The European telecommunications research and development program RACE and its software project SPECS

by M. Dauphin
M. M. Marques
A. P. Mullery
P. Rodier

This paper presents the RACE program and the objectives and achievements of SPECS, a representative RACE project. The European Commission has set up the research and development program RACE for the preparation and promotion of an integrated broadband communication system in Europe. The SPECS project develops methods and techniques for the development of the complex software needed by this communication system. Its approach is the use of formal methods and maximum automation. A unique feature of this approach is the support of multiple specification languages, including the ability to mix specification languages within a given system design.

The Commission of the European Communities in Brussels and the member states of the European Community (EC) have long realized the importance of providing Europe with a modern telecommunications infrastructure. Some years ago, Michel Carpentier, General Director of Information Technology and Telecom, European Commission, said:

> Communication is central to all human activities. Effective and cheap communication services are vital to economic performance and are therefore crucial to Europe's economic and social development. Already, more than half the jobs in Europe are related to information and services and involve the use of telecommunications in all its forms. Advanced infrastructures for information exchange and services will be as dominant in the latter part of the 20th century as canal, rail, and road transport infrastructures were in the 18th, 19th, and mid-20th centuries.

This paper presents the Integrated Broadband Communications (IBC) system, a response to this need; the RACE program (Research and Development in Advanced Communications in Europe), which is to prepare and promote IBC; and SPECS (Specification and Programming Environment for Communication Software), a representative RACE project, which addresses the software aspects of IBC.

IBC

During the 1980s, the European Commission and the member states of the EC, in liaison with industry, government administrations, research centers, universities, and users, defined the objectives and the course to follow to ensure that Europe would have a modern telecommunications infrastructure. The resulting plan was approved by the European Parliament. One of its most important recommendations was to build an integrated and high-performance European system called IBC and to launch the RACE program that would develop the necessary architecture and technology. The aspect of integration is essential because it will permit cross-European country communication and avoid a proliferation of incompatible equipment and services.

IBC will be a system of terminals, cables, switches, computers, and satellites that handle telephone, television, data transmission, and services in an integrated way. It will allow a user—at work or at home—to receive and transmit information a hundred thousand times faster than with current European videotex systems, and a thousand times faster than the ISDN (Integrated Services Digital Network). Typically, the information transmitted will be a mix of high-fidelity voice, high-definition color video images, and alphanumerical data. For such transmission a communication capability of 155 megabits per second will be provided at the user's desk and several gigabits per second between switches.

The plan is to have IBC commercial operations start at the end of 1995 and have the IBC system progressively equipped with optical fibers. Fifty percent penetration of IBC access is to be achieved by 2010.

This plan is possible thanks to the development of electronic and optical technologies, especially digital and optical fiber, that will give improved performance, reduce manufacturing costs, and facilitate the required complex integration. But it will also require a significant amount of research and development.

IBC will be, in fact, the nervous system of Europe that, at the end of the 1990s, will allow people and machines to communicate easily and at reasonable cost. It will replace the current mosaic of national telecommunication networks and will be the major component of the European telecommunications infrastructure in the 1990s.

The important technological thrust of the IBC will dramatically change data processing (DP) products and services. DP business growth is highly dependent upon a modern infrastructure for telecommunications. In providing a three-orders-of-magnitude bandwidth increase and three levels of integration (voice/data/images, services, countries) for the office and the home, IBC will permit high-bandwidth, multimedia processing (images, high-definition video), distributed real-time processing, high-volume data transfer, etc. It will generate new spectrums of applications and services, with a concomitant tremendous increase in the need for DP power.

RACE

RACE is the research and development program launched and financed up to 50 percent by the EC in order to prepare the buildup and the use of IBC.[1] It is focused both on the system—the standards and techniques for development, installation, and operations—and on the participants—the telecommunication and DP manufacturers, customers, and operators.

RACE aims at pushing Europe's telecommunication operators, users, and industries to join forces in domains not directly linked with commercial products: defining the architecture and the technology base of the future network, studying the problems of integrating components from various countries and organizations, developing new integration techniques, and promoting standards necessary to build the new network.

Most of the work done to prepare the European inputs to the standards related to private and public networks, network management or more generally communication management, quality of services, management network performance, security, mobile communications, etc. is done in RACE.

The first phase of RACE, called RACE 1, is currently composed of 85 projects each having a duration of four to five years, and performed by 85 consortia with approximately 355 European organizations. This phase represents the effort of nearly 9000 people-years of high-level professionals jointly working until the end of 1992. The sec-

ond phase, RACE 2, was started in January 1992 and will finish by the end of 1994. It is composed of 65 projects and represents the effort of about 2000 people.

RACE is structured into three main parts: Part I—IBC development and implementation strategies, Part II—IBC technologies, covering technological cooperation in precompetitive research and development in key areas, and Part III—prenormative and functional integration, including the development of application pilots and tools to test and verify integrated systems.

The results of Part I are available in the public domain and are expected to make a major contribution to the development of a common European approach to the introduction of IBC. IBC development and implementation strategies are formulated by taking into account key results from Part II (technologies) and Part III (application pilots and tools).

The coherent, concise, and customized framework of the results of Part I are contained in the *Common Functional Specifications* (CFS). A second issue of the complete set of the CFS is currently available.[2] These specifications represent a broad consensus of technologists, network operators, service providers, and users.

With the goal of commercial introduction of IBC set for the end of 1995, the following main RACE milestones as defined by the European Commission have already been met:

- Mid-1988—Establishment of a set of initial assumptions on the configuration and environment of an IBC system; the number of users, their distribution and calling rates; etc.
- Mid-1989—Firm decisions on a first IBC network, its strategy for introduction, and its strategy for evolution.
- Mid-1990—Definition of a set of system architecture proposals. These proposals will be tested and validated in all EC countries.
- End 1991—Agreement on an IBC system architecture. This key product of Part I of the RACE program will be the basis of proposals for common functional specifications in international discussions on standards development.

To illustrate the program, SPECS, a representative project of Part II, is presented in detail.

SPECS

The software challenge. The implementation of IBC will require extensive, standardized, and complex software, ranging from the control software in high-speed communication switches to software providing high-level telematics services to the end users of the IBC network. The characteristics of this software are huge volume, high complexity, high efficiency, high reliability, continuous service, customer and country dependencies, multiple suppliers, heterogeneous and distributed environments, and long life. Although these individual characteristics are not specific to the implementation of the IBC, their sum means that achievement of acceptable cost, performance, and timeliness will be difficult without an improvement in currently available support techniques for the engineering aspects of the development of this software.

Many techniques with appropriate tool support for the managerial and organizational aspects of software development have been elaborated in recent years and are being applied in industrial organizations. The engineering aspects, however, can be improved to yield an important increase in the productivity of the software industry.[3] This is the domain to which SPECS is applied.

The goals of SPECS. To meet the software challenge, the SPECS project has been set up in the RACE program with the primary goal of specifying *methods* and *tools* to provide maximum automation and optimization of the software engineering of IBC software. SPECS considers the entire process from requirements and specification through design, implementation, test, execution, maintenance, and adaptation.

