SCAT, AN AUTOMATIC-PROGRAMMING TOOL FOR TELECOMMUNICATIONS SOFTWARE

S. Barra, O. Ghisio, F. Manucci

CSELT - via G. Reiss Romoli, 274 - 10148 Turin (Italy)

ABSTRACT

The size, complexity and long life-time of telecommunications software, e.g. the programs for store program control (SPC) telephone exchanges, call for an increased software productivity and maintainability other than an improved quality. The availability of programming support environments based on standardized specification and programming languages greatly improves the software development process. Artificial Intelligence techniques are very promising aiming at further improvements and can provide a short-term payoff especially within an evolutionary approach leading up to an hybrid programming environment, i.e. a software environment made of both conventional and intelligent tools. The paper describes an intelligent tool, dubbed SCAT, based on ideas exploited by various automatic programming systems, like CHI, Programmer's Apprentice and DEDALUS. SCAT is strictly related to the telecommunications domain, thus it differs from other systems in the domain specifity. SCAT partly automatizes the most crucial phase in the software development process, i.e. the transition from project's detailed specification to the actual software implementation. SCAT has been tested in a few experimental software developments and in an actual application,i.e. the message handling system (MHS) to be made available in the Italian public packet switching network (ITAPAC).

1. INTRODUCTION

The most crucial phase of the software development process is the transition from detailed specification of the project to the actual implementation. A first attempt in designing and implementing tools supporting this phase via conventional techniques proved to be not very promising. Thus we addressed the problem of automatizing this phase according to the method of knowledge engineering and the transformational approach [1] [2]. Referring to the telecommunications field, the specification and implementation formalisms must be the Specification and Description Language (SDL) and the CCITT HIgh Level programming Language (CHILL) recommended by CCITT, the International Consulting Committee representing all telecommunications administrations. Background information about SDL and CHILL languages is referred to in section 2.

SCAT, SDL to CHILL Assisting Transformer, is based on ideas exploited by PSI [3], Programmer's Apprentice [4] and DEDALUS [5] systems.

Specification acquisition phase, synthesis phase and knowledge organization are similar to PSI ones. In the phase of specification acquisition, a program network is built. The program network is transformed into a complete and consistent description of the program (a program model represented via a hierarchical structure of frames) during the synthesis phase. Finally, the program model is translated in the target language. A detailed description of these two phases is given in section 3.3. The organization of the three kinds of knowledge (knowledge about SDL and CHILL, application domain knowledge and incremental knowledge) is presented in section 3.2.

SCAT provides the users with an assistance similar to that provided by Programmer's Apprentice system. User interaction is needed as SCAT could ask for missing information required in generating a working program corresponding to a particular specification (see section 3.1). Referring to the transformational approach adopted in SCAT, this system could be considered close to DEDALUS, in particular referring to the rules stating for specification and implementation languages and the mapping between them. Actually, transformations in SCAT are applied in a stiff and predetermined way, instead of depending at any time on the analysis of the results gained in previous transformations. In addition, as SCAT refers to SDL, i.e. a formal language, syntax and semantics of specification are far away from natural language or input/output predicates.
Information relevant to the gained experience in using SCAT and discussions about SCAT implementation are in section 4.

2. BACKGROUND

To overcome at least some of the difficulties derived by the dimension and complexity of telecommunications systems, CCITT (International Telegraph and Telephone Consultative Committee) has promoted the definition of international standards providing formal languages to support design and implementation of switching software.

Two languages are recommended : SDL (Specification and Description Language) [6] and CHILL (CCITT HIgh Level Language) [7]. The

former is usable by both Administrations to provide functional specifications of telecommunications systems and Manufacturers, to produce descriptions, i.e. standard documentation (Fig.1). The latter is a high level programming language, fulfilling typical requirements of the specific telecommunication area.

SDL and CHILL languages are rigourous means for specification and implementation activities. However, problems still remain in the transition from specification to implementation, owing on one hand to the different nature of the two languages, on the other hand to the specific usage of SDL and CHILL.

The formality degree required at SDL and CHILL levels is different, that is the first allows a certain degree of informality in the specification, while the second must obey the strongest formalization rules.

SDL and CHILL definitions have been carried out by their own CCITT groups, that even if emphasizing compatibility between the languages, designed them under different points of view. SDL is expected to provide a mean of specifying/describing the behavior of telecommunications systems: for this reason people involved in SDL definition focused on description problems specific to the telecommunications field. Conversely CHILL is a general purpose programming language. Thus, CHILL provides syntactic constructs typical of a high level programming language, while most of them are missing in SDL grammar.

