Structured Command Interaction Based on a Grammar-Interpreting Synthesizer

Sten Minör

LU-CS-TR:88-34

Also published in *Proceedings of the Second IFIP Conference on Human-Computer Interaction*, 1987

Department of Computer Science
Lund University

P.O. Box 118
S-221 00 Lund
Sweden

Structured Command Interaction based on a Grammar Interpreting Synthesizer

Sten Minör

Department of Computer Science
Lund Institute of Technology / Lund University
Lund, Sweden

Abstract

This paper describes a generic user interface for command interaction based on a grammar interpreting synthesizer. The interpretive approach facilitates interactive modifications of the command grammar which is useful both for design and personalization of the command interface for a specific application.

A brief description of the underlying synthesizer is given. The command grammars are described and the consequences for the interaction are examined. Two approaches for integrating the command interface with an application are given together with a discussion of the applicability of each approach. An example of the interaction style in a command synthesizer is given and the relation with other styles of interaction is discussed. The possibility to personalize the interface by modifying the grammars is examined and some directions for future work are given.

Categories and Subject Descriptors (Computing Reviews 1987): D.2.2 Software Engineering, Tools and Techniques - *User interfaces*

Additional keywords: human-computer interaction, interaction techniques, command grammars, adaptable user interfaces, synthesizers

1. Introduction

Several approaches for command interaction are used today ranging from dialog oriented text commands to a variety of graphical input, e.g. popup-menus, pulldown-menus, buttons, etc. Most command interfaces are not based on formal specifications and are implemented using ad hoc techniques. Using formal approaches uniform and consistent interfaces may be obtained from a high level description leaving out implementation details. The separation between user interface and application will also be distinct.

Formal approaches for user interface design has been utilized previously for different purposes. State transition networks has been used for interaction specification and automatic interface generation [1,2]. Variants of BNF grammars has been used for automatic generation of user interfaces [3,4], as a description of user commands and system response [5], as base for a interface simulation tool [6], and for prediction of user errors [7]. Examples of other formalisms are the Command Language Grammar [8], a framework for describing interaction at different levels, and Input-Output tools [9], a specification language for interactive systems.

The command interface described in this paper is based on SbyS (Structure by Structure synthesizer) developed by the author. SbyS is a generic synthesizer in the sense that it *interprets* a BNF-like grammar in building a hierarchical structure constrained by the grammar. The interpretive approach has the advantage that the grammar may be modified incrementally "on-the-fly". Compared with interfaces generated from a grammar, where only the *implementor* may affect the grammar (and thus the interaction), the grammar may in SbyS be modified interactively by the *user*.

The SbyS based command interface is hence a generic interface component which allows the user to create and modify the command grammar which will affect the interaction instantly. When developing user-interfaces for a certain application the advantages of rapid prototyping are obtained. No prototype has to be built, however, the real interface may directly be constructed and tailored. The interface is not only applicable for development. Different users, experts or novices, may model their own grammar in order to get a personally convenient and satisfactory interaction.

The SbyS interface is used for its own command interaction. Commands to SbyS are handled by a separate copy of SbyS which interprets a command grammar describing the

SbyS command language.

The rest of this paper is organized as follows. Section 2 describes the capabilities of SbyS, the underlying synthesizer. Section 3 describes the grammars and how they influence the interaction and presentation. A small example of how the interface may be used is given in section 4. Section 5 describes how the interface may be integrated to an application. In section 6 some implementation notes are given. A discussion of the qualitative properties of SbyS is found in section 7. Some possible future enhancements of the interface are suggested in section 8 followed by a summary in section 9.

2. SbyS - an overview

Program synthesizers has been an active research area during the last decade. The Cornell Synthesizer [10], probably the best known, was a source of inspiration to many following such as the Synthesizer Generator [11], ALOE [12], and GNOME [13]. Synthesizers are, in contrast to traditional editors/compilers where programs in textual form are *analyzed* and transformed to a syntax tree, *synthesizing* the syntax tree in dialog with the user. A program is synthesized by successively filling in templates representing language constructs. Those readers not familiar with synthesizers may glance at section 4, where a small example of synthesizing a command structure is given.