The project has a firm basis from which this goal can be realized: the use of *formal methods*. Formal methods are based on the use of formal languages with a precisely defined semantics for the description of the software under development. Formal methods exploit the results from academia in the domain of mathematical approaches to specification, analysis, and transformation of parts of software systems.[4-6] This exploitation can produce an important increase in development productivity and software quality.[7] However, formal methods cannot and will not eliminate the need for and use of informal methods, i.e., methods based on the use of either natural

language or semi-formal notations or both; the handling and integration of informal methods is included as an important objective of SPECS.

Software developers often reject formal methods because of the overhead in learning these methods and the assumed difficulty in applying them. This opinion is at least partially founded on developers' experience with formal methods at a time when almost no tool support existed. However, formal methods enable automation, and automation makes formal methods amenable to human use. Therefore, an important goal of SPECS is the definition and prototyping of tools for aiding the software developer in applying formal methods. These tools provide productivity gains in at least two ways:

- They free software developers from clerical tasks attached to the use of formal languages, such as checking the syntax.
- They execute complex algorithms for the validation of formal specifications (e.g., deadlock detection) that, for practical reasons, would be impossible to apply manually.

Through this extensive tool support and support for informal methods, SPECS allows a smooth introduction of formal methods in the development work.

Another important goal of SPECS is *openness*. The methods and tools defined by SPECS should be open to adaptation to particular languages and methods that are envisaged or are currently in use in developing software for telecommunications. Due to variations in roles, sizes, etc. of the organizations involved in building IBC, it is unlikely that each of these organizations will use SPECS methods and tools in the same manner. Before being used in a certain organization, these methods and tools need to be tailored to a specific context. The openness of the methods and of the tools architecture makes this tailoring feasible. SPECS assumes that IBC software development may require several methodologies. A particular methodology may be tailored by selecting the SPECS methods and tools that best suit a specific organization. The tools are to communicate via well-defined, exposed, and published interfaces. Organizations can choose SPECS tools as required and combine them with their existing in-house tools. This, in turn, protects the investment in existing tools and minimizes the cost of installing SPECS tools.

The IBC software represents an important development effort, and its lifetime is expected to exceed 20 years. Therefore, the SPECS methods and tool architecture, which are expected to be used throughout this lifetime, should be open so as to allow new developments in software engineering methods, tools, and techniques to be integrated, strengthening further the power of the environment.

The achievements of SPECS. The SPECS project started in 1988 and is to terminate at the end of 1992, the last year being essentially devoted to the consolidation and evaluation of the results. The whole project represents an effort of 320 people-years.

SPECS has elaborated *a set of methods* covering:

- The transformation of informal requirements into formal specifications
- The analysis of these specifications
- Their transformation into implementations
- The generation of test suites from these formal specifications and the execution of these test suites

These methods have been adapted to the specification languages SDL[8,9]—a CCITT (International Telegraph and Telephone Consultative Committee) recommendation—and LOTOS[10,11]—an ISO (International Organization for Standardization) standard. In both languages, a system is described as a dynamic set of concurrent processes that react to external stimuli. In SDL, a process is specified as an Extended Finite State machine; in LOTOS, processes are built from elementary communication actions and operators on processes, such as sequential composition, parallel composition, and nondeterministic choice. A major difference between the two languages lies in the way processes communicate between themselves and with the system environment. LOTOS is based on multiway synchronization, whereas SDL is based on asynchronous message passing, each SDL process having an infinite message queue attached to it. In SDL, one can moreover express the static hierarchical system structure. For the specification of data types and operations on data, both languages use algebraic abstract data types, i.e., the user defines the effect of the application of a

function or operator by means of axioms. A description of SDL and LOTOS, together with examples, can be found in Binding et al. in this issue of the *IBM Systems Journal*.[12]

To link together the individual methods it has developed, SPECS has defined a *methodological framework* that can be applied to various life-cycle models.

SPECS has defined a tool architecture, called the *SPECS architecture*, for the support of these methods; an important element of this architecture is its "common semantic layer." A tool set that adheres to this architecture is called a *SPECS environment*. Because this architecture is open, many different SPECS environments can be built. SPECS has developed one prototype environment, called the *SPECS prototype*, that includes

- Structural editors for the various types of specification documents
- A navigation tool to store and exploit relationships between specification objects
- A structural editor, a static semantic checker, and a help tool for both SDL and LOTOS
- Translators of SDL and LOTOS to a common semantic representation
- An interactive simulation tool operating on this common semantic representation but interfacing with the user at the LOTOS or SDL level, or both
- Tools to generate implementations from SDL and LOTOS specifications and a specific run-time environment to execute them
- Test support tools

SPECS has defined a *component model* to support programming-in-the-large and reuse and has instantiated this component model for various formalisms occurring in the SPECS architecture. SPECS has defined a *property language* for the rigorous expression of temporal ordering requirements and for communication with analysis tools.

In 1992, the SPECS methods are being evaluated on applicability and benefits through a pilot case study that uses the SPECS prototype.

The SPECS methods and architecture are public[13] and can be used by computer-aided software engineering (CASE) tool developers to provide commercial SPECS environments.

What follows is an overview description of the SPECS architecture, a presentation of the SPECS methods and their support tools, a discussion of the design of the common semantic layer, and some concluding remarks.

The SPECS architecture

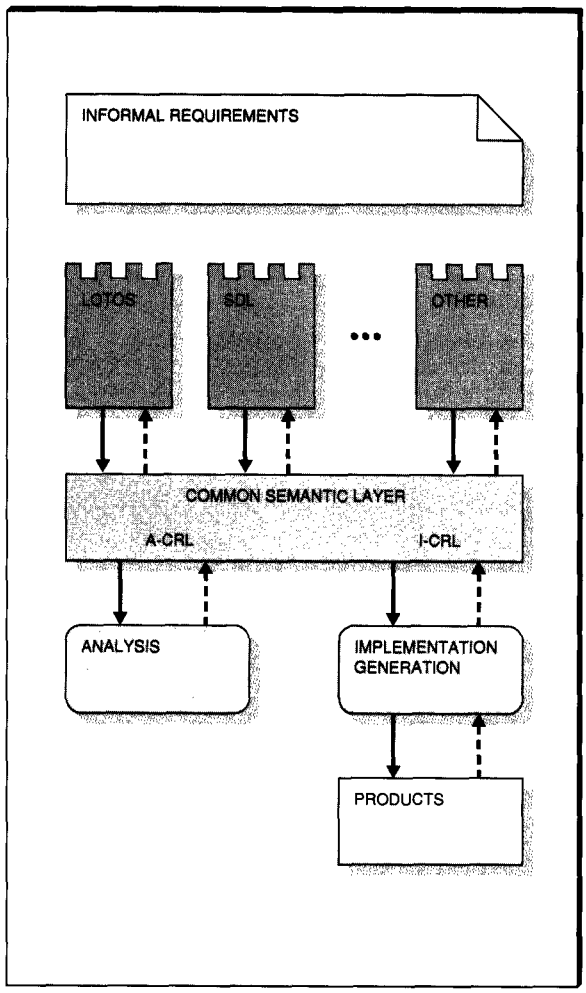
This section first presents the rationale for the design of the SPECS architecture and then outlines this architecture.