As a matter of fact, some concepts are syntactically similar but semantically different, while others have the same semantics but are syntactically referred to in a different way. Then problems arise in modeling SDL semantics by CHILL.

The best way of matching the two languages semantics cannot be immediately identified: a match can be per formed using a set of CHILL constructs to reproduce a particular SDL concept.

On the basis of the problems previously outlined, an intelligent tool has been developed at CSELT, in order to support and partially automate some phases of software development and maintenance (Fig.2) [8].

3. THE SCAT SYSTEM

SCAT (SDL to CHILL Assisting Transformer) is an intelligent tool providing an assisted transformation in the production of a complete CHILL program corresponding to an SDL specification.
This allows to bridge the gap between specification and implementation levels due to the different informality degree of the former with respect to the rigourous formalism of the latter.

SCAT takes as input an SDL specification (see top of Fig.3) in a textual form, produced by a set of tools relevant to the specification (graphical editor, graphical-textual translator, syntax analyzer) and produces as output the corresponding CHILL code (see bottom of Fig. 3). The specification is acquired and analyzed by the specification analyzer, then the specification solver performs the synthesis activity, cooperating with the user's assistant and consulting the knowledge base.
The coder produces the CHILL output that will be processed within a conventional CHILL programming support environment [9].

3.1 User's Assistance

In the assistance approach it is the system which drives such intervention, asking the user to provide information missing at SDL level and necessary to obtain a compilable CHILL program. The system stores such an information, gradually increasing its knowledge about the SDL specification and the application domain. That allows SCAT to use, when applicable, the recorded information in other SDL to CHILL transformations.

Ideally, SDL Reccomendation Z-104 [6] allows the user to define in advance a good deal of information needed by SCAT. However, system designers could not use the full capabilities of SDL Reccomandations and they could prefer to be assisted in providing some information, e.g. data types.
Thus, the ratio of interaction between SCAT and the user can depend upon the application and the designer's attitude.

Moreover, SCAT offers the user the chance of choosing the most suitable CHILL statement to be used, among the alternative ones corresponding to a particular sequence of SDL constructs. Such a capability allows a wider and more appropriate use of the CHILL language, getting the best of its facilities. An example can be the use of CHILL "do while","do for" and "exception handling" statements.

Finally an effective help provided by SCAT is the capability to aid in updating a previously transformed specification, taking care in solving the inconsistencies possibly arisen in such an activity. In such a way SCAT permits to work at a higher level, i.e. the specification, imposing a standard process of project development and ensuring an always up to date documentation.

3.2 Three kinds of knowledge

All SCAT components strongly interact with the knowledge base, containing information about SDL specification rules, CHILL programming rules and the specific application domain.
In particular, two components make SCAT's knowledge. The first is a descriptive component consisting of information about the application domain and the system to be implemented. The second is a normative component, that is made up of a set of rules working on the descriptive component and regulating the transformation and updating processes.

The descriptive knowledge is composed of a program model and an incremental knowledge, both exploiting frames as storing units.

The program model is a hierarchical structure of frames maintaining the organization typical of an SDL specification. Each frame represents a particular SDL concept whose slots assume values either drawn from the specification or requested to the user.
Such values are stored in the slots during the instantiating process; as a consequence, some actions are made when a procedural attachment is linked to the slots involved; finally, the values are retrieved from slots in the coding phase.
The program model is dynamically built using

prototypical frames, that is frames not yet instantiated, and then suitably instantiating their attributes.

The incremental knowledge is organized in a set of frames retaining the information which appears to be reusable during the transformation of the same and/or other specifications of a particular project.

Such information is used in transforming SDL tasks into CHILL procedures and is made of CHILL code pieces and some further information characterizing it, that is parameters types, procedure attributes, file name containing the procedure code.

Figure 4 shows an example of task frame instance for the SDL task MAKE_ASSOCIATION(A,B). This task could be transformed into a CHILL procedure named MAKE_ASSOCIATION having A and B as present parameters. Since this task has been previously classified as "procedure with parameters", a procedural attachment handling present parameters, is started.

The procedure "manage_proc_with_param" recovers the CHILL name and the present parameters from the SDL name; retrieves the formal parameters of the procedure from the incremental frame, corresponding to the MAKE_ASSOCIATION task (Fig.5) if it already exists; assigns such types to the present parameters, suitably instantiating a declarative frame.

