Structural Correspondence Specification Environment

Yongfeng YAN

Groupe d'Etudes pour la Traduction Automatique
(GETA)
B.P. 68
University of Grenoble
38402 Saint Martin d'Hères
FRANCE

ABSTRACT

This article presents the Structural Correspondence Specification Environment (SCSE) being implemented at GETA.

The SCSE is designed to help linguists to develop, consult and verify the SCS Grammars (SCSG) which specify linguistic models. It integrates the techniques of data bases, structured editors and language interpreters. We argue that formalisms and tools of specification are as important as the specification itself.

INTRODUCTION

For quite some time, it has been recognized that the specification is very important in the development of large computer systems as well as the linguistic computer systems. But it is very difficult to make good use of specification without a well defined formalism and convenient tool.

The Structural Correspondence Specification Grammar (SCSG) is a powerful linguistic specification formalism. The SCSGs were first studied in S.Chappuy's thesis [1], under the supervision of Professor B.Vauquois. In their paper presented at Colgate University in 1985 [6] SCSG was called Static Grammar, as opposed to dynamic grammars which are executable programs, because the SCSG aims at specifying WHAT the linguistic models are rather than HOW they are calculated.

A SCSG describes a linquistic model by specifying the correspondence between the valid surface strings of words and the multi-level structures of a language. Thus, from a SCSG, one can obtain at the same time valid strings, valid structures and the relation between them. A SCSG can be used for the synthesis of dynamic grammars (analyser and generator) and as a reference for large linguistic systems. An SCS Language (SCSL) has been designed at GETA, in which the SCSG can be linearly written.

The SCS Environment (SCSE) presented here is a computer aided SCSG design system. It will allow linguists to create, modify, consult and verify their grammars in a convenient way and therefore to augment their productivity.

Section I gives a outline of the system: its architecture, principle, data structure and command syntax. Section II describes the main functions of the system. We conclude by giving a perspective for futher developments of the system.

I. AN OVERVIEW OF THE SYSTEM

1. ARCHITECTURE

The SCSE can be logically divided in five parts: 1. SCSG base 2. monitor 3. input 4. output 5. procedures

The SCSG base consists of a set of files containing the grammars. The base has a hierarchical structure. A tree form directory describes the relationship between the data of the base.

The monitor is the interface between the system and the user. It reads and analyses commands from the input and then calls the procedures to execute the commands.

The input is the support containing the commands to be executed and the data to update the base. There is a standard input (usually the keyboard) from which the data and commands should be read unless an input is explicitly specified by a command.

The output is a support receiving the system's dialogue messages and execution results. There is a standard output (usually the screen) to which the message and results should be sent unless an output is explicitly specified by a command.

The procedures are the most important part of the system. It is the execution of procedures that carries out a command. The procedures can communicate directly with the user and with other procedures.

2. THE PRINCIPLE

An SCSE session begins by loading the original SCSG base or the one saved from the last session. Then the monitor reads lines from the command input and calls the corresponding procedures to execute the commands found. When an SCSE session is ended by the command "QUIT", the current state of the base is saved. The SCSG base can only be updated by the execution of commands.

The original SCSG base contains two SCSGs: one describes the syntax of the SCSL and the other gives the correspondence between the directory's nodes and the syntactic units of the SCSL. The first grammar is read-only but the second one can be modified by a user. This allows a user to have his prefered logical view over the base's physical data. These two grammars serve also as an on-line reference of the system.

Several interactive levels can be chosen by the user or by the system according to the number of errors in the command lines. The system sends a prompt message only when a "RETURN" is met in the command lines. So one can avoid prompt messages by entering several commands at a time.