Motivation. The idea of using formal languages and methods wherever feasible and useful is at the heart of the SPECS project. The SPECS architecture therefore provides tools to support and automate as much as possible the use of formal specification languages.

At the end of the "definition phase" of the SPECS project in 1986–1987 it was concluded that none of the current specification languages such as SDL, LOTOS, and ESTELLE[14] alone fulfills all of the needs for the specification of telecommunication systems, but that most desirable features are available somewhere among the languages.[15] Therefore, the SPECS methods and architecture should not be bound to one particular formal specification language. The work of the project concentrated on SDL, widely used in the telecommunications industry, and LOTOS, which provides a higher level of abstraction. The SPECS architecture, however, was designed to be open to new languages that may emerge from the current work in academia on concurrency.

In the SPECS view of the development process, the system is taken through a succession of design steps, and in an iterative process, from an informal description to a quality product implementation. The SPECS architecture provides support for these transformations. The formalism in which the system is expressed evolves during this process from informal specification through formal specification to executable code. Wherever possible, given the state of the art, the formalisms supported by the architecture are linked to one another by formal relationships, e.g., expressing constraints on the preservation of the correctness of properties of the temporal behavior of processes. Therefore, the tools supporting the development steps in a SPECS environment should be able to cooperate at a semantic level, at which

Figure 1 Key elements of the SPECS architecture



the formal relationships can be expressed and checked.

Description. The major elements of the SPECS architecture are outlined in Figure 1.

Informal requirements. The box at the top of Figure 1 represents the tools supporting methods for the generation of formal, functional specifications from informal requirements. Formally specifying a system is not, in general, an easy task: a thorough understanding of the problem must be reached before a formal specification corresponding to the system requirements can be correctly developed. The SPECS approach for developing a specification relies on a "divide and conquer" strategy that supports and records the iterative elaboration of an understanding of the problem to aid in the actual production of the formal specification. The record of the elaboration of the understanding is useful also during later phases of the development cycle, e.g., when modifications are needed. More details are presented later.

The towers. This part of Figure 1 (LOTOS, SDL, etc.) has been adapted from generic compiler architectures; it allows "towers" to be built for each specification language. A tower is a set of tools that operates on a specification language. The core of a tower is composed of a parser and static semantics checker that builds an internal representation of a specification and of a translator to the common semantic layer described near the end of this paper. More user-oriented tower tools include: a syntax-directed editor, a report (e.g., cross-reference) generator, and a language-specific interactive help tool, providing, for example, context-sensitive access to an on-line language reference manual. The tower is also the place where existing tools of a particular specification language can be integrated.

The SPECS prototype includes towers for the standardized formal specification languages SDL (including its object-oriented extension OSDL[16]) and LOTOS. Each of these towers includes

- A syntax-directed editor and parser
- A static semantics checker
- A help tool providing explanations on the syntax and semantics of each language construct and hypertext-like navigation facilities between help messages
- A translator to the common semantic layer

An important feature resulting from this architecture is the ability to mix specification languages within a given specification. Each part of a given specification can be done in the most appropriate specification language; yet, because of a common internal representation, such mixed specifications can be analyzed, animated, prototyped, automatically implemented, and tested. The precise description of the connections between system parts, possibly specified in different languages, is expressed in the *Interconnection Language* (ICL)[17] (not represented in Figure 1). An example of an ICL description linking an SDL and a LOTOS specification can be found in Figure 2.

Figure 2 A simplified ICL description linking a LOTOS and an SDL specification

```
SYSTEM network_node
  USING
    switching_unit,                 /* expressed in SDL                    */
    control_unit                    /* expressed in LOTOS                  */
  WHERE
  ACTION control_unit.connect ! * ->
    SIGNAL switching_unit.connect(*) VIA switching_unit.c_in;
                                    /* A LOTOS event 'connect' is          */
                                    /* transformed into an SDL signal      */
                                    /* 'connect' and sent to the SDL channel*/
                                    /* 'c_in'                              */
  SIGNAL switching_unit.connected(*) VIA switching_unit.c_out
    -> ACTION control_unit.connected ! *;
                                    /* An SDL signal 'connected' sent on the*/
                                    /* channel 'c_out' to the environment  */
                                    /* is transformed into a LOTOS event   */
                                    /* on gate 'connected'                 */
ENDSYSTEM
```

Common semantic layer. Central in the architecture is the *common semantic layer*. For this layer, an abstract formalism, called the MR/CRL (Mathematical Representation and its Common Representation Language),[18] has been developed that has an expressive power exceeding that of the standard specification languages, and that is capable of being extended with many other concepts once they have sufficiently matured. It should be noted that the MR/CRL is not intended to be used by IBC software developers; tools hide this language completely from the user. MR/CRL serves as a target for translators from all of the specification languages used, and as the source for analysis and implementation-generation tools. The MR/CRL is an exposed interface of the SPECS architecture available to tool builders. Thus, to integrate new tools, it is sufficient to make them operate on MR/CRL. Through the tower-to-CRL translators, they can then be applied to all specification languages supported by a SPECS environment. More details on the MR/CRL and on some of its design issues are presented later.

Analysis. This box in Figure 1 includes specification analysis tools that operate on the semantics of the specification languages and are not dependent on their particular syntax (as opposed to syntax checkers, for example). To support all tower languages and their mixing, these tools are applied at the common semantic layer. They may either operate directly on MR/CRL or first translate MR/CRL to a representation that is more appropriate for a particular analysis tool. These internal representations and any necessary transformations are intended to remain invisible to the user. The architecture provides a specific mechanism, the so-called "hooks table," which is built up by the translators, to support reporting back the results of analysis in terms meaningful to the user, i.e., referring only to tower language concepts.

The SPECS prototype includes an interactive simulation tool that operates on the MR/CRL (but interfaces with the user at the tower level). SPECS has also prototyped a translator to finite state automata, on which the temporal logic model checker EMC[19,20] could be applied.

Implementation generation. MR/CRL is also the source for compilers of implementation languages and operating systems. However, due to the high expressive power of MR/CRL and its abstract nature (e.g., nondeterminism), an efficient compiler could not be built for the full MR/CRL. Therefore, the I-CRL (implementation-oriented CRL) has been defined (see later discussion on conflicting requirements on the common semantic layer). The SPECS architecture is not bound to a specific target operating system; it is open to allow a number of targets for the generation of code.

For the SPECS prototype, C was chosen as the implementation language, and a library of specific run-time support functions for lightweight process management and synchronous and asynchronous interprocess communication has been developed on top of the UNIX** and AIX* operating systems. The SPECS prototype includes translators from SDL and LOTOS via the I-CRL to C augmented with this library.

SPECS methodological framework

It is not the objective of SPECS to prescribe *one* particular methodology, but rather to present a *framework* in which the various methods applicable to SPECS can be assembled into a consistent set and from which individual methods can be adapted to fit the needs of a particular company. Thus, although all of the individual steps in the design process are considered, a traditional "waterfall model"[21,22] for IBC software development is not assumed, though such a model could be applied. Other models, such as the iterative[23] or spiral[24] models, can be applied as well.

