarchical Graph Model. Microprocessing and Microprogramming, Vol.20, 1987,183-188.

/11/ Lesk,M.E.: LEX : A lexical Analyser Generator.
CSTR 39, Bell Laboratories, Murray Hill, N.J., 1973.

/12/ Johnson,S.C.: YACC : Yet Another Compiler Compiler.
CSTR 32, Bell Laboratories, Murray Hill, N.J.

/13/ Ritchie,D.M.; Kernighan,B.W.: The C Programming Language.
Prentice-Hall, Englewood-Cliffs, N.J., 1978

/14/ Kernighan,B.W.: PIC - A Language for Typesetting Graphics. Software - Practice and Experience, Vol. 12, 1982, 1-21.

/15/ Shu,Nan C.: Visual Programming Languages : A Perspective and a Dimensional Analysis, In "Visual Languages", Shi-Kao Chang (Ed.), Plenum Press, New York, 1986, 11-34.

/16/ Wegner,P.: Learning the Language, Byte, March 1989, 245-253.