The Cornell Synthesizer was "hand-crafted" for one specific language while many of its successors were generated automatically from a language grammar by a synthesizer generator. The evolution is similar to that of parsers; earlier parsers were tailored for one language while most parsers today are generated by parser-generators.

SbyS is *not* a synthesizer that is generated from a grammar. Synthesizer generators work in a batch-oriented manner reading a grammar producing a synthesizer. Modifications to the grammar may not be done interactively and the turn-around time is considerable. SbyS is a generic synthesizer that *interprets* a context free grammar in synthesizing a target structure. The grammars may be edited simultaneously in exactly the same manner as the target structure, in fact by another copy of the same synthesizer. Consequently modifications to the grammar will come into effect immediately.

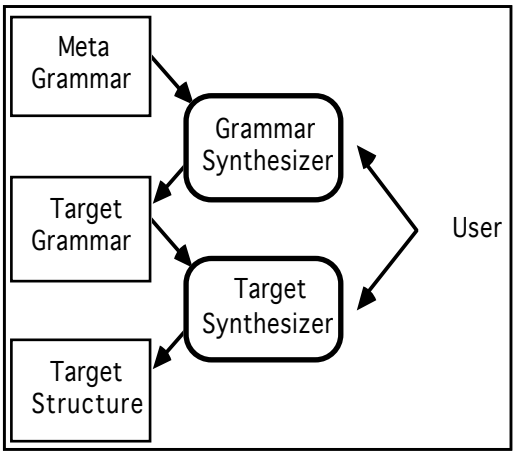

Figure 1

A simplified view of SbyS

Figure 1 shows a simplified view of SbyS. Two instances of the synthesizer are present. The *target synthesizer* which is used to manipulate the target structure interprets a target grammar. Since a grammar is represented in the same way as a target structure the target grammar may be target structure of another synthesizer, the *grammar synthesizer*. The grammar interpreted by the grammar synthesizer is describing a grammar and is thus a meta-grammar. The user may freely alternate between the two synthesizer instances and interactively modify the target grammar while the target structure is edited. A more detailed description of SbyS and its implementation is given in [14].

Feeding SbyS with a target grammar describing commands the synthesizer may be used for command synthesis, i.e. commands are constructed by successively filling in templates representing incomplete parts of the command. When a command is complete it may be executed by an application.

3. Command grammars and interaction

Describing the command interaction by a grammar has several advantages. A command grammar may be used as a specification of the command structures readable by humans. The specification is a compact high-level description free from annoying implementation details. Consistency of the command language can be enforced and checks for completeness and ambiguities may be performed resulting in a safer interface. Furthermore, in SbyS the specification may be executed resulting in a ready-to-use

interface separated from the application.

The grammar used in SbyS is separated in two parts. One part, the abstract grammar, specifies the representation of a command, i.e. its *logical* structure. The other part, the concrete grammar, describes the presentation of the logical structure, i.e the *physical* layout on the screen.

3.1 The abstract grammar

An abstract grammar is a slightly modified form of BNF. Terminal symbols in a traditional BNF grammar only represent syntactical sugar and are thus left out as only the logical structure of e.g. a command is described. An abstract grammar is built from four production types similar to those suggested by Ito and Cameron [15], namely construction, alternative, list, and lexem. A construction production describes an element with a certain number of sub-elements. An alternative production states that the user may select one of a number of alternatives. The list production describes an element with an arbitrary number of components of one specific type and the lexem production represents an element with a text component. Furthermore, an element may be specified as optional which means that it may be left out. Figure 2 gives an example of a small abstract grammar.

```
command ::= send-mail | read-mail | otherAlternative
send-mail ::= receiver-list & text & textConstruction
receiver-list ::= receiver*List
receiver ::= lexemLexem
...
```

Figure 2

Example of productions in an abstract grammar

The alternative productions do not only specify legal alternatives in a command structure, they also affect the interaction considerably. If the user wants to replace a template described by an alternation production the different alternatives is in SbyS presented in a menu from which the user selects the appropriate alternative. The chosen alternative may, however, be described by a new alternation production. The sub-alternatives will then be presented in a sub-menu, and so on. The alternative productions thus defines a

menu hierarchy. Putting all alternatives in one production will create a flat menu hierarchy with all alternatives in one menu. By grouping alternatives in sub-alternatives a deeper menu hierarchy is obtained with fewer commands in each menu but with a larger number of sub-menus. The user is in SbyS free to rearrange the menu hierarchy in order to obtain a convenient interaction, e.g. grouping commands after functionality or use-frequency.