A major feature of this framework is the emphasis on object orientation in the design process. Object orientation is the starting point for the definition of the component model that serves as a basis for structuring and reuse throughout the design process.

SPECS proposes methods and tools applicable to the entire IBC software development life cycle. Three aspects are highlighted in the following sections: developing specifications, analyzing specifications, and implementing specifications.

Developing specifications

The starting point of the specification generation process is an informal requirements description written in either natural language or diagrammatic terms, or both, or an already existing specification that one wants to modify, or a combination of both.

Generation of formal specifications from the informal description is concerned with three related issues:

- The actual generation of a formal specification from the informal one
- The recording of all the knowledge acquired during that process, including questions raised, decisions taken, etc.
- The handling of all the documents produced and of the relations among them, e.g., for browsing through a set of related documents

This development step is very critical, because a correct and thorough understanding of its users' needs is vital for the success of a software product; a wrong decision taken at this stage may render the subsequent development effort useless. This step may be revisited several times during later phases of the development, depending on the life-cycle model adopted. The documents produced act as an input for these phases (e.g., design, implementation, testing) and are a useful basis for the dialog between the client and the developer.

Each of the issues is now described.

Generation. A thorough understanding of the problem, which is usually described informally, is achieved by structuring and re-expressing the specification at different levels of abstraction and by analyzing these from different perspectives. To aid in the process, SPECS has structured the informal-to-formal path into three activities:

- To get an initial understanding, the developer structures the informal specification and re-expresses it in terms of concepts of the application domain (*classification*).
- To increase this understanding, the problem is analyzed by using different paradigms in order to obtain different views (*rigorization*).[25]
- To consolidate the understanding of the problem, a formal description on which the automatic transformations of the subsequent development steps will be based is developed (*formalization*).

With this strategy, the different tasks of the developer (to understand, to thoroughly analyze, to model, to handle the specification parts, and to interact or check with the client) are explicitly separated, and SPECS developed specific support for each of them.

The activities are not serial—it is not necessary (or even desirable in many cases) to complete classification before beginning to analyze the

problem or to complete the analysis before beginning to formalize. Feedback to classification and requirements is expected. The classical "waterfall" model need not be applied.

These three activities and the support SPECS has developed for them are now described in more detail.

Classification. During classification the requirements are structured using application domain concepts. There is no fixed set of concepts; SPECS has developed an object-oriented method for constructing new concepts and for storing them in a reuse library. Classification is therefore an open process. Classification takes place against both functional and nonfunctional information. The SPECS approach developed for the classification process has some similarities with Coad's and Yourdon's Object-Oriented Analysis (OOA),[26] even though it has been developed independently. The SPECS approach is more formal, especially on data aspects, and better adapted to telecommunications software. SPECS has developed a textual notation to support it (an example is given later in Figure 4); this notation, however, still requires further improvements.

Rigorization. Rigorization provides a means for the developer to increase the understanding of the requirements along the path to building formal specifications. SPECS has evaluated a number of techniques and has chosen the following ones on which to focus:

- DCFD (data and control flow diagrams)[27,28] to give a first overview of a system through the identification of the main parts of the system and their interactions and to check this understanding with customers
- STD (state transition diagrams)[28] for modeling simple behavioral aspects, and state-charts[29,30] for modeling complex behavioral aspects
- MSC (message sequence charts)[31] to represent some of the temporal sequences of interactions between processes during a particular period of time
- E/R (entity-relationship diagrams)[32] to define static globally shared data and their relationships
- ASN.1 (Abstract Syntax Notation One)[33] to model the structure of messages exchanged between processes

These techniques cover a wide spectrum of aspects, are well known, and have simple diagrammatic notations. The set chosen is pragmatic and not fixed: an organization may substitute or add other techniques, using the guidelines provided by the methodology for that purpose so that the SPECS approach is *open* to other techniques.

SPECS has developed guidelines to help the developer in building different views of the classified and informal specification according to these techniques.

Formalization. SPECS provides specific support for the exploitation of all the knowledge acquired and recorded in the two previous steps when writing down a formal specification. A complete set of guidelines was obtained by enriching existing stepwise language-driven methodologies (for example, Reference 34). These guidelines help the developer to take advantage of all of the understanding acquired during the classification and rigorization processes. Predefined mappings between concepts of the techniques used at the rigorous level and the concepts of the formal languages are part of the set of guidelines. Hypertext-like navigation facilities for accessing and browsing the information recorded during classification and rigorization help the developer to exploit the acquired knowledge.

Recording. SPECS recommends that throughout these activities all of the development and handling information (e.g., questions, decisions, links) be stored and made available to the engineer. SPECS has designed a storage model in which the classified, rigorous, and formal descriptions recorded are organized into "components" containing specification parts of the system under consideration and information relevant to the development process itself. The collected information is thus the result of the evolving specification that includes, only as a part, the formal specification resulting from this process.

Example. In order to give a sampling of the possibilities of the methodology, some extracts of the specification of an alarm call service are presented below. The example is documented with comments to draw the reader's attention to the particular aspects it is to illustrate. In Figures 4, 6, and 7, the language keywords are printed in uppercase letters.

Figure 3 Excerpts from an informal specification of an alarm call service

```
[P1]
The alarm call service (AC) is one of the services provided by the Call
Control Coordinator to a subscriber. ...

[P2]
The operation of AC in keypad mode makes use of the KEYPAD and DISPLAY
information elements inserted in adequate messages of the basic call
control. ...

[P3]
For activation of the AC service the user shall send a SETUP message with
the KEYPAD FACILITY information element with the following coding:
< servicecode > < Hour > < Minute >. Service Code = 313,...

[P4]
The Call Control Coordinator receives the messages from the user and
analyzes them. The ones related to the alarm call service are processed
by the AC processor. ...
```

Informal specification. Some extracts of the informal specification of the alarm call service are presented in Figure 3. Each paragraph is identified by a tag (e.g., [P1]) for later reference. In the SPECS prototype, these references are handled by a hypertext-like tool.

Classification. Some excerpts from the classified description are given in Figure 4. The syntax defined for the classification process introduces some formality, but informal text can be used in most of the constructs, imposing only a light structure. The notation, however, still needs to be made more user-friendly. [P1] allowed the universe of the problem (Context) to be structured into two parts: the User (i.e., the subscriber) and the Call Control Coordinator. This is recorded in the CONCEPT STRUCTURE of the CLASSIFIED COMPONENT Context, and a reference link to the relevant paragraph is made (IREFERENCE [P1]). A similar rationale is behind the structuring of Call_control_coordinator into Analyzer and AC_processor. This is recorded in the CONCEPT STRUCTURE of the CLASSIFIED COMPONENT Call_control_coordinator, and a reference link to the relevant paragraph is made (IREFERENCE [P4]). The decision to not consider the basic call control protocol messages described in [P2] is recorded, and a reference link to the appropriate paragraph of the informal specification is recorded in IREFERENCE [P2]. A part of the behavior of the Analyzer is presented, in particular because it will be used throughout the whole example. In order to have a perspective of its context, a part of the behavior of the User is also presented, as well as some interface details.

