

COMPUTER-AIDED CHILL PROGRAM GENERATION FROM SYSTEM SPECIFICATIONS :
DESIGN EXPERIENCE FROM THE MELBA PROJECT

L.N. Jackson*, C.J. Fidge*, R.S.V. Pascoe*, P.H. Gerrand**

* Royal Melbourne Institute
of Technology

** Telecom Australia

This paper reviews the design challenges encountered and experience gained during development of MELBA, an automatic code generation system. MELBA combines computer-aided system specification using SDL with the automatic conversion of the specification into CHILL code.

The system specification methodology utilizes a three-level hierarchy of inputs beginning with the proposed hierarchical partitioning of the system, specified using *Module Structure Diagrams*, followed by process (subsystem) specifications using the 1980 CCITT *Specification and Description Language* (SDL), and finally detailed data structure information from a library of *Abstract Data Types*.

The paper compares the major features of MELBA with other known similar projects throughout the world. Initial experience gained in applying MELBA to an undergraduate design project is discussed.

1. INTRODUCTION1.1 Background

MELBA is an ambitious project involving the design and implementation of an interactive software system which aids the specification of complex systems, such as telecommunications systems, and then automatically converts the system specifications into compilable high-level language code.

The MELBA project began in 1979. The project has been based at the Royal Melbourne Institute of Technology and Telecom Australia's Research Laboratories.

1.2 SDL and CHILL

SDL and CHILL are two international languages designed and standardised by the CCITT. Both languages are intended for SPC (Stored Program Control) telecommunications switching but are capable of wider application.

SDL (the CCITT Specification and Description Language first specified in 1976 and again in 1980 [1]) provides a graphical means of specifying the behaviour of telecommunications systems. It can be used both as a software development tool and as a high level documentation technique. Based primarily on state transition diagrams, SDL combines traditional flowcharting features such as operations (tasks) and decisions with the concept of signals, allowing communication and synchronization between concurrently executing program units. Each SDL diagram is used to describe a single concurrent process. SDL can be used in either

of two forms: the more common graphical form or the equivalent program-like form (known as SDL/PR). The new 1984 CCITT Recommendations are expected to provide a greatly enriched repertoire of SDL facilities.

CHILL (the CCITT High Level Language) is a relatively new (1980 [2]) programming language based primarily on Pascal, PL/I and Algol-68. Reviewing existing programming languages, the CCITT found that none were entirely suitable for SPC use and proceeded to develop a new language specifically for this purpose. CHILL differs from most of its predecessors in its approach to types, the extensive concurrency features included, and in its modularisation and exception handling constructs.

The advantages of a system such as MELBA can now be readily appreciated. SDL and CHILL are both standard languages developed by the same body with many conceptually similar features (the CCITT is emphasising compatibility between SDL and CHILL). Also both languages are (or will be) familiar to the majority of telecommunications engineers and will be widely used in the near future. If current trends continue system development and documentation will commonly be performed using SDL while the system itself will often be programmed using CHILL. A technique for automatically converting SDL specifications to CHILL code will greatly enhance design productivity and increase system reliability. Also documentation and code will always match since one is derived from the other.

1.3 Comparisons with other automatic code generation systems

The impending definition of SDL and CHILL in the late seventies resulted in several groups

International Switching Symposium
Florence, May 7-11 1984

throughout the world starting work on projects similar to MELBA. In fact MELBA was inspired by SX1 [3], one of the first of such projects and the most experienced automatic code generation system known to the authors, and stimulated by CADDIE [4], an early computer-aided graphical SDL specification system whose functional behaviour was itself specified using SDL, in the form of a Dialogue State Transition Diagram [4].

The SX1 system represented the state-of-the-art for automatic code generation when the MELBA project began in 1979. It used a textual form of input which was processed to produce an SDL-like diagram and PO-CORAL code. Nevertheless, despite being well developed and tested, SX1 (now known as CADOS [5]) seems to be less ambitious than the current version of MELBA, since it requires the user to include target language statements in the graphical symbols. MELBA aims to be target language independent, the SDL user working purely with English-like statements.