3.2 The concrete grammar

The concrete grammar describes how a command structure shall be presented on the screen. It contains the syntactical sugar described by the terminal symbols in a traditional BNF grammar together with formatting information. Each construction, list, and lexem production in the abstract grammar has a production in the concrete grammar describing the presentation of that construction. Since the alternative productions in the abstract grammar only represent possible selections they do not have a description of how to be presented. A production in the concrete grammar is a template, different for each production type, consisting of texts and formatting information such as newlines and indentations.

An example of productions in the concrete grammar is given in figure 3. The template for presentation of a construction contains references to the presentation of its sub-components, <1>, <2>, and <3>, together with the text and formatting information for the actual component. A list template contains three parts, one that is displayed before the first list-element, one between each element, and one after the last element. Underscore denotes that nothing shall be displayed. Finally, a lexem template contains one part that is displayed before the lexem text and one that is displayed after.

```
send-mail ::=     "Send mail to: "                    Construction
                  <1>
                  <newline>"Subject: "
                  <2>
                  <newline>"Mail text: "<newline>
                  <3>
                  "End of message"
receiver-list::= before:_                             List
                  in: ", "
                  after:_
```

```
receiver::=     before:_                    Lexem
                after:_
...
```

Figure 3

Example of productions in a concrete grammar

Splitting up a grammar in one logical and one physical part has the advantage that the physical part, the concrete grammar, may be changed without affecting the structure of the language described by the grammar. Furthermore several different concrete grammars may be used together with one abstract grammar presenting the target structure in different ways. Hence the way a command structure shall be presented may be determined by the user. Cryptic incomprehensible commands can be replaced and a more personally satisfying presentation may be obtained.

4. An example

In this section a small example of how the SbyS user interface is used is given in order to give a flavor of how it works from the users point of view. The example is based on the abstract and concrete grammars of the previous section describing a subset of the command structure in some mail system. Figure 4-a shows the initial template generated from the start production of the abstract grammar in figure 2. The template is incomplete which is denoted by brackets.

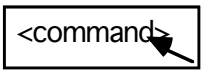

Figure 4 a

In figure 4-b the incomplete template has been selected by pointing at it with the cursor. A command is in the abstract grammar described by an alternation production containing three alternatives. The alternatives are presented in a menu and the user chooses to send mail.

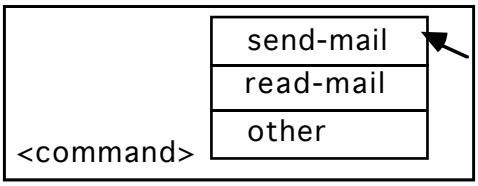

Figure 4 b

The send-mail command is described by a construction production containing three sub-components, a list of receivers, a subject, and the mail text. The send-mail template is presented, figure 4-c, using the concrete grammar of figure 3. The first sub-component is described by a list production stating that it may have an arbitrary number of receivers. The list is currently empty which is denoted by an underscore.

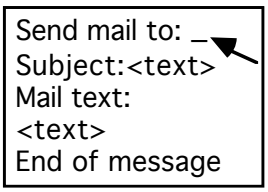

Figure 4 c

In figure 4-d the empty list has been selected and extended with two receiver templates. One receiver template has been selected for replacement. A receiver is in the abstract grammar described as a lexem which means that the user must type in the contents in textual form.

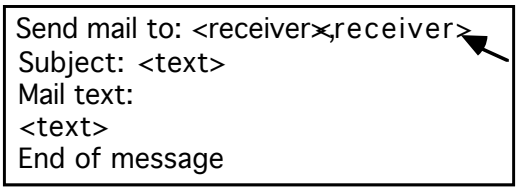

Figure 4 d

When the receiver templates are replaced some other incomplete template may be selected for replacement as in figure 4-e.