Rigorization. A DCFD of the Call_control_coordinator is presented in Figure 5. This highlights the flow of information between Analyzer and AC_processor, giving a particular view for the concept structure and interface aspects of the Call_control_coordinator classified component.

A part of a statechart-like textual description of the Analyzer is presented in Figure 6. This gives a particular view for the behavior aspects of the Analyzer classified component. This extract highlights the use of the development information: a question was raised by the analysis given by this view and recorded. Reference links to the Analyzer classified component were omitted here to simplify the notation.

Formalization. A part of the SDL description of the behavior of the 'Analyzer' (PROCESS AC_analysis) is presented in Figure 7. It was built upon the Analyzer classified component, an MSC description of the interactions between User and Analyzer (not presented here) and the statechart de-

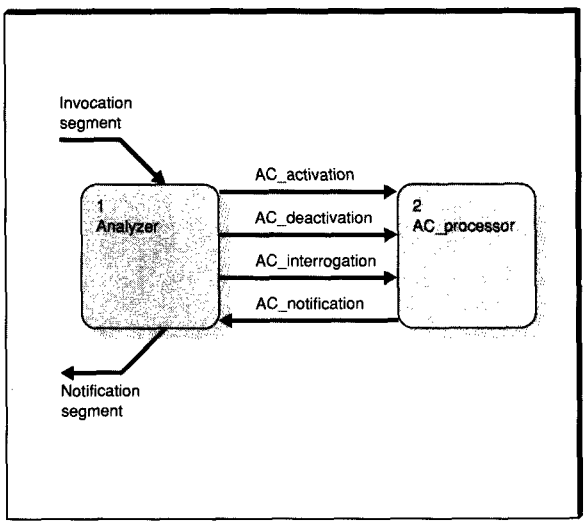
Figure 4 Excerpts from a classified specification of the informal specification of Figure 3

```
CLASSIFIED COMPONENT Context
CONCEPT STRUCTURE
  DECOMPOSITION LINKS (IREFERENCE [P1])
  -> Call_control_coordinator
  -> User
  ...
CLASS Context
INTERFACE ASPECTS
  INTERNAL INTERFACES
    'User' 'request' INTERFACES TO 'Call_control_coordinator' 'indication'
  DECISION The basic call control protocol messages (e.g., SETUP message, digits,
    control characters) are not considered in this classification; only
    the events corresponding to a successful sequence of such protocol messages
    (e.g., request) are considered (IREFERENCE [P2]).
...
CLASSIFIED COMPONENT User
CLASS User
BEHAVIOR ASPECTS
  INTERNAL EVENT request (IREFERENCE [P3], ...)
    DO ATOMIC ACTION
    SENDING OF an object 'invocation_segment' (through the 'request' output)
     conveying the 'User' 'Address' and a 'keypad_information_element'
     invoking a supplementary service in 'keypad mode'.
INTERFACE ASPECTS
  EXTERNAL INTERFACE
    request: output 'invocation_segment'
...
CLASSIFIED COMPONENT Call_control_coordinator
CONCEPT STRUCTURE
  DECOMPOSITION LINKS (IREFERENCE [P4])
  -> Analyzer
  -> AC_processor
CLASS Call_control_coordinator
INTERFACE ASPECTS
  EXTERNAL INTERFACES
    indication: input 'invocation_segment'
    response: output 'notification_segment'
  INTERNAL INTERFACES
    'analyzer' 'AC_activation' INTERFACES TO 'AC_processor' 'activation'
     ....
CLASSIFIED COMPONENT Analyzer
CLASS Analyzer
BEHAVIOR ASPECTS
ON EXTERNAL EVENT indication DO
  CASE selector IS 'keypad_information_element' OF (the received) 'segment'
  WHEN AC_activation_coding (IREFERENCE [P3])
DO ATOMIC ACTION SENDING OF an object with
   "hour", "minute" and 'User' "address"
    through 'AC_activation' output.
....
```

scription of the Analyzer, above. Reference links to those descriptions have also been omitted here.

The SPECS formalization guidelines have been used in this example. It explains the similarities that can be found between the classified, the rigorous, and the formal descriptions. For example, a simple guideline recommends mapping statechart states onto SDL states. Another guideline recommends using the statechart technique to an-

Figure 5 Data flow diagram



alyze behavior aspects if the intention is to use SDL for the formal specification.

Handling. To accompany these facilities, the SPECS architecture provides a set of functions for editing the specifications and generating reports. Editing a specification consists of two distinct activities: *creating* a specification and *modifying* an existing one. These activities are supported by browsing and navigation facilities for different representations (text and graphical). The SPECS architecture also includes reporting facilities that generate different types of reports, such as working reports to be used for further development of a specification and more persistent external reports that are part of the system documentation.

An evaluation. The methodological separation of the three activities of classification, structuring, and formalization activities is believed to be a unique solution, supporting the different tasks of the developer and making a bridge between the use of structuring methods and formal methods. In particular, the approach behind the classification activity—the analysis of the informal specification using a set of application concepts—seems invaluable in forming a link between developer's and client's terms.

Some drawbacks should also be mentioned. Openness is not always an advantage. It provides the opportunity to choose and allows, therefore, different, perhaps incompatible, solutions, which may be not desirable in large team work. Guidelines on when to proceed from classification and rigorization to formalization do not seem easy to establish, particularly when the work is carried out by distributed teams. More work is needed to refine the current SPECS guidelines on this subject.

Analyzing specifications

The analysis of specifications plays an important role in the development process. It ensures that the specification properly reflects the requirements; it detects inconsistencies and deficiencies (i.e., errors) of specifications. In current industrial practice, it is only very late in the development process that the system behavior can be checked by executing implementation code. SPECS, however, aims to provide the user with comprehensive information about the system under development very early on. This analysis contributes to early error detection and also provides guidance for further refinement in the design of a specification.

The SPECS methodology incorporates various analysis methods, supported by analysis tools in the SPECS architecture. They operate on specifications expressed in one or several specification languages. Although particular and specialized internal representations may be used for analysis, the results that are reported back to the user are tied to the elements of the source specification. These methods and tools can be classified as follows:

- Static semantics—A first level of analysis consists of syntax and static semantic analysis. For specifications developed with a SPECS environment, syntactical correctness is naturally obtained through the use of syntax-directed editors. Static semantic correctness ensures that all of the used objects (variables, types, processes, etc.) are properly defined and used. The SPECS prototype includes tools to check the static semantic constraints for SDL and LOTOS as defined by the CCITT recommendation and the ISO standard.