3. DATA STRUCTURE

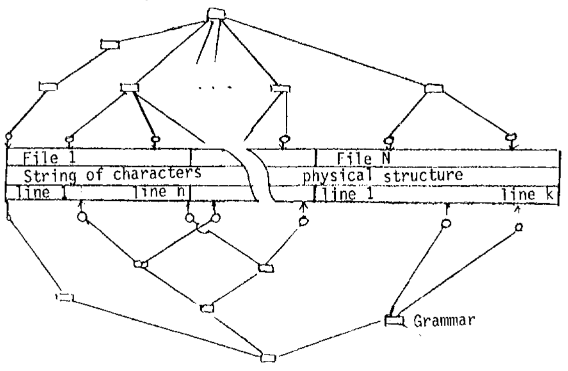
There are two data structure levels.

The lower one is linear, supported by the host system. The base is a set of files containing a list of strings of characters. The base can be seen as a single string of characters that is the concatenation of all lines in the files of the base so that the structure is said to be linear. This structure is the physical structure.

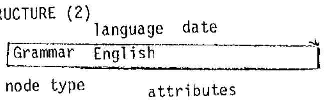
The higher one is hierarchical, defined by the directory of the base. The base is composed of a number of SCSGs ; each grammar contains a declaration section, a rule (chart) section ... etc. and the components of a grammar (declaration, rules ... etc.) have their own structure. The hierarchical structure is the logical structure of the base.

The directory has a tree form. A node in the tree represents a logical data unit that is its content (for instance a grammar). Every node has a type and a list of attributes charactering the node's content. The internode's content is the composition of those of its descendents. The leaf's content is directly associated with a physical data unit (a string of characters). The following figure shows the relation between the two structures.

LOGICAL STRUCTURE (1)



LOGICAL STRUCTURE (2)



The directory is similar to a UNIX directory. But in our directory, the leaves do not correspond to files but to logical data units and furthermore an attribute list is attached to each node. The correspondence between two structures is maintained by SCSE. We shall see later that this organisation allows a more efficient information retrieval.

It is possible for users to have access to the data by means of both structures. The logical one is more convenient but the physical one may be more efficient in some cases.

4. COMMAND SYNTAX

The general command format is :

<operator> <operand> <options>

- The "operator" is a word or an abbreviation recalling the operation of the command.
- The "operand" is a pattern giving the range of the operation.
- The "options" is a list of optional parameters of the command.

For example, the command : V GRAMMAR (LANGUAGE = ENGLISH)

visualizes, at the standard output, all the English grammars in the base. Here V is the operator, GRAMMAR(LANGUAGE=ENGLISH) is the operand pattern and no option is given.

The operand being mostly a node in the directory tree, the pattern is usually a tree pattern. When the pattern matches a subtree of the directory, the part that matches a specially marked node is the effective operand.

The pattern is expressed by a geometric structure and a constraint condition. The structure is a tree written in parenthesized form perhaps containing variables each representing a tree or a forest. The condition is a first order logic predicate in terms of the attributes of the nodes occurring in the geometric structure. More sophisticated conditions may be expressed by a predicate combined with geometric structure to efficiently select information from the base.

Pattern writing should be reduced to a minimum. In the above example, the geometric structure is simply a grammar type node and the constraint is the node's language attribute having the value: English.

The use of a current node in the directory allows not only the simplification of pattern writing but also the reduction of the pattern matching range. The effective operand becomes the new current node after the execution of a command.

II. THE MAIN FUNCTIONS

We shall just describe the functions that seem essential. The functions may be divided into four groups: 1. general 2. SCSG base updating 3. SCSG base inquiry 4. SCSG verification.

1. GENERAL FUNCTIONS

These functions include: SCSE session options setting, the system's miscellaneous information inquiry and access to host system's commands.

The following options can be set by user commands: 1. interactivity 2. dialogue language 3. auto-verification 4. session trace 5. standard input/output.

One of the 4 following interactive modes may be chosen: 1. non-interactive 2. brief 3. detailed 4. system controled.

In non-interactive mode, no question is asked by the system. An error command is ignored and a message will be sent but the process continues. In brief mode, the current accessible command names are displayed when a command is completed and a RETURN in the command lines is found. In detailed mode, the function and parameters of the accessible commands are displayed and if an error is found in the user's input data, the system will diagnose it and help him to complete the command. A prompt message is sent every time RETURN is found in the command lines. In the system controlled mode, the interactivity is dynamically chosen by the system according to the system-user dialogue.

