INTERVAL

Formal Design, Validation and Testing ofReal-Time Telecommunications Systems

IST-1999-11557

Title : Requirements Analysis Report (v2)

Author(s) : Jean-Luc ROUX, Stefan STROMQVIST, Dieter HOGREFE, Helmut NEUKIRCHEN, Daniel VINCENT, Alain KERBRAT, Iulian OBER, Vangelis KOLLIAS, Vassilis VELENTZAS.

Editor : ERICSSON

Date : 9 October, 2000

Identifier : IST/11557/WP1/07.00/ERIC/D11

Document Version : 4

Status : Proposed

Confidentiality : Public

Abstract : *(10 lines max.)*

The purpose of this document is to describe the different aspects of real-time systems to consider when defining timing extensions to SDL, MSC and TTCN languages. These requirements are gathered from industrial projects, from partners' expertise, from the work of standardisation bodies, and from inputs by the Interest Group. A classification is operated so priorities can be assigned to the most important requirements.

Project end-users present their needs with extending the emphasis to more general real-time concerns than only in Telecom. Besides tool providers are also expressing recommendations for tool support.

Deliverable D11 has 2 versions: First version was the document released on 04 August, 2000; then based on discussions with the Interest Group, a second version consolidated has been produced.

ITM LÜBECK (D), TELELOGIC (F), SOLINET (D), ERICSSON (S),
FRANCE TELECOM R&D (F), TELETEL (GR).

History

Date	Version	Comments
07 July 2000	1	First draft
12 July 2000	2	Updating Ericsson's parts
04 August 2000	3	Integration of partners' contributions and comments
09 October 2000	4	Updates for the final revision

Table of Contents

1. Objectives of this document

The purpose of this document is to collect industrial user requirements regarding real-time in the engineering process of reactive systems, from initial analysis down to system testing.

The study is based on industrial experience in projects where timing constraints are of the highest relevance. Input has been contributed from:

- Inside the consortium by the telecom equipment manufacturers and the telecom operators, as well as by the providers of engineering tools. In particular, each end-user present the current practices in his organisation, showing important links between the notations and the methodologies that are used.
- Outside the consortium by the members of the Interest Group who provide additional needs eventually for non-telecom industrial applications.

Requirements have been identified from several viewpoints: specification, design, verification, validation, testing, as the industrials have separate teams coping with the different stages of the development process.

A large part of requirements concerns analysis and modelling, focusing on necessary concepts to represent time progress (semantics of time), time constraints (deadlines), timing characteristics (execution times), resources characteristics and their utilisation. These modelling capacities are needed to perform timeliness, schedulability, performance and resource usage analyses. In liaison with these requirements, there are requirements on analysis techniques for timed systems, specially verification and testing.

It must be noted that this document does not address in detail the methodology and process sides. However the increase in complexity of real-time systems, the demand for testing as earlier as possible, and the need for a more flexible development flow impose strict requirements on the process models, the methodologies and the tools supporting the engineering of such systems.

Therefore it is important that a methodological approach accompanies the use of the extended notations and the analysis techniques. Additional requirements will be considered in this respect when designing the toolset and its application guidelines; for instance methods and tools should provide a help to formulate timed properties that the user wants to verify, as well as to interpret the analysis results e.g. the impact of a parameter on the system behaviour. More generally, a development process should provide the necessary guidance on the order (phases, increments, prototypes, etc..) in which a real-time system must be realised, defining also completion criteria for every stage.

2. Types of real-time systems

The INTERVAL project aims at offering technical solutions to real-time systems designers using Formal Description Techniques (FDTs). The needs of these designers are induced by the different problems they have to face. Current solutions for real-time systems design provide some answers for the design, validation and coding phases. However aspects such as:

- Verification in presence of timed constraints
- Prediction of system performance
- Dimensioning analysis of the architecture
- Refinement of timed constraints
- Research of timed schedules (limits)

are little considered in these solutions. They can only be handled with models that include constructs for expressing behaviour, performance, reliability, scheduling etc. It should also be noted that the definition of appropriate formalisms must be accompanied the development of analysis and simulation techniques that can give the expected results to the user.

2.1 *Characteristics of real-time systems*

A real-time system is always made of 2 parts: an application and its environment. The system must control the dynamic behaviour of the environment with response times that cannot be neglected regarding its dynamic evolution. Characteristics of real-time applications can be identified according to the following criteria:

- Is the system a reactive system or a transformational system?
- How many connections has the system with the environment?
- What is the physical architecture of the system: Mono- processor or Multi-processor?
- Does the system implement concurrent activities?
- With respect to timed constraints and delays, is the application:
  - Hard real-time
  - Soft real-time
- With respect to dependability, is the application safety critical or not?
- Does the system have high, medium or low response times?

A common element with real-time systems is the consideration of timed constraints (more generally of the time parameter) as either:

- A logical factor of the functioning (hard real-time), or
- A performance factor of the system (soft real-time), or
- A cost factor of the architecture.

Many objectives assigned to a real-time system are translated into timed constraints. Industrial applications usually impose response times i.e. bounded timed constraints that monitor in some way most of the application functions: information transfer, fault tolerance, access time, resource sharing, communication delay, clock synchronisation, emergency stop, etc..

With respect to the scheduling problem i.e. the execution of a set of inter-dependent tasks within given time and resource utilisation constraints, time is considered as a special resource needed to realise the different tasks, and allows to decide the scheduling of tasks, the assignment of resources to a task, and the detection of failures (reliability).