Since the beginning of the MELBA effort a number of "rival" systems have surfaced including the "SDL to CHILL skeleton transformer" [6]. As the name implies this system is capable of generating the outline of a CHILL program but cannot generate complete, compilable code. This system appears to be remarkably similar in aim to the 1979 version of MELBA.

Other software development systems based on SDL include the "System Generator for ESS" [7], CROSS [8] and EGS [9] each of which relieve much of the programmer's burden. They use SDL-like inputs.

The main user interface to the MELBA system is a utility allowing the user to draw SDL diagrams at a graphics terminal by manipulating pre-defined symbols with a cross-hair cursor. A large number of automated SDL drafting tools with similar capabilities have been developed recently [4,10,11]. Many of these provide drawing facilities much more sophisticated than those available in MELBA, however very few have considered the question of automatically converting the diagrams thus produced into code and none appear to have made serious inroads into this research area. The most sophisticated SDL drawing aid known to the authors is the one developed by Pierre de Chazal of the CSIRO Division of Computing Research, Canberra, Australia in 1981-82.

Finally, papers submitted to CCITT meetings, sometimes contain intriguing, but brief, mentions of SDL to CHILL translation systems developed by various organizations throughout the world, but since this is still very much a research area the details of such projects are often closely guarded.

2. THE MELBA SYSTEM AND ITS METHODOLOGY

2.1 Overview

The current aims of the MELBA system are:

- (a) To aid the *system designer* by providing computer-aided system partitioning using nested

Module Structure Diagrams.

- (b) To aid the *process designers* in specifying the partitioned subsystems using target-language-independent SDL, with high level checking of syntax and text.

- (c) To enable an *abstraction designer* to create and manage a library of abstract data structures for use in target systems.

- (d) To permit automatic generation of CHILL code (action code and data declarations) as a faithful implementation of the system specified using MELBA.

Figure 1 provides an overview of MELBA's functional design, showing the interfaces with the *designers* and the data flow between the major subsystems.

2.2 System Partitioning using Module Structure Diagrams [MSD's]

The *system designer* can define the structure of a proposed system in a series of hierarchical diagrams consisting of modules and processes [12]. Modules are a means of restricting visibility. Each module can be decomposed into further modules and/or processes. A process cannot be decomposed in an MSD diagram; its functions are described using SDL. Figure 2 illustrates the form of the MSD.

Communication between processes is shown in an MSD by drawing all the signals that are sent to/from each process (solid lines in Figure 2). These must correspond to the signals used in the SDL diagram for the particular process. Two processes in the same module can communicate directly but if a process needs to send a signal to a process in another module, the signal must go via a 'port'. A port can be described as a hole in a module boundary through which signals can pass. Where ports are connected, the connection may be viewed as a unidirectional pipe or channel (dotted lines in Figure 2). In this way processes that cannot "see" each other due to visibility restrictions caused by modules can still communicate via ports and channels.

2.3 Use of SDL

MELBA provides the *process designer* with a major subset of the 1980 CCITT SDL specifications. Recent developments from the CCITT have been examined but no attempt will be made to incorporate them in MELBA until a new recommendation has been finally defined and approved in 1984. The optional "pictorial elements" are not used and the embryonic "functional block diagrams" are replaced by Module Structure Diagrams.

MELBA allows a third type of signal apart from the external and internal types found in SDL: the "encapsulated" signal. Encapsulated signals solve two problem areas that were both left undefined in the 1980 SDL standard: they can be used to implement signals that do not follow the usual semantic rules (e.g. conditional signal acceptance, continuous signals, signals with priorities, etc.) and they cover the concept of shared data between processes, solving the mutual

exclusion problem that normally results from attempting to share variables between two or more process instances.

In fact all of the modifications required by the MELBA system were necessary to rigidly define areas that were not included in the 1980 SDL specification. These problems are currently under review by the CCITT but until the 1984 SDL definition is available the MELBA system must provide its own solutions to these difficulties.

2.4 Introduction of Data

In MELBA all data information must be stored as specific, well-designed abstract data types. The 1980 SDL does not define the use of data, an essential requirement in the generation of high level code. Data structure information can be obtained where necessary from a library of data structures and operations manually coded and incorporated in the MELBA software system. The library approach was used since it relieves the *process* and *system designers* from the burden of defining data structures, something that is not normally done when using SDL.