For the time being, only French is used as the dialogue language. But the multi-langueage dialogue is taken into account in design. It is simpler in PROLOG to add a new dialogue language.

The auto-verification option indicates whether the static coherence (see 4. SCSG verification) of a grammar will be verified each time it is modified.

The trace option is a switch that turns on or off the trace of the session.

The standard input/output option changes the standard input/output.

Some inquiries about the system's general information, such as the current options and directory content, are also included in this group of functions.

The access to host system's commands without leaving SCSE can augment the efficiency. But any object modified out of SCSE is consided no more coherent.

2. SCSG BASE UPDATING

This group of functions are: CREATE, COPY, CHANGE, LOCATE, DESTROY and MODIFY. They may be found in all the classic editors or file management systems. The advantage of our system is that the operand of commands can be specified according to the logical structure of the base.

For example, the command : DESTROY CHARTS(TYPE=NP)

Destroys all the charts which describe a Noun Phrase.

The SCSE has a syntactic editor that knows the logical structure of the texts being edited. This editor is used by the commands MODIF and CREATE.

The command CREAT <operand> <options>

calls the editor, creating a logical data unit specified by the operand. If the interactive option is demanded, the editor will guide the user to write correctly according to the nature of the data. Following the same idea of different interactive levels, we try to improve on the classical structural editor, for instance that of Cornell University [5], so that one can enter a piece of text longer than that prompted by the system. If the interactive option is not demanded, one just enters into the editor with an empty workspace.

The command "MODIF <logical unit>" calls the system's editor with the logical data unit as the workspace. The data in the workspace may be displayed in a legible form which reflects its logical structure. The multi-windows facility of the editor makes it possible to see simultaneously on the screen the source text and the text in structured form.

The SCSE editor inherits the usual editing commands from the host editor. Thus one can change all the occurrences of a rule's name in a grammar without changing the strings containing the same characters, using a logical structure change :

C NAME(type=rule) old_name new_name,

while the physical structure command :

C/old_name/new_name/**

changes all the strings "old_name" in the workspace by new_name.

When an object's definition is modified, all its occurrences may need to be revised and vice versa even if the modification does not cause a syntactic error. A structure location command finding the definition and all the occurrences of an object can be used in this case.

Only the logical units defined in the directory and the SCSL syntax can be manipulated by the structural commands.

3. SCSG BASE INQUIRY

These functions allow users to express what they are interested in and to get the inquiry results in a legible form. A part of the on-line manual of usage in the form of SCSG may also be consulted by them.

The operand patterns discussed above are used to select the relevant data. The operator and options of commands choose the output device and corresponding parameters. A parametered output form for each logical data unit has been defined. The data matching the operand pattern are shaped according to their output form. The data may of course be obtained in their source form.

One may wish to examine an object at different levels (e.g. just the abstract or some comments). The options of the command can specify this. If one just wants to change the current node in the directory for facilitating the following retrieval, the same locating command as before may be used.

4. SCSG VERIFICATIONS

Two kinds of verifications may be distinguished : static and dynamic. The static verification checks whether a grammar or a part of a grammar respects the syntax and semantics of the formalism. The dynamic verification tests whether a given grammar specifies what we want it to.

Static verification

An internal representation of the analyzed text is produced and used by the system for structural manipulation. The analyser may produce a list of cross references of nameable objects and a list of syntaxo-semantic errors found in the text. The exemples of nameable objects are the charts, the macros, the attributes. The list of cross-references reveals the objects which are used but never defined or those defined but never used.

A chart may refer to other charts. This reference relation can be represented by an oriented graph where the nodes stand for a set of charts. A hierarchical reference graph is often given before writing the charts. A program can calculate the effective graph of a grammar according to the result of analysis and compare it with the given one.