- Animation—This is the symbolic execution or simulation of specifications. It assists the user in gaining insight into the behavior of the specified system and in validating the specification

Figure 6 Statechart-like textual description of part of the Analyzer

```
...
STATE wait for invocation
  INPUT invocation_segment /* from User */
  DECISION service_code = 313
    (TRUE):
      DECISION is AC_activation_coding
        (TRUE):
          OUTPUT hour, minute, address
          /* AC_activation to AC_processor */;
          NEXTSTATE wait for notification;
      ENDDECISION
      ...
     (FALSE):
       /* No description of what happens if service code
       is not 313. QUESTION to be raised */
```

Figure 7 Partial SDL specification of the Analyzer

```
...
PROCESS AC_analysis ( 1, max_calls ) ;
...
STATE wait_for_invocation ;
  INPUT
    invocation ( address , service_code , invocation , time ) ;
    DECISION service_code ;
      ( AC_service_code ) :
      DECISION invocation ;
        ( AC_activation ) :
        OUTPUT AC_activate ( address , time ) ;
        NEXTSTATE wait_for_notification ;
        ...
    ELSE : NEXTSTATE wait_for_invocation ; ENDDECISION ;
    /* DECISION: it was decided to keep the state
       (wait_for_invocation) when the service code is not 313.
       RREFERENCE to statechart of 'Analyzer'/ QUESTION at state
       "wait for invocation" */
...
```

with respect to requirements. Animation can be done interactively, where the user has full control over the execution of the specification and its interaction with the environment; it can be test-case-driven or be done with randomly generated inputs. A mix of these approaches is also possible. The SPECS prototype includes an interactive animation tool that operates on the MR/CRL but interfaces with the user at the tower level. It supports the animation of mixed specifications.

- Model checkers—These tools compute the complete behavior (state graph) of a specification and then check that a property holds in all of the states. This operation, of course, is only possible for systems with a finite behavior, e.g., finite state machines such as simple protocol

machines. Model checking cannot be applied to specifications that have a very large or infinite state graph, e.g., due to the occurrence of infinite data types (such as integers). Once the desired property to be checked is specified, its complete verification for a given specification is automatic (with no user interaction as with animation), and the result (truth of a property) is guaranteed to hold. The checked properties correspond to specific requirements on the behavior of a system, e.g., absence of deadlock or a certain ordering of particular events. The user expresses these properties in a formalism developed by SPECS, the "property language." It is composed of a general scheme, which is a first-order temporal logic based on the work by Stirling and Walker,[35] and a set of language-specific basic predicates. The SPECS property language has been defined for SDL, LOTOS, and MR/CRL.

Here is an example of a property expressed in the LOTOS property language. It refers to the specification of an elevator that was developed within the project to illustrate some methods.

$$GX_{open\,!\,n}(\neg active(moving\_elevator,pid)UX_{closed\,!\,n}tt)$$

This property states that the elevator will not move if its door is open. More literally, it means that on any execution trace (G), after the door on floor n has opened ($X_{open\,!\,n}$), the process *moving_elevator* will not be active until (U) the door on floor n has been closed ($X_{closed\,!\,n}$).

- Bisimulation—The theory of formal processes enables the definition of various equivalences for processes whose behavior is "similar" despite being syntactically different; the most commonly used equivalence relation is that of bisimulation. A bisimulation checker is a tool that determines whether two processes exhibit equivalent behavior or not. Typically, a user would call this tool during the design phase to make certain that he or she did not modify the external behavior of a system in some refinement transformations. As for model checkers, current bisimulation tools can only operate on specifications whose state graph is finite.

  SPECS has developed a very efficient algorithm for a variant of bisimulation, called *branching bisimulation*, and built a prototype tool using this algorithm.[36,37]

The tools that use these techniques are integrated into the SPECS architecture. Although different formalisms are required for the different tools, these formalisms are mostly imbedded in the architecture as internal representations that are hidden from the user. The results, as mentioned earlier, are given in the terms of the specification languages.

The current state of the art in analysis also includes structural analysis, which is the most powerful but also the most complex method for validating specifications. This method consists in proving a property for a specification based on a proof system, in the way proofs are carried out in mathematics. It is currently not possible to automate these methods completely, but some support for the development and validation of a proof can be provided. However, these methods require the user to have a good knowledge of the proof technique. The application of such methods is only economically justified for safety-critical systems. The SPECS architecture does not therefore include structural analysis tools.

Implementing specifications

SPECS promotes the early formalization of a system in a formal specification language. Such an abstract functional specification describes *what* the system should do. It presents an external view ("black box") of what the system should do, but does not define how it is to be done. During the design activity, such an abstract specification is made more precise and implementation-oriented, describing the internals of the system ("white box"). Design has several aspects: architectural design defines the structure of the system implementation; functional design refines the algorithmic parts of the system. In the SPECS approach, both the black box and the white box are expressed in the same formalism. Design is thus a transformational activity.

Obviously, it is not possible to automate the design activity completely. It requires creativity and experience to tune the system architecture and algorithms to satisfy the imposed performance and reliability requirements.

The refinement process goes on until the specification is complete, consistent, and can be implemented automatically by the SPECS implementation generation tools, which compile the I-CRL into high-level programming languages. The SPECS prototype includes a compiler[12] supporting most of LOTOS and SDL and generating C code.

The currently supported specification languages LOTOS and SDL do not completely support all of the design activity. Certain types of implementation-related information, e.g., target architecture mapping, programming language and style, selection of reusable components, modularity, and connections to the operating system or the environment, cannot be expressed in SDL or LOTOS. The SPECS architecture contains specific formalisms associated with each tower language for the user to express this type of information. The code generation tools take this design information as input and orient the translation according to this information. In the SPECS prototype, these formalisms have the form of annotations to the LOTOS or SDL text.

The SPECS architecture foresees a flow of information from analysis tools to code generation tools to allow the code generation tools to exploit certain properties derived by the analysis tools to optimize the generated code. The analysis tools can automatically provide information on dynamic properties of a system (e.g., which operations are the most frequently used on a data structure or which processes communicate frequently and should thus be merged into a single implementation thread) to allow a higher degree of optimization of the implementation than currently available from automatic tools. However, it has not been implemented in the SPECS prototype.

The SPECS architecture is not bound to a specific target operating system or implementation language. Its generic structure eases the adaptation of the implementation-generation tools to a number of environments. The translation algorithms and their implementation in the SPECS prototype do not rely on any peculiar operating system facility.

The fact that the same language or languages (for example, SDL or LOTOS) are used throughout the stages of specification, design, and implementation can contribute significantly to reducing maintenance costs of these designs and implementations. In addition, the open-endedness of the SPECS architecture and the possibility of mixing different specification languages allow the later introduction of new specification languages to develop new or changed parts of an existing product. The semantic basis of SPECS is also believed to be strong enough to cover languages based on new paradigms as they mature for IBC.

The reporting-back features of the SPECS environment have the potential for indicating errors found during execution in terms of the specification languages as is done for animation. This potential, however, requires that the particular runtime operating system be adapted to access the required reporting-back interfaces of the SPECS environment.

The common semantic layer

The common semantic layer, the MR/CRL, is a process calculus based on the seminal work by Milner on CCS,[4,38] including features from Hoare's CSP[5] and Bergstra's and Klop's ACP.[6,39] In MR/CRL, a system representation is built from basic actions that can be algebraically combined using a set of process operators. The basic actions represent synchronous communication between (concurrent) processes or between the system and its environment. This communication can involve transmission of data. Processes can be composed in sequence or in parallel. The nondeterministic choice operator composes two processes and yields a process that behaves like either one of the two processes. It is possible to make the execution of process dependent on a Boolean condition, called guard. A set of actions can be encapsulated in a process to force synchronization between subprocesses of this process. Other operators provide for action renaming, abstraction, modeling of persistent data objects, and the definition of local declaration environments to support modularization. The data part of MR/CRL provides the full power of many-sorted first-order logic, and specific support for modularization, based on Module Algebra.[40]

A key issue of SPECS is centered around how to make this single common semantic layer and its language realizable and practical. That this issue is a key one was recognized from the very beginning of SPECS; it has been the subject of intensive work. This work will result in a final recommendation for a realizable and practical semantic

layer. Some of the difficulties associated with this issue are now discussed.