The full advantages of the abstract type concept are used by allowing types to be parameterised. These types (from the "permanent" abstractions library) are recognised by the system and can be used as if they were built-in types in a typical programming language. Their parameterisation allows them to act as building blocks for creating an even wider range of types.

Although it is hoped that a thorough library of abstractions can be constructed it would be unrealistic to assume that this library, no matter how large, will be suitable for all MELBA user's needs. To overcome this it is possible for an *abstraction designer* to create a separate temporary library for a particular project under development (the "project" abstractions library) containing any special types required.

2.5 Generated CHILL Code

MELBA currently generates a proper subset of CHILL as output (many of the more esoteric features of the language are simply not required). To make the code as readable as possible, all identifiers used are left intact and there is a direct correspondence maintained between the MSD, SDL and library input and the CHILL output.

2.6 Other aspects of MELBA Methodology

One of the most important features of the system from the user's point of view is the self-documenting text employed in SDL. The SDL user actually works solely with high-level operations defined in the abstractions library, however the syntax for these operations has been defined so that they look more like English than procedure calls. With the use of meaningful identifiers for local variables the *process designer* can draw SDL diagrams containing compilable text that reads like natural language and is fully self-documenting. This conforms with SDL's main

aim of being a specification rather than a programming language.

Also, since the *system* and *process designers* work with types and operations available from the library, they do not need to be aware of the target language and in fact do not even need to know which language is being generated.

3. IMPLEMENTATION OF MELBA

The first two phases of the project (1979 to 1981) saw prototype software developed in a number of languages including BASIC and Pascal on a variety of computers ranging from a micro to a very large mainframe. Unfortunately, the diversity of languages made the software difficult to maintain and hardware problems caused the system to be unreliable at times. At the beginning of the third phase of the project, the MELBA development group obtained access to new hardware, far more reliable than the original system. As a result it was decided in 1981 to abandon the original software and develop a totally new system, this time written almost exclusively in ISO Pascal to give a degree of portability.

3.1 Hardware

The *system* and *process designers* prepare MSD and SDL diagrams on a microprocessor and associated hardware forming a graphics workstation. The principal hardware items in the workstation are a DEC compatible LSI 11/2 minicomputer and a Tektronix 4014 graphics display terminal. The minicomputer has 60 kilobytes of usable memory, dual floppy disc drives of total capacity 2.5 megabytes, serial and parallel I/O ports and a Centronics type printer port. The serial I/O ports are used for the console terminal and connection to a 1200 baud telephone modem. The parallel port provides a high speed connection to the Tektronix graphics display. All commands and responses to/from the system are via a console terminal; the 4014 is simply a facility for graphics output and cursor position input.

A 1200 baud modem is used to connect the workstation via a dial-up telephone line to a DEC VAX-11/780 mainframe (20 kms away) where the off-line code generation programs reside. Communications software developed by the project team for the LSI 11/2 allows the user at the graphics workstation to send files directly to the VAX, process them and receive the results, all in a single workstation session.

3.2 Software

The MELBA software suite consists of seven main units (the rectangles in Figure 1). All code is written in Pascal. Overall the system design was divided into two areas to allow for independent (and concurrent) software development. The three software units on the left of Figure 1 form an interactive graphics facility, resident in the workstation, while the remaining units are the off-line code generation programs.

The interactive graphics facility is supported by three programs: SDL.GR, MSD.GR and the Graphics Handler. The Graphics Handler implements a sub-

set of the CORE graphics standard [13] and provides a sophisticated interface between the other two graphics programs and the graphics terminal hardware. MSD.GR and SDL.GR manage the development of MSD and SDL diagrams respectively. They are both menu driven and easy to operate. Symbols are placed and connecting lines drawn by menu selection and manipulation of a cross-hair cursor on the graphics terminal. This facility provides the main user interface to the system. At the end of each MSD.GR or SDL.GR session a file is automatically produced describing the diagram created. This file is processed by the off-line programmes to generate code.