Figure 5 shows an instance of incremental knowledge task frame with some values useful for the CHILL definition of MAKE_ASSOCIATION procedure. This kind of information is acquired through an user interaction phase and recorded by the system, which exploits it again, when needed (e.g. for the present parameters definition of the task met).

The normative knowledge is made up of about 300 rules. In particular, they are concerned with :

- syntactic rules of SDL and CHILL languages (decoding and coding rules);
- correspondence rules between the two languages (equivalence rules);
- the rules performing slot instantiation and retrieval (property rules);
- the rules checking some consistency constraints, to be used in the updating activity (consistency rules).

An example of SCAT rules for the task concept is shown in Fig. 6. The decoding task rule (1) expresses that the SDL primitive at hand is a task if this primitive consists of a keyword TASK followed by the SDL name.

Two coding rules correspond to the task rule, depending on what the task stands for (i.e. abstraction or informal task (2) or assignment (3)). The choice of the suitable coding rule depends on the evaluation of the equivalence rules (6) and (7). The instance and the retrieval in the task frame for <SDL_NAME> and <CHILL_NAME> respectively is performed by the property rules activation (4) and (5).

3.3 Transformational Process

In the transformational process (Fig. 7), SCAT gradually acquires the specification, through the application of the SDL rules, building at the same time the program model (specification analyzer), suitably instantiating its slots with those values at the moment available in the specification; then, linking it to its parent frame. Looking at those empty slots in the program model, intended for storing CHILL information, SCAT becomes aware of what it needs to issue the CHILL code. Then, it fills up such slots consulting the model itself, the incremental knowledge and, when needed, the user (complete model builder, user interface). Finally, it generates the CHILL code through the model scanning and the application of the programming rules (coder).

As far as the updating activity is concerned, SCAT allows the user to substitute a chunk of a previously acquired specification or to insert further pieces in it. A submodel for the SDL piece to be added or substituted is created and fulfilled by the same components acting in the transformation process. The submodel is then inserted at the right place in the starting model; at the same time, checks are made throughout the whole model to identify and solve the possible inconsistencies caused by the updating (model modifier). The code generation is finally performed in a completely transparent way with respect to the carried out changes.

4. EXPERIENCE OF USE AND IMPLEMENTATION

A few experimental applications of SCAT has been carried out and the system is presently used in an actual application development [10]. A communication protocol [11] fully specified in SDL is automatically implemented in CHILL using SCAT system.

The detailed specification of a communication protocol is not an easy task: its implementation based on this specification is even more complicate, requiring a good knowledge of the problem itself. Usually the implementation of a communication protocol consists of a considerable amount of code, therefore it represents a good test for an automatic translator.

This experience made explicit that software productivity can be increased by a factor from 5 to 10.

SCAT system has been developed in PROLOG language, in particular in CProlog (1.2 and 1.5 versions) and in Quintus (1.2 version), running on VAX-11 machines (UNIX and VMS operating systems) and SUN workstations.

Some differences of behavior, depending on the PROLOG versions and the computer used, have to be pointed out. Primary requirement to translate an average-high SDL specification (about 500 SDL lines) in the corresponding CHILL implementation is to change the memory dimension of CProlog. In particular CProlog stacks have to be arranged in order to avoid an abort of the transformation process. The transformation of the same specification in VAX/VMS environment and in VAX/UNIX environments using CProlog 1.5

produces two different behaviors. In the VAX/VMS stack dimensions have to be fixed in the following manner:

- global stack : 4100K
- local stack : 4800K
- heap stack : 500K
- trial stack : 150K

In the VAX/UNIX the transformer cannot be run because the greatest dimension of a UNIX task is 4Mb, as also occurs on Sun workstation.
The CPU time required by the transformation of the above mentioned specification is about 30.54 sec. and the generated program is about 2,000 CHILL lines long.

5. CONCLUSIONS

SCAT system is based on some ideas exploited by various automatic programming systems: PSI's synthesis phase and knowledge organization, the assistance approach provided by Programmer's Apprentice, the transformational approach adopted in DEDALUS.

Actually, SCAT does not provide new general applicable ideas; it rather shows how some ideas from automatic programming field can be instantiated in the specific telecommunications domain.
Such a domain is one of the largest software application areas, thus it justifies huge efforts in attempting to increase software productivity and maintainability other than to improve its quality.