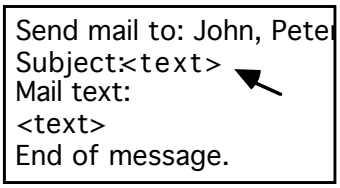

Figure 4 e

In figure 4-f the remaining incomplete templates has been replaced and the command structure is complete.

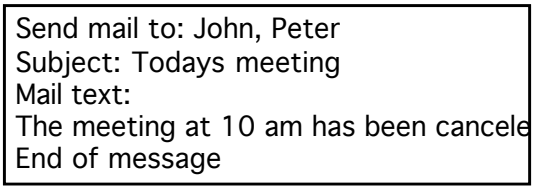

Figure 4 f

The example given in this section has shown the way command structures are synthesized and how the abstract and concrete grammars influence the interaction. The example is quite small because of the limited space in this paper. Many realistic command structures would be more complex with constructions containing other constructions, list containing lists or constructions, alternatives containing new alternatives, and so on. A complex command structure is, however, synthesized in the same manner as in the given example.

5. Integration with an application

The SbyS command interface is a separate module handling only command interaction. Separation of interaction and application functionality is desirable. An application can be developed focusing on issues in the application problem domain while the interaction module is developed focusing on interaction issues. It should be possible to modify or replace each part independently from the other. The separation is natural in a generic grammar driven interface like SbyS. The communication between the user interface module and the application must be flexible and well defined. Two approaches for communication are discernible in SbyS, textual and structural.

5.1 Textual integration

Textual integration is based on the physical presentation of the command structure. When a command structure is complete the physical presentation is sent in textual form to the application. The interface module may, using this approach, be integrated as a shell program on top of an existing traditional text-based application, e.g. an operating system or a mail system.

If the application command language is cryptic and inconvenient two concrete grammars may be used. One concrete grammar, synonymous of the other, is used for presentation. The user will see the commands and manipulate them using this grammar. The user is also free to modify the grammar after his/her own taste or needs. The other concrete grammar specifies the physical representation of the command-language used by the application. It can be viewed as the communication language between interface and application.

A macro-facility is easily obtained using textual integration. The user may extend the abstract grammar with a desired command possibly corresponding to several commands in the application language. The concrete communication grammar is also extended describing the physical representation of the command in the application language.

Textual integration has been tested by connecting the SbyS interface to the command-shell of a traditional operating system. The test has at present only been of technical nature. An evaluation of the relative merits of the SbyS interface compared with the original one remains to be done.

5.2 Structural integration

Structural integration is based on the logical representation of a command which serves as the communication language between interaction module and the application. An application connected to SbyS using structural integration has to be designed to interpret that representation. The applicability is thus restricted to "new" applications using that protocol. The interface-application communication is, however, standardized and well defined by the abstract grammar. It is also totally separated from the physical

presentation of the command structure.

In order to obtain a flexible interaction with possibilities for feedback, e.g. for help information or error messages, SbyS is notifying the application after each operation performed on the command structure. A notification contains both where a modification is performed and what kind of operation that is done. The application may decide how to react on a certain operation, e.g. checking the modified part for correctness or consider the command as complete and execute it. Parser-generators have a similar mechanism, usually called semantic actions. A semantic action is bound to a production and is executed when that production is used to parse the input. In SbyS the semantic actions are handled by the application, SbyS only notifies that a certain operation is performed by the user.

The concrete grammars may be modified freely using structural interaction but only restricted modifications to abstract grammars may be done. The hierarchy of alternations that effect the nesting of menus may be changed without affecting the logical representation of a command. If construction or list productions are modified or new productions are added the application must be able to interpret the command structure. In this case the flexibility of the application module sets the limit of how extensive the modifications may be.