The SDL/PR Generator accepts the intermediate description of an SDL diagram produced via the graphics software. Based on this it produces the equivalent SDL/PR description of the process. This listing will contain error messages for all syntax errors in the diagram as well as detectable semantic errors. This serves as immediate feedback to the process designer who can then modify the SDL diagram accordingly. When error free the SDL/PR code is saved for later processing by the Code Generator.

The Abstractions Management System is a utility for the creation and modification of the abstractions libraries. It allows the *abstraction designer* to create, enquire about, delete and modify library entries. Only privileged users may gain access to this utility.

The Code Generator processes the complete project description to produce final CHILL code. This is the only target language dependent program in the system. It uses all stored files to construct and generate CHILL code. It also does complete error checking before the final code generation (listed in the Code Generation Report). Errors detected at this stage include problems with the MSD diagram, use of abstract types that do not exist in the library or misuse of existing types, identifier visibility errors or identifier clashes, etc. The program can also run in a checking mode where only individual items (e.g. MSD's, processes or library entries) are examined and reported on, without code generation. This enables feedback to the system users without the need to perform a complete code generation run.

It should be emphasised that MELBA is still a research tool rather than a commercial product. User experience is very limited and as a result the practical viability of the system under industrial conditions remains to be proven.

4. EXPERIENCE WITH MELBA

In the latter part of 1983 the first major attempt was made to use MELBA by people outside the MELBA development team. These people (a group of three) were undergraduate students in communication engineering as were another group of three which were used as a control group (the 'ad hoc' group).

Essentially an experiment was conducted to informally compare the performance of the group

using MELBA with the 'ad hoc' group. The observations made of their performances are detailed below.

The two groups were instructed to specify, using SDL, and then implement in a subset of CHILL, a simple telephone switching facility involving queuing. The MELBA group adhered strictly to MELBA methodology, whilst the 'ad hoc' group was free to use any approach of their choice. Both groups were familiar with the basics of SDL and the CHILL subset used. The MELBA group was given additional instruction in the special features of the SDL subset, data facilities and the operation of the existing MELBA software. The students assimilated this information very quickly.

Since the MELBA software was incomplete at the time of this experiment, the MELBA code generation function had to be simulated by hand with one member of the MELBA research team acting as automaton.

The project involved specification and implementation (production of code) of a simple automatic call distributor catering for 20 incoming lines and 3 servers. The project was further divided into 2 parts:

(A) A process is to be constructed to maintain a FIFO queue (of length 6) with 'music' being provided for the held calls.

(B) As in part (A) but with a more flexible dequeuing mechanism. Servers can take held calls from the queue in strict FIFO discipline or from the front 3 positions on a 'caller attribute' basis, overriding the FIFO mechanism.

Part (A) was made fairly trivial by giving the students access to a reference [14] that included a simple queue managing process specified in SDL and was mainly intended to help introduce the students to SDL and MELBA. The second part presented a much greater challenge including a reasonably complex data structure and the additional problems of mutual exclusion among the servers attempting to examine and access the queue.

For both parts the MELBA group was provided with a small abstractions library containing a number of typical Abstract Data Types (ADT's). Care was taken to avoid unfairly biasing the project with this library: only types that would be found in a 'real-life' permanent library were provided.

This library was in no way 'customised' for the project. The "queue" type, for example, was a general purpose generic type with typical operations - Add, Full?, Empty? and Get. Learning how to instantiate library modules was expected to present problems to the students who had little programming experience and were totally unfamiliar with the concept of parameterised types. It is important to note that for part (A) the supplied library was adequate for all of the MELBA group's needs. For Part (B) additional types were required for an efficient solution.

Although relatively simple and capable of

solution with a single MSD, a number of significant observations were derived from part (A). The MELBA group was divided into *system*, *process* and *abstraction designer* and maintained this structure throughout the duration of the project with occasional overlapping when difficulties or time constraints were experienced. By contrast the ad hoc group was dominated by one forceful member and only a token effort was made by the group to work together. It soon became obvious that almost all work was being performed by one student. This is not uncommon with student projects and it is noteworthy that the MELBA group also contained a dominant personality (the *process designer*) but despite this the group structure did not break down.