Distinctly different specification languages. The MR/CRL must support a large variety of tower languages with potentially very distinct semantic paradigms so that the user can choose the one that best fits his or her particular needs. To illustrate why this is a problem, consider the difference between LOTOS and SDL.

LOTOS and SDL differ on several points: communication, data variables, and system structure. LOTOS is based on synchronous rendezvous communication, SDL is based on asynchronous communication via infinite queues. In LOTOS, data variables are seen in a functional way. Once a value has been assigned to a variable, the variable is substituted (i.e., syntactically replaced) by this value, and thus "disappears." In SDL, however, variables behave as in procedural programming languages; they are objects that at a given moment have a certain value. This value may evolve over time. In LOTOS, variables are always local to a process. In SDL, there are two mechanisms to inspect variables in other processes. SDL contains specific constructs to indicate the static system structure, whereas LOTOS contains only constructs to define the externally visible system behavior.

Conflicting requirements on the common semantic layer. The common semantic layer has to fulfill several requirements, which, to a certain degree, are in conflict. The requirements of the common semantic layer in the SPECS architecture are as follows:

- The MR/CRL is used to represent specifications of telecommunication systems expressed in the tower languages. As the SPECS architecture is open to new tower languages, MR/CRL should support not only SDL and LOTOS, but also the modeling paradigms of plausible new tower languages. To fulfill this objective, MR/CRL must have a high expressive power.
- The MR/CRL is the formalism on which the analysis tools operate, either directly or via a translation to another language. Depending on the analysis technique used, different properties are desirable for MR/CRL.
- The MR/CRL is an intermediate step in the implementation-generation path from the towers. In order to enable the generation of efficient implementations, MR/CRL should contain all the implementation-related concepts that exist at the tower level, so that information that is valuable for the implementation generation is not lost in the translation from the towers to MR/CRL. For these more implementation-related concepts, however, it is often difficult to define a clean semantics that fulfills the need of analysis techniques, especially structural analysis.

To satisfy these requirements, the structure of the common semantics layer has been refined into two closely related formalisms—one oriented toward analysis (A-CRL) and the other toward implementation (I-CRL). A clear semantic relation exists between both so that analysis results obtained on the A-CRL can be interpreted in I-CRL terms. I-CRL does not contain the A-CRL constructs that can be implemented only very inefficiently or not at all because the semantics is undecidable. The following example might illustrate this point. In the A-CRL data part, it is possible to make the execution of a process depend on the truth of any first-order logic formula, which, in the general case, is not decidable. For the I-CRL data part, a more pragmatic, but less powerful, approach has been chosen. The I-CRL data part is based on a few basic data types (Booleans, integers, rationals, characters), type constructors (array, record, list, etc.), and a simple functional language to build expressions and define functions. (More details can be found in Reference 12.) A link exists between the I-CRL data part and the A-CRL data part: it is possible to define the semantics of the I-CRL data part in terms of the A-CRL data part.

Conclusion

Based on the state of the art in software engineering, the SPECS project has elaborated a coherent set of methods for the development of the huge and highly complex software for the European Integrated Broadband Communication system. These methods are based on the extensive use of formal specification languages, enabling significant automation of the software development activities. SPECS has defined an open architecture for a tool support environment for these methods and has built a prototype tool set according to this architecture.

The work done by SPECS is an important contribution to the introduction of formal approaches in

industrial environments. First applications of SPECS results are to be expected in the organizations that are members of the SPECS consortium, but the SPECS results are in the public domain and may be used by an organization interested in the application of advanced techniques to improve its productivity and the quality of its software products. Even before the availability of commercial SPECS environments, the SPECS methods may be applied with the support of some of the currently available CASE tools that represent a first approximation of the ideal SPECS architecture.

The diversity of the composition of the SPECS consortium (network operators and their research labs, telecommunication and DP manufacturers, software and service developers) and the good cooperation within the project created an enabling environment for the adaptation of advanced research results to the needs of IBC software development. This outcome illustrates one of the important achievements of the RACE program: the creation of the necessary cooperative spirit among the various telecommunication and data processing participants aimed at making the IBC become a reality in Europe.

Acknowledgment

The SPECS consortium consists of 14 partners (GSI-TECSI, CNET, STET-CSELT, PTT-RNL, IBM France, DCU, TFL, INESC, GPT, ABB, Alcatel Bell Telephone, ELIN, Alcatel Alsthom Recherche, and SESA) and several subcontractors, located in 13 countries in Europe. Despite the severe handicap of this distribution, SPECS has achieved common work based on the consensus of all partners. Participants from all of these partners have contributed in one form or another to the project; the vision, the approach, and the techniques reported in this paper are the work of all of these participants.

The work in this paper was partially supported by the European Community's SPECS RACE R1046 project. The paper reflects the view of the authors.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of UNIX Systems Laboratories, Inc.

Cited references and notes

1. *Research and Development in Advanced Communications Technologies in Europe RACE 1992*, CEC DG XIII/F, 200, Rue de la Loi, Brussels (March 1992).
2. The Common Functional Specifications are published by the RACE Industrial Consortium, Rue de Trèves 61, 1040 Brussels.
3. *Software Engineering: The Policy Challenge*, OECD, Paris (1991).
4. R. Milner, *Communication and Concurrency*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1989).
5. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1985).
6. J. C. M. Baeten and W. P. Weijland, *Process Algebra*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge (1990).
7. *IEEE Transactions on Software Engineering* **16**, No. 9 (September 1990); special issue on formal methods in software engineering.
8. *Functional Specification and Description Language (SDL)*, CCITT Blue Book, Vol. X, CCITT, Geneva (1989).
9. F. Belina and D. Hogrefe, "The CCITT Specification and Description Language SDL," *Computer Networks and ISDN Systems* **16**, No. 4, 311–341 (March 1989).
10. *LOTOS-A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, International Organization for Standardization—Information Processing Systems—Open Systems Interconnection, International Standard 8807, ISO, Geneva.
11. T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Networks and ISDN Systems* **14**, No. 1, 25–59 (January 1987).
12. C. Binding, W. Bouma, M. Dauphin, G. Karjoth, and Y. Yang, "A Common Compiler for LOTOS and SDL Specifications," *IBM Systems Journal* **31**, No. 4, 668–690 (1992, this issue).
13. The public SPECS deliverables can be obtained from: The SPECS Project Manager, GSI-Tecsi, 6 cours Michelet, 92064 Paris La Defense Cedex 52, France.
14. *Estelle: A Formal Description Technique Based on an Extended State Transition Model*, International Organization for Standardization—Information Processing Systems—Open Systems Interconnection, International Standard 9074, ISO, Geneva (1988).
15. J. Bruijning (the SPECS Consortium), "Evaluation and Integration of Specification Languages," *Computer Networks and ISDN Systems* **13**, No. 2, 75–89 (1987).
16. *Z.100 Draft for SDL-92*, Document of 04-13 December 1991 WP X/3 Meeting, CCITT, Geneva (December 1991).
17. H. Saria, H. Nirschl, and C. Binding, "Mixing LOTOS and SDL Specifications," *Fourth International Conference on Formal Description Techniques (FORTE 91)*, G. Rose, Editor, North-Holland Publishing Co., Amsterdam (December 1991).
18. *Definition of CRL/MR Version 3*, Deliverable 46/SPE/WP5/DS/A/010/b1, The SPECS Consortium (1991); see Reference 2.
19. The EMC program was developed by E. M. Clarke and M. Browne from Carnegie-Mellon University, as described in Reference 20.
20. E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions*