Structural integration will be further examined in the Mjølner project. Mjølner is a joint Nordic research project on programming environments for object-oriented languages [16]. The Mjølner environment will be an integrated environment consisting of several components taking care of different tasks. Several instances of SbyS with different command grammars will be used by the different components, giving a homogeneous interaction style throughout the programming environment.

The integration alternatives are depicted in figure 5. The upper part shows textual integration. The physical representation is generated from the logical representation and the concrete grammar. The physical representation is sent to an application with a traditional textual command interface, which in the figure is represented by an arrow. The lower part shows structural integration. The logical command structure described by the abstract grammar is directly accessible from the application. SbyS is notifying the application whenever any modification occurs.

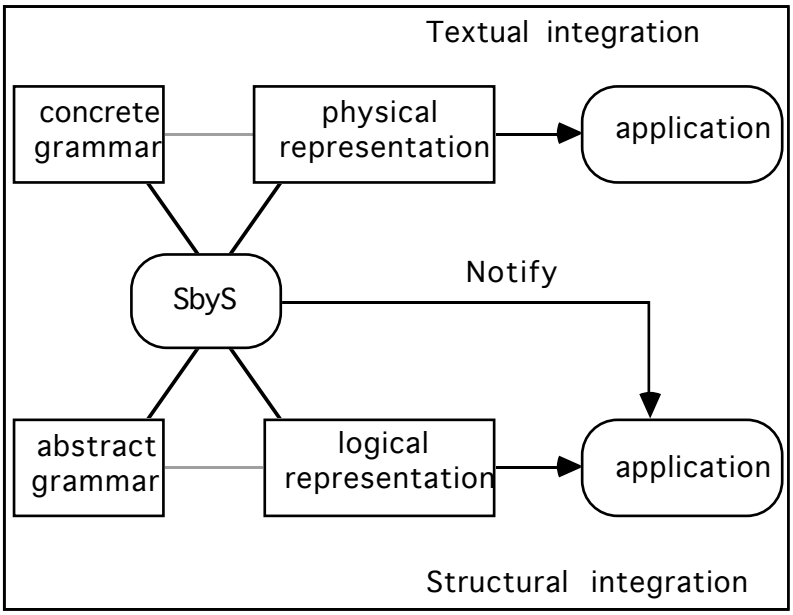

Figure 5

Textual and structural integration

As a summary, textual integration can be used to connect SbyS to traditional text-based tools in order to enhance the interaction and structural integration may be used in designing new tools using SbyS as a command interface with a well defined communication language.

6. Some implementation notes

A prototype of the SbyS interface has been implemented. The implementation is currently not of "production quality", but it will successively be enhanced. The basic functionality has been implemented: manipulation of structures such as replacing incomplete structures according to the abstract grammar, copying, cutting, pasting, and presentation of structures according to the concrete grammar. So is also simultaneous manipulation of the grammars and the target structure, and mechanisms for textual and structural integration. History, redo, and undo mechanisms are currently not implemented.

The prototype was written in Simula [17], a (truly) object oriented language including

dynamic instantiation of classes, inheritance, and coroutines. These concepts turned out to be very powerful in the implementation of the SbyS based user interface. The interface is described by a class that may be instantiated in several copies. Command, grammar, and metagrammar structures are represented uniformly as trees shared between the target and grammar synthesizer instances. The synthesizer instances are executing in parallel using coroutines to allow rapid alternation with preserved internal state.

Creating a new instance of a command interface with grammars is thus from the implementors point of view fairly simple. Several instances of command interfaces for different purposes executing in parallel are natural to express and straightforward to implement.

The class describing the SbyS synthesizer only contains the functionality of a general structure synthesizer. Other classes describing some other functionality may inherit the properties of the SbyS class. A class inheriting the SbyS class will be extended with the facilities SbyS offers. The SbyS based interface contains the mechanisms for integration with an application. Textual and structural integration are two alternative classes both inheriting the synthesizer properties from SbyS.

Commands to the SbyS user interface are synthesized in the same way as commands to an application. The interface handling the interaction with the SbyS interface is called the SbyS *meta interface*. There is no substantial difference between the two interfaces. The meta interface is simply another instance of SbyS which is structurally integrated with the SbyS interface in the same manner as with any application. The creation of the meta interface instance must be done dynamically, e.g. when the user wish to enter a command, as the meta interface in turn has a meta-meta interface in an infinite recursive manner.