One clearly significant feature of the ad hoc group's approach was that the implementation code was always produced before the SDL diagrams. The group went through several complete drafts of their system but at no time were the SDL diagrams available before the code despite the group being well aware of the use of SDL as a design as well as documentation tool. The explanation offered was simple: their SDL diagrams had to be drawn by hand, a time consuming process, whereas the HLL code could be easily modified on a computer terminal. Also the diagrams, when eventually completed, often did not match the code which was changing faster than the diagrams could be produced; overall their design solution was poorly documented. For the MELBA group these problems were simply impossible since SDL and the MSD must be produced before code! The MELBA graphics software alleviated the problem of drawing neat MSD and SDL diagrams.

The MELBA group had little difficulty learning how to use the data facilities. All requirements were quickly solved by building the required types from the available library entries.

It was suggested that adding some entirely new types would make the SDL text less verbose and might result in a more efficient solution but the group made no moves towards this aim until part (B). The self-documenting text was well received and learnt very quickly. The few errors initially made were detected by the MELBA SDL/PR Generator and soon corrected.

Both groups successfully completed part (A). The ad hoc group emphasised a reduction in the number of conceptual 'states' when developing their process but the net result was the same and there seemed to be little advantage to either approach.

In the final analysis the MELBA group expended slightly less effort in part (A). Considering work directly related to production of the final product, the MELBA group spent a total of 14.5 manhours compared to 15 for the ad hoc group. The ad hoc group went through 2-3 times more drafts of their design in order to optimise it. The SDL diagrams produced by the ad hoc group were difficult to interpret due to the frequent use of cryptic, short identifiers (easier to fit into symbols). The MELBA text was far more intelligible.

The HLL code produced by the two methods was quite different. The ad hoc code was compact and efficient but very hard to read - single character identifiers were common. The MELBA code was more readable with its lengthy identifiers and rigid, unvarying format. Nevertheless, the MELBA code was approximately three times larger! This was due to the instantiation and generation of abstract operations that were ultimately never used by the queue managing process, and by the large number of modules and visibility controlling statements in the code. This weakness in MELBA has been identified and can be easily corrected.

For part (B) the overall approach by both groups was the same and all of the above comments apply equally.

The most significant feature of the MELBA group's effort in part (B) was the need to establish new types in the Library. The *system* and *process designers* drew initial diagrams and from these determined that new types were required. Based on this the *abstractions designer* proceeded to develop and add new ADT's to the library. By copying existing entries and using the AMS 'help' facilities the *abstractions designer* successfully added all the entries needed to support the new queue managing process.

The final code for part (B) reflected the same results as part (A) - the MELBA code was much larger, but more readable. The MELBA group expended approximately 24 manhours compared to the ad hoc's 37. Clearly the larger and more complicated problem was better suited to the MELBA methodology.

This experiment represented the first major attempt to use the MELBA data methodology by people outside of the development team. The ease with which the students adopted the MELBA approach to abstract types and the English-like text was particularly encouraging and it is hoped that these results will be confirmed by further experience. The data-driven design methodology enforced by the ADT library undoubtedly improved production - a clear unvarying approach to solving the problem was presented in obvious contrast to the arbitrary, haphazard approach of the ad hoc group. With larger projects the system should be even more productive.

5. CONCLUSION

The MELBA system utilizes a three level hierarchy of inputs: Module Structure Diagrams (MSDs), the CCITT Specification and Description Language (SDL) and data structure information from a library of Abstract Data Types (ADTs). MELBA strives for, and achieves, target language independence for the *system* and *process designers*. The design challenges encountered and experience gained during the development of the automatic code generation system MELBA have been reviewed.

Experience to date, although limited, indicates that MELBA will reduce development time for large complex systems, at the cost of relatively inefficient but reliable and well documented generated code. MELBA should therefore be well

suited to rapid prototyping.

6. ACKNOWLEDGEMENT

The permission of the Chief General Manager, Telecom Australia to publish this paper is hereby acknowledged.