Significant payoffs have been achieved in SCAT development exploiting AI techniques according to evolutionary approach which aims at hybrid software environments.
First of all, AI techniques allow to improve the cooperative assistance to the user and to efficiently organize the knowledge required in the transformation task.
Moreover, the joint use of frames and rules has allowed to quickly write a first SCAT prototype, afterwards tailored to the particular application in a tuning activity.

The use of Prolog language has outlined its suitability to represent and evaluate a knowledge organized in a production rules form; furthermore, it made it possible to realize in a short time a first prototype of the system; finally, it assures an easy SCAT adaptability to others programming and specification languages.

SCAT makes easier, cheaper and more reliable software development and maintenance, taking care of burdensome and error-prone tasks, as consistency and match between specification and implementation, and leaving to the user the responsibility for the most crucial decision making.

REFERENCES

[1] E. Lerner "Automatic Programming" Computers Software II IEEE 1982

[2] J. Phillips "Self-Described Programming Environments-An application of a Theory of Design to Programming Systems" Technical Report SIAN-CS-84-1008, Kestrel Institute 1983

[3] E. Kant, D. Barstow "The refinement paradigm : the interaction of coding and efficiency knowledge in program synthesis" IEEE Transaction of Software Engineering Vol. SE-7 n.5 1981

[4] C.Rich "Inspection methods in Programming" - Technical Report AI-TR-604-M.I.T. A.I. Labs 1981

[5] Z. Manna, R. Waldinger "Synthesis : dreams -> programs" IEEE Transaction on Software Engineering Vol. SE-5 n.4 1979

[6] CCITT Recommendation Z.100 - Z.104 "Functional Specification and Description Language (SDL)" 8th. Plenary Assembly Malaga-Torremolinos 1984

[7] CCITT Recommendation Z.200 "CCITT High Level Language (CHILL)" 8th. Plenary Assembly Malaga-Torremolinos 1984

[8] Barra S., Ghisio O., Modesti M. "The use of artificial intelligence in the transformation from SDL to CHILL" II SDL users and implementors forum Helsinki 1985

[9] Bagnoli P. et al. "Towards a Software Engineering environment for telecommunication systems based on CCITT standards" - XI International Switching Symposium -Florence 1984

[10] Barra S., Ghisio O., Modesti M. "Experience and problems of applications of automatic translation from SDL specifications into CHILL implementations" 6th. International Conference on Software Engineering for Telecommunication Switching Systems Eindhoven 1986

[11] CCITT Recommendation X.411 "Message Transfer Layer" 8th. Plenary Assembly Malaga-Torremolinos 1984