on Programming Languages and Systems **8**, 244–263 (April 1986).
21. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1981).
22. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of Wescon* (August 1970). Reprinted in *Proceedings of the Ninth International Conference on Software Engineering*, ICSE-9, IEEE Computer Society Press, Washington, DC (1987), pp. 328–338.
23. M. Lehman, *Program Evolution*, Academic Press, London (1985).
24. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer* **21**, No. 5, 61–72 (May 1988).
25. Rigorization means here "to analyze the problem using semiformal (also called rigorous) techniques." The output of the rigorization process is a *rigorous* description.
26. P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press/Prentice-Hall, Inc., Englewood Cliffs, NJ (1990).
27. P. T. Ward and S. J. Mellor, *Structured Development for Real-Time Systems*, Volumes 1–3, Yourdon Press/Prentice-Hall, Inc., Englewood Cliffs, NJ (1985 and 1986).
28. D. J. Hatley and Imtiaz A. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House Publishing Co., Inc., New York (1988).
29. D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* **8** (1987).
30. D. Harel, "The STATEMATE Method for System Development Automation," *Conference on Methodologies and Tools for Real-Time Systems: IV*, Washington (1987).
31. J. Grabowski and E. Rudolph, "Putting Extended Sequence Charts to Practice," in *SDL '89, The Language at Work*, Elsevier Science Publishers B.V. (North-Holland), Amsterdam (1989), pp. 3–10.
32. P. Pin-Shan Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems* **1**, No. 1, 9–36 (March 1976).
33. *Specification of Abstract Syntax Notation One (ASN.1)*, CCITT Recommendation X.208, CCITT, Geneva (November 1987).
34. O. Faergemand, "Stepwise Production of a SDL Description," in *Formal Description Techniques III*, Elsevier Science Publishers B.V. (North-Holland), Amsterdam (1991).
35. C. Stirling and D. Walker, "CCS, Liveness, and Local Checking in the Linear Time Mu-Calculus," *Automatic Verification Methods for Finite State Systems*, LNCS, Vol. 407, Springer-Verlag, New York (1990), pp. 166–178.
36. *Methods for the Transformation and Analysis of CRL*, Deliverable 46/SPE/WP5/DS/07/b1, The SPECS Consortium (1990); see Reference 2.
37. J. F. Groote and F. Vaandrager, "An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence," *Proceedings of ICALP 90*, M. S. Paterson, Editor, LNCS 443, Springer-Verlag, New York (1990), pp. 626–638.
38. R. Milner, *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag, New York (1980).
39. J. A. Bergstra and J. W. Klop, "Process Algebra for Synchronous Communication," *Information and Control* **60**, 109–137 (1984).
40. J. A. Bergstra, J. Heering, and P. Klint, "Module Algebra," *Journal of the ACM* **37**, No. 2, 335–372 (April 1990).

Accepted for publication July 15, 1992.

Michel Dauphin *Compagnie IBM France, IBM Centre d'Etudes et Recherches, 06610 La Gaude, France (electronic mail: dauphin@vnet.ibm.com).* Mr. Dauphin is a consultant on software development processes and tools at the IBM La Gaude laboratory. He joined IBM in 1984 in the Advanced Technology department of the IBM La Gaude laboratory, where he contributed to a development and execution environment for fault-tolerant telecommunication software. In 1987, he joined the IBM RACE team, where he participated in the SPECS project in the areas of semantics, analysis, and implementation of specifications of telecommunication systems. Since 1991, he has been the technical coordinator of the SPECS project. He has published several papers on the application of formal methods and languages to telecommunications software development. Mr. Dauphin received his engineering degree from the Ecole Polytechnique, Paris, in 1984. He won a First Prize in the International Mathematical Olympiad in Washington, D.C., in 1981. His main research interests are in discrete mathematics, theoretical computer science (especially concurrency), logic programming, and formal methods.

Maria M. Marques *INESC - Instituto de Engenharia de Sistemas e Computadores, Rua Alves Redol, 9, 1000 Lisboa, Portugal (electronic mail: mmm@inesc.pt).* Mrs. Marques is the leader of the Communication Software Engineering Group at INESC. She joined in 1980, where she contributed in particular to the definition of the software architecture of a medium-size digital PABX, in the scope of a Science for Stability NATO project. Since 1988 she has been involved in the SPECS/RACE 1046 project, as partner leader. She contributed to SPECS mainly in the domain of generating formal specifications from informal descriptions; she has been responsible for the "specification generation" work package since January 1991. She is also responsible for INESC participation in a RACE project dealing with service creation. Mrs. Marques received her electrical engineering degree from the IST-Instituto Superior Tecnico, Lisboa, Portugal, in 1974. She obtained the master of science degree in electrical engineering and computers in the same university, in 1985. Her main research interests are in analysis and specification methodologies, conceptual modeling, help systems, and formal methods. She is mainly concerned with the first phases of the software life cycle, on bridging the gap between customers and developers.

Alvin P. Mullery *RACE European Telecom Projects, IBM Centre d'Etudes et Recherches, 06610 La Gaude, France (electronic mail: al@vnet.ibm.com).* Mr. Mullery received a Sc.B. in E.E. in 1958 from Brown University and an S.M. in applied mathematics in 1959 from Harvard University. He joined the IBM Thomas J. Watson Research Center in 1960 as a research staff member. There he managed projects on a computer with a high-level programming language as a machine language, on an object-oriented operating system, on an expert system for medical use, and on the design of a telecommunications controller, among other projects. He has been at the IBM CER in La Gaude since 1984. There he developed a distributed, fault-tolerant operating system for telecommunications. Since 1986 he has been part of IBM France's group participating in RACE and is manager of the participation in the SPECS and ROSA projects, and responsible for

IBM's participation in other RACE projects in the areas of service creation, security, network management, and customer premises networks.

Pierre Rodier *Compagnie IBM France, Tour Descartes, 92066 Paris La Défense CEDEX 50, France.* Mr. Rodier is currently Director of European Telecommunication Projects. He began working at IBM France as a systems engineer, then was a salesman and branch office manager. After a U.S. assignment in product line management as DB/DC manager, he held various positions such as manager of forecasting and planning, manager of information system development, manager of special business operations, and business manager of line switching in IBM Europe.

Reprint Order No. G321-5491.