7. REFERENCES

- [1] CCITT: "Functional Specification and Description Language (SDL), Recommendations Z.101-Z.104", CCITT Yellow Book, Vol. VI.VII, Geneva, 1981.
- [2] CCITT Study Group XI: "CHILL Language Definition", CCITT Recommendation Z.200, March 1982.
- [3] Weaver, R., Coakley, F.P., and Hill, A.J. 1978, "The SX1 Automatic Programming Project," IEE SETSS Conference, Helsinki, Finland.
- [4] Gerrand, P.H., and Nguyen, K.D.: "CADDIE: A Computer Graphics Aid for the Behavioural Specification of New Communications Facilities for the Telecommunications Network", Australian Telecommunication Research Journal, Vol. 15, No. 1, 1981.
- [5] British Telecom: "Computer-Aided Design for Software", British Telecom Research Laboratories Review 1980/81.
- [6] Farnetani, F., Giarratana, V., and Modesti, M.: "An SDL to CHILL Skeleton Transformer", CCITT SDL Implementators Forum, Florence Italy, September 1982.
- [7] Mizuhara, N., Osamu, T., Hiyama, K., and Kurosaki, T.: "A System Generator for ESS Based on the Distributed State Transition Method", IEEE Globecom, 1982.
- [8] Kurosaki, T., Watanabe, T., and Oishi, T.: "Software Development Support System for Communication Systems", Hitachi Review, Vol. 31, No. 5, 1982.
- [9] Bagnoli, P., Catoni, E., and Saracco, R. (1981): "Software Tools for SDL Documentation Handling and Software Development Methodology for Automatic Generation of SDL Documentation", IEE SETSS Conference, Warwick, U.K., July 1981.
- [10] Koning, H.: "CASA, a Computer Aid for SDL Applications", CCITT SDL Implementors Forum Contribution, Florence Italy, September 1982.
- [11] Macalleno, G.: "Drawing SDL Diagrams Is Not So Easy...", CCITT SDL Implementors Forum, Florence Italy, September 1982.
- [12] Cain, G.J., Jackson, L.N., Vesetas, R., Walter, A., and Yong, W.B.: "Computer-Aided Software Generation (the MELBA system for generating CHILL code)", IEE SETSS Conference, University of Warwick, U.K., July 1981.
- [13] Michener, J., and Van Dam, A.: "A Functional Overview of the CORE System with glossary", Computing Surveys, Vol. 10, No. 4, December 1978.
- [14] Gerrand, P.: "SDL and Its Use in Australia and Overseas, Part 1: Introduction to SDL", Telecommunication Journal of Australia, Vol. 31, No. 3, June 1981.