7. Discussion

The relative merits of text and menu interaction are often discussed. The tradeoffs between the different styles are summarized by Norman in [18]. According to Norman textual interfaces are flexible in expressing parameters and modifiers and fast for expert users who know "what to do" and "how to do it". Menus are not as fast as commands for expert users but they offer valuable assistance for novices and occasional users and are often self-explanatory. Furthermore they eliminate typing and syntax errors.

Synthesizing commands gives a different style of interaction than text and menu interaction, but it resembles both in different respects. The alternative productions in the abstract command grammar is presented by SbyS as a menu giving assistance of possible alternatives. The flexibility of command languages is not restricted, parameters and modifiers can be expressed provided that they may be described by a context free grammar. Syntactical errors are eliminated as only commands specified by the grammar may be constructed.

The adaptability offered by SbyS improves the speed of searching for a certain menu entry among alternatives. Alternatives rarely used can be "stuffed away" in a menu for odd commands or even discarded. Alternatives chosen frequently may be placed in the first menu level for rapid access. Flat or deep menu hierarchies may be chosen by different users on different occasions. Despite the fact that the speed will be improved compared with traditional static menus, it will probably not be as fast as command languages for expert users.

The synthesizer interaction style also resembles "form fill-in" [19]. A traditional form fill-in design is easily expressed in the abstract grammar by a construction with lexems as sub-elements representing textual fields to be filled in. The actual layout of the form with field labels is described in the concrete grammar. Command synthesis in general may actually be viewed as "structured form fill-in" where the fields not only may contain letters and digits but also syntactic structures according to the grammar.

The possibilities to tailor the SbyS interface may be used for several purposes. System designers can use it for building prototypes of a command interface by modeling the grammars. The prototypes may be used for experiments both with the command structure and the presentation with very short "turn-around time". When grammars yielding a satisfactory interaction is obtained the interface is ready, the prototype has successively been developed to the "real" interface. Variants of the grammars can be designed for different user categories. A grammar for novices may contain assistance, explanations, and hints while grammars for expert users may be more brief. The users may later continue to model the grammars in order to personalize the interface obtaining an individually comfortable interaction.

An advantage of synthesizing commands is that the underlying model for constructing and modifying command structures is simple and the operations are few: select a command structure or parts of it, replace an incomplete template, cut, copy, and paste. The latter operations offer possibilities to edit command structures. Complex commands that are used frequently can be stored on a clipboard and later pasted to the target

command, eventually after minor modifications. A history mechanism for command structures, undo, and redo at the syntactical level would be desirable and could probably be added to the synthesizer in a straightforward manner.

No quantitative evaluation of the SbyS interface involving a large group of users has been made yet. Such an evaluation of *program* synthesizers has been made at Carnegie-Mellon University where the Gnome synthesizer was used to teach programming to undergraduates [13]. The experiences were almost exclusively positive giving a significant improvement in terms of program quality and time spent for teaching irrelevant issues as syntax. There is thus good reasons to believe that synthesizers may be practically feasible for command interaction. This issue has, however, to be further examined.

8. Future directions

At present a concrete grammar only support textual presentation of command structures. The most compelling extension is graphical presentation. If the concrete grammar was augmented with templates describing graphical object as lines, boxes, etc, a command structure might be presented both textually and graphically.

A generic parser is under development. The parser will be connected to the synthesizer serving as an alternative for textual input. The parser will be based on grammar interpretation in the same manner as SbyS is.

The synthesizer extended with parsing facilities and graphical presentation will be one step towards a general framework for text commands and graphical interaction.

Only input to an application is addressed in this paper. The response from an application would also gain from being specified in a formal manner. Widening the scope of the interface to handle application response or to describe complete dialogs like Scheidermans multiparty grammars do [5], is also an interesting subject for future work.

9. Summary

A generic command interface based on a grammar interpreting synthesizer has been described. The interpretive approach facilitates interactive modifications of the

grammars which may be used by the interface implementor for design and by the user for personalization.