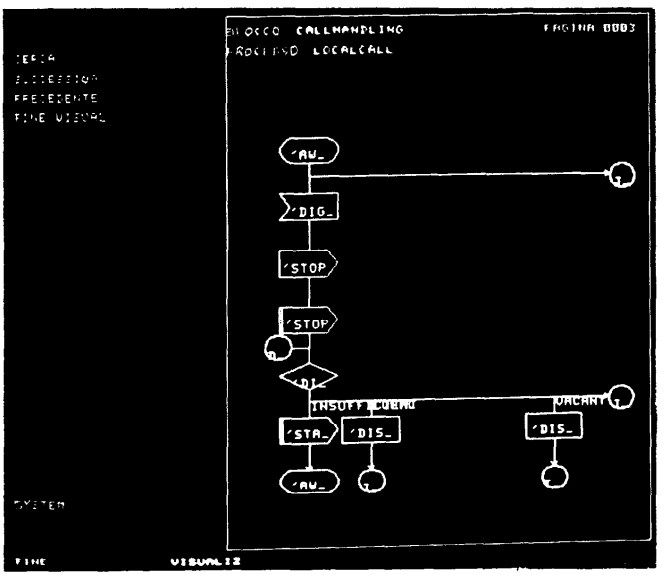

Fig. 1 - An example of SDL specification in graphical form

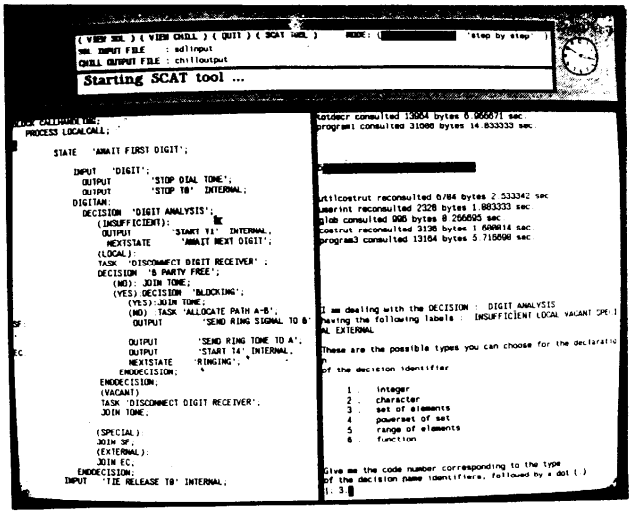

Fig. 2 - The SCAT window environment

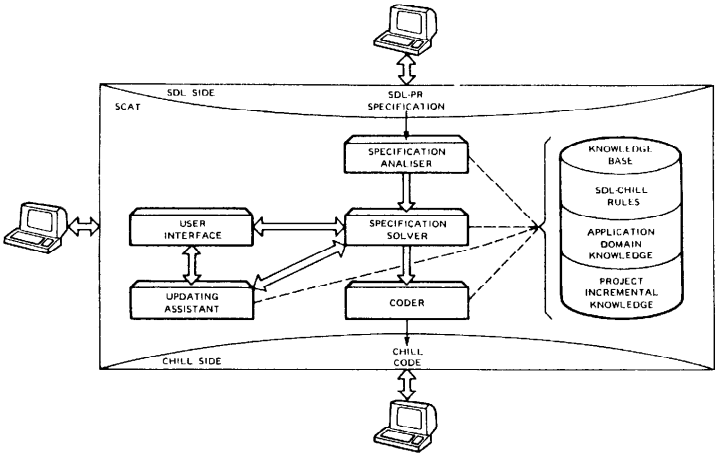

Fig. 3 - The SCAT system

```
(1) TASK <SDL_name> [ <comment> ] ; --> <task>

(2) <call_action> --> CALL <CHILL_name>
    ( <present_par> {, <present_par> }* )
    [ <comment> ] ;

(3) <assign_action> --> <CHILL_name>
    [ <comment> ] ;

(4) <SDL_name> --> fill_slot(SDL_NAME)

(5) <CHILL_name> --> get_slot(CHILL_NAME)

(6) <task> --> <call_action>
          if <SDL_name> is "abstraction"

(7) <task> --> <assign_action>
          if <SDL_name> is "assignment"
```

Fig. 6 - Example of rules

SLOT	VALUE	PROCEDURAL ATTACHMENT
SDL-NAME	MAKE-ASSOCIATION (A, B)	
IMPLEMENTED	CHILL	
KIND	PROCEDURE WITH PARAMETERS	•
CHILL-NAME	MAKE-ASSOCIATION	
PRESENT-PAR	A, B	
COMMENT		
LABEL		

```
proc:   MANAGE-PROC-WITH-PARAM (chill-name, present-par)

        chill-name - recovering (chill-name);
        param-types-recovering (present-par, types);
        present-par-slots-instantiation (present-par, types)

end;
```

Fig. 4 - A frame example of program model

SLOT	VALUE	P.A.
NAME	MAKE-ASSOCIATION	
FORMAL-PAR	TYPE1, TYPE2	•
KIND	PROCEDURE	
RECURSIVE	NO	
RETURN		
CODING_LANGUAGE	CHILL	
FILE_NAME	FILE	

```
proc:   ASSIGN_TO_PRESENT_PARAM (formal_par, name);
        task_frame_recovering (name, present_par);
        assign_types (present_par, formal_par);

end;
```

Fig. 5 - An example of incremental knowledge

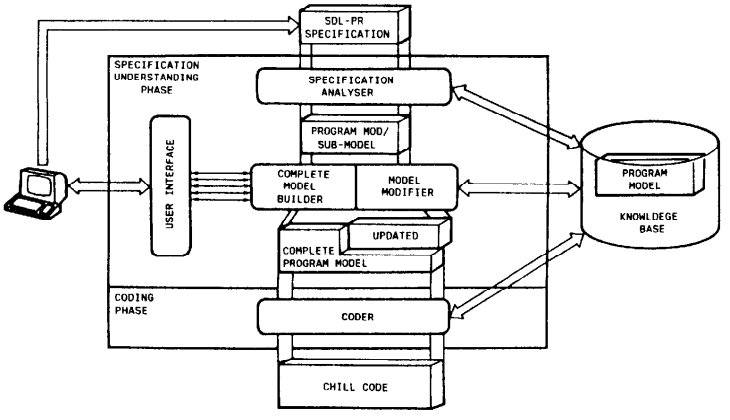

Fig. 7 - The transformational process