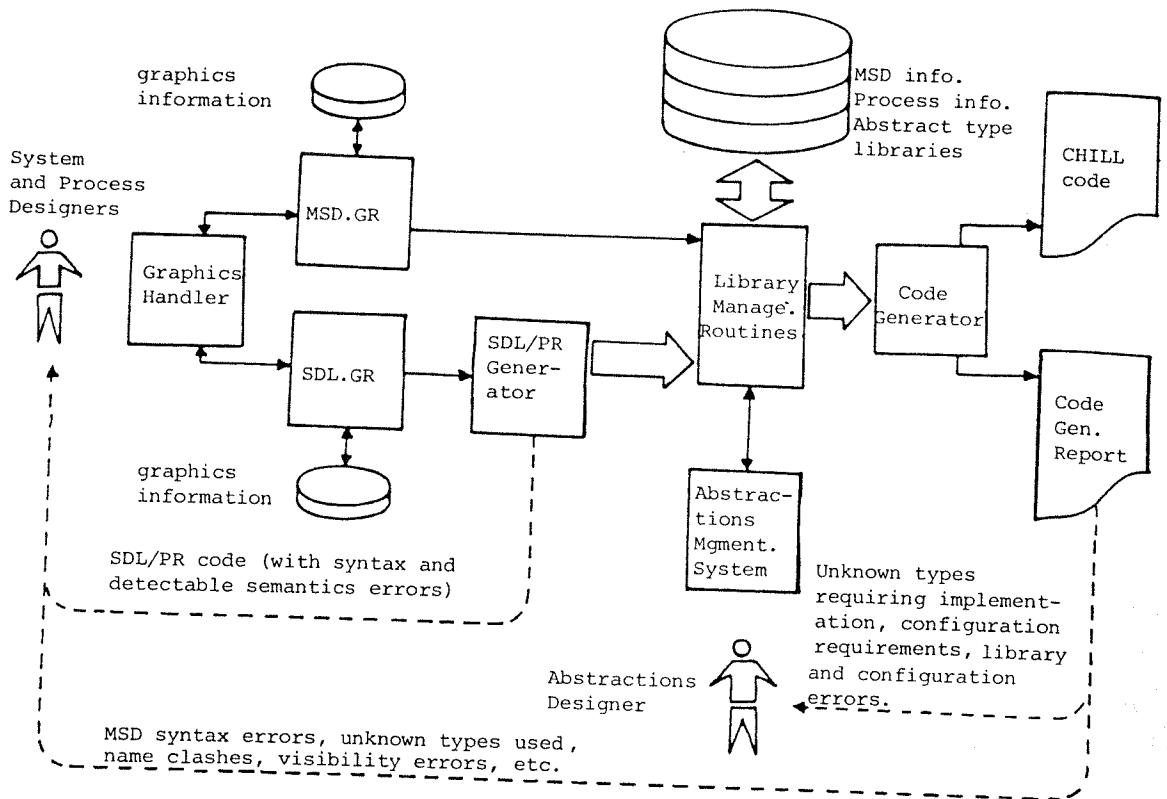


FIGURE 1 Overview of MELBA's Functional Design

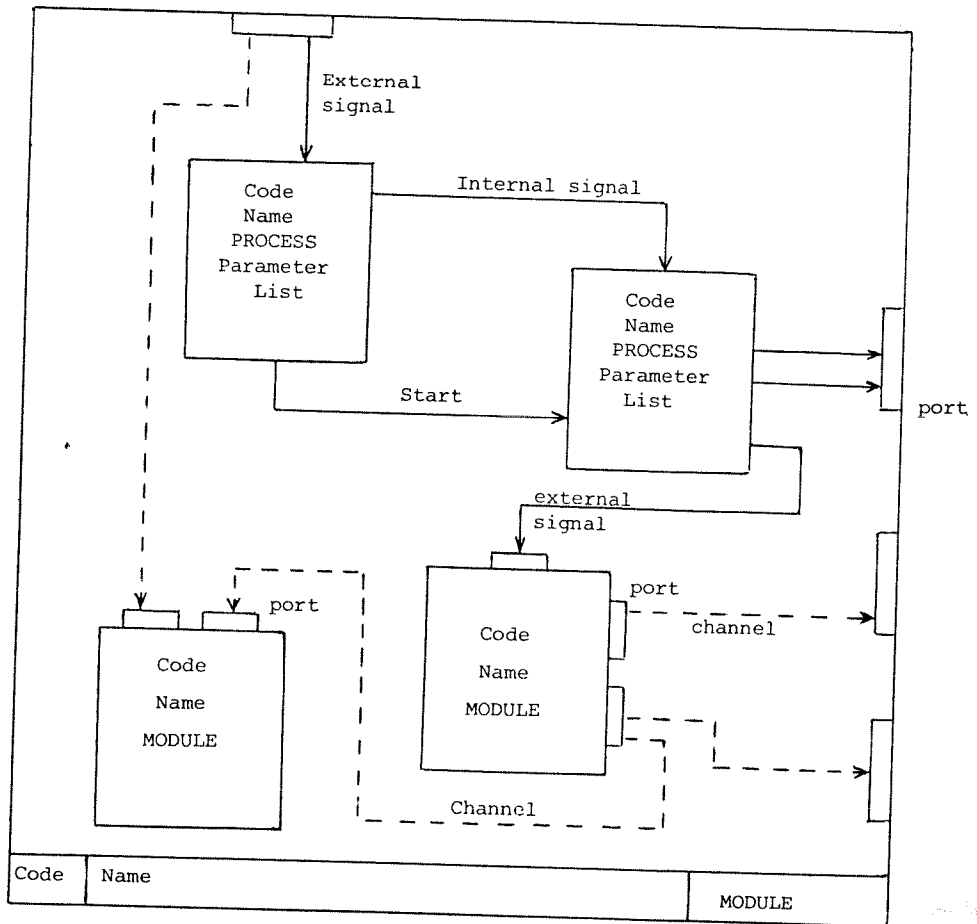


FIGURE 2 Module Structure Diagram