The interface is a separate module which may be integrated with traditional textbased application as well as with new applications using a well defined protocol. Several instances of the interface executing in parallel used for different purposes can be created in a straightforward manner.

The interaction style of command synthesis resembles textual command languages, menus, as well as form fill-in. A combination of the flexibility of command languages and the assistance offered by menus together with the possibilities to personalize the interface may result in a user interface that could be appreciated both by novices and expert users.

Acknowledgements

This work was supported by the Mjølner project, partially funded by a grant from the Nordic Council of Ministers.

The author wants to thank Anders Gustavsson, Sten Henriksson, and Boris Magnusson for many valuable comments on the contents and style of this paper.

References

1.Jacob, R.J.K., *Using Formal Specifications in the Design of a Human-Computer Interface*, Communications of the ACM, 26 ,4 , 1983

2.Jacob, R.J.K., *Executable Specifications for a Human-Computer Interface*, CHI'83 Conference Proceedings, Special Issue of the SIGCHI bulletin, 1983

3.Olsen, D.R., Dempsey, E.P., *Syntax Directed Graphical Interaction*, Proceedings of SIGPLAN '83 Symposium on Programming Language Issues in Software Systems, SIGPLAN Notices, 18, 6, 1983

4.Olsen, D.R., Dempsey, E.P., *SYNGRAPH: A Graphical User Interface Generator*, Proceedings of SIGGRAPH '83, Computer Graphics, 17, 3, 1983

5.Shneiderman, B., *Multiparty Grammars and Related Features for Defining Interactive Systems*, IEEE Transactions on Systems, Man, and Cybernetics, SMC-12, 2, 1982

6.Hanau, P.R., Lenorovitz, D.R., *Prototyping and Simulation Tools for User/Computer Dialogue Design*, Proceedings of SIGGRAPH '80, Computer Graphics, 14, 3, 1980

7.Reisner, P., *Formal Grammar and Human Factors Design of an Interactive Graphics System*, IEEE Transactions on Software Engineering, SE-7, 2, 1981

8.Moran, T.P., *The Command Language Grammar: a representation for the user interface of interactive computer systems*, International Journal of Man-Machine Studies, 15, 1981

9.van den Bos, J., Plasmeijer, M.J., Hartel, P.H., *Input-Output Tools: A Language Facility for Interactive and Real-Time Systems*, IEEE Transactions on Software Engineering, SE-9, 3, 1983

10.Teitelbaum, T., Reps, T., *The Cornell Program Synthesizer: A Syntax-Directed Programming Environment*, Communications of the ACM 24, 9, 1981

11.Reps, T., Teitelbaum, T., *The Synthesizer Generator*, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, 1984

12. Medina-Mora, R., *Syntax-Directed Editing: Towards Integrated Programming Environments*, PhD thesis, Carnegie-Mellon University, Pittsburgh Pa., USA, 1982

13.Garlan, D., Miller, P., *GNOME: An Introductory Programming Environment Based on a Family of Structure Editors*, Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, 1984

14.Minör, S., *A Generic Synthesizer and its Implementation*, Department of Computer

Science, Lund Institute of Technology, Lund, Sweden, 1987

15.Cameron, R.D., Ito, M.R., *Grammar-Based Definition of Metaprogramming Systems*, ACM Transactions on Programming Languages and Systems, 6, 1, 1984

16.Dahle, H.P., Löfgren, M., Lehrmann-Madsen, O., Magnusson, B., *Mjølner - A Highly Efficient Programming Environment for Industrial Use*, Department of Computer Science, Lund Institute of Technology, Lund, Sweden, 1986

17.Birthwistle, G., Dahl, O-J., Myhrhaug, B., Nygaard, K., *Simula begin*, AUERBACH Publishers Inc., Philadelphia, USA, 1973

18.Norman, D., *Design Principles for Human-Computer Interfaces, CHI'83 Conference Proceedings*, Special Issue of the SIGCHI bulletin, 1983

19.Shneiderman, B., *Designing the User Interface*, Addison-Wesley, 1986