The command options may cancel the output of these two lists and the graph calculation. The graph calculation may also be executed alone. One of options indicates whether the analysis will be interactive.

Dynamic verification

The dynamic verification is the calculation of a subset of the string-tree relation defined by a grammar. A member of the relation is a pair: <string,tree>. The command gives the grammar and the subset to be calculated. The subset may be one of the four following forms :

1. a pair with a given string and a given tree (to see whether it belongs to the relation)

2. pairs with a given string and an arbitrary tree

3. pairs with an arbitrary string and a given tree

4. all possible pairs

The calculation is carried out by an interpreter. The user may give interpretation parameters indicating interactive and trace modes, size of the subset to be calculated and other constraints such as a list of passive (or active) charts during this interpretation, the depth and width of trees and length of the string etc..

As SCSGs are static grammars, no heuristic strategy will be used in the interprete's algorithm. So the interpretation will not be efficient. Since the goal is rather to test grammars than to apply them on a real scale, the efficiency of the interpreter is of no importance.

CONCLUSION

The system presented is being implemented at GETA. In this article, we put emphasis on the system's design principles and specification rather than on the details of implementation.

We have followed three widely recommended design principles: a) early focus on users and tasks, b) empirical measurement and c) interactive design [2].

The specification of the functions are checked by the system's future users before implementation. The user's advice is taken into account. This dialogue continues during the implementation. The top-down and modular programming approaches are followed so that, even if the implementation is not completly achieved, the implemented part can still be used.

The system is designed for being rapidly implemented and easily modified thanks to its modularity and especially to a high level logic programming language: PROLOG [3]. We have tried our best to make the system as user-friendly as possible. The system's most remarkable character is that the users manage their data according to the logical structure adapted to the human being.

What is interesting in our system is not that it shows some very original ideas or the most recent techniques in state-of-the-art but it shows that the combination of well-known techniques used orignally in different fields may find its application in other fields.

Long term perspectives of the system are numerous. With the evaluation of the SCSG, some strategic and heuristic meta-rules may be added to a grammar. Equipped by an expert system of SCSG, SCSE could interprete effciently a static grammar and synthetise from it efficacious dynamic grammars.

It is also interesting to integrate into SCSE an expert system which could compare two SCSGs of two languages and produce a transfer grammar or at least give some advice for constructing it.

Using its logical structure manipulation mechanism, SCSE can be extended to deal with other types of structured texts. Thanks to its efficient interpreter or in cooperation with a powerful machine translation system such as ARIANE, SCSE could be capable of offering multi-lingual editing facilities [4].

BIBLIOGRAPHIE

1. S.Chappuy,
"Formalisation de la Description des Niveaux d'Intepretation des Langues Naturelles. Etude Menée en Vue de l'Analyse et de la Génération au Moyen de Transducteur.",
Thèse de troisième cycle à l'USMG-INPG, juillet 1983.

2. JOHN G. COULD & CLAYTON LEWIS,
"Designing for Useability: Key Principles and What Designers Think",
Communication of the ACM, March 1985 Volume 28 N° 3.

3. Ph. Donz,
"PROLOG_CRISS, une extention du langage PROLOG",
CRISS, Universite II de Grenoble, Version 4.0, Juillet 1985.

4. HEIDORN G.E., JENSEN K., MILLER L.A., BYRD R.J., CHODOROW M.S.,
"The EPISTLE text-critiquing system.",
IBM Syst. Journal, 21/3, 1982.

5. TEITELBAUM T. et al,
"The Cornell Program Synthesizer: a syntax directed programming environments.",
Communication of ACM, 24(9), Sept. 1981.

6. B. VAUQOIS & S. CHAPPUY,
"Static Grammars : a formalism for the describtion of linguistic models",
Proceedings of the conference on theoretical and methodological issues in machine translation of natural language, Colgate University, Hamilton N.-Y., USA, August 14-16, 1985

-o-o-o-o-o-o-o-o-