A number of complementary characteristics and constraints can be listed with respect to some items already given:

- Reliability/Dependability: It must be possible to guarantee a minimum mean time between failure and a useful degraded service in case of hardware failure.
- Responsiveness: The system must react to changes in its environment in a timely predictable way on all possible situations that may occur both within the environment and the system itself.
- Hard real-time: Some timed constraints impact strongly on the logical behaviour of the system e.g. making the result useless or even dangerous.
- Soft real-time: Time is only a performance factor not impacting on the logical behaviour e.g. the application is able to work under average time constraints.
- Continuous/Discrete time: Continuous time allows to manipulate time with an infinite resolution i.e. with a unit of time as small as necessary.
- Fault detection: Fault processing in a real-time context demands special techniques to detect them, recover them and evaluate their impact on the global behaviour of the application.
- Functional/Non-Functional: Non-functional requirements define constraints that the system must support while keeping on functioning. Timed constraints belong to this class of requirements
- Cost and flexibility: A real-time application must be easy to upgrade, and as inexpensive to produce, run and maintain as possible.
- Operating system: The performance of the application will be conditioned by the underlying real-time operating system and the real-time primitives that are provided.

2.2 *Specifics of telecom applications*

Within today telecom systems, covering fixed telephony, mobile telephony, and now converging with the Internet, a strong requirement of underlying protocols is that they must provide the most efficient use of the bandwidth available and they must transfer large amount of data (audio & video) in real-time. Hence end-user Quality of Service (QoS) aspects such as high-speed transfer rate, are of prime importance in this industry. New telecom standards for multimedia applications such as the ones that will be studied in the project should be able to ensure real-time traffic.

Typical real-time properties of such protocols are:

- Continuous media transmission
- Detection and recovery of losses
- Security and content identification
- Timing reconstruction of transactions

Regarding the Internet, real-time communication is a major issue that limits today the benefit from high-quality services specially due to excessive response times.

Industrial networks on the other side belong to the category of hard real-time systems, for which bad response times may imply a system failure with costly consequences.

A common characteristic of telecom applications is their distributed nature. Therefore:

- Communication and synchronisation models that include timed constraints are necessary, in order to express and validate the system execution. A distributed architecture demands structuring and scheduling methods/tools to respect the response times given in the requirements.
- Real-time scheduling of a distributed application becomes also far more complex to realise than in centralised systems. In a distributed universe, the control part executes without support of a common memory, and relies on the co-operation between the communicating processes located in different sites. The scheduling strategy must decide when and where the processes will execute.

As a summary, telecom systems are reactive systems that accept stimuli from their environment and react appropriately. Usually, several concurrent activities execute in parallel in most telecom applications; a multi-tasking mechanism is implemented to support these parallel activities.

In order to deal with all these above problems, possible solutions range around automata models, queuing networks, (semi-)formal specification languages, structuring and communication mechanisms, methods for interconnecting different formalisms. In addition support tools e.g. graphical editors, simulators, test environments etc., are required. Finally all these possible directions for investigation must remain inside the standardisation framework and must associate standardisation groups in the work.

3. Current development process

3.1 *France Telecom development process*

FT R&D is not meant to be a software manufacturer, but rather a designer and network integrator of telecom systems including a big amount of software based functions. Thus the development process used there is generally not a complete one, focusing rather on precise phases of the software life cycle : requirements/specification, global design for the early steps, and then system conformance or robustness testing. In the particular phases which are described below, languages and methods around SDL or TTCN are often well suited.

Model Description and Verification for new protocols or real time applications

Depending on the level of confidence one wants to have in his model, the initial requirements are more or less formalised, going from simple UML diagrams to completely formalised SDL.

Formalisation is especially used in the area of reliable multimedia protocols with hard real time constraints, or complex network architectures and applications (SS7, IP, IN…). The goal here is to produce good and unambiguous specifications that will then be used by a telecom equipment manufacturer. This is done by intensive model checking against requirements using SDL verification tools. We verify both if the model is correct (no deadlocks, no dead code…) and if the specification fits the needs.
Detection of IN service interactions has for example been one domain where FT R&D has made a big usage of these techniques.

Performance modelling and prediction

Validation of multimedia protocols require some events to happen in limited delays, and SDL tools do not offer enough flexibility for time control and performance analysis.
One more and more attractive technique for that is, instead of using complex mathematical modelling, to use simulation of performance models in which you describe both (simplified) functional behaviour and network or resources characteristics.
For the moment, even if we have a complete SDL functional model, the performance model has to be rewritten using some low-level programming language used by commercial tools (SES Workbench, Opnet…). Some ongoing studies aim at facilitating a partial performance model generation from an SDL model.

Conformance Testing and QoS

As said before, FT.R&D in the area of network equipment does not produce its own software, but purchases it from telecom equipment manufacturers according to more or less complete specifications. When these specifications are formal (by using e.g. SDL), it is then possible to derive conformance tests to be applied to complete systems when they will be delivered by suppliers. The ISO 9646 conformance testing method and the TTCN language are then often used for this purpose.

Other testing like robustness or QoS testing has also to be taken into account. But at the moment, these tests are mostly prepared and performed manually, due to lack of automation means and methods.

3.2 *Teletel development process*

TELETEL is following a well structured software development process in all software related projects. These are managed according to a defined life cycle, which focuses on the following phases:

- The **requirements capture and analysis** phase where the customer requirements are recorded and formally agreed. This process translates end user, marketing or customer requirements into development requirements. This phase actually occurs through the entire development life cycle, with requirements coming from several sources.
- As soon as the customer requirements have been defined (requirements specification) a **top-level design (Architectural design)** of the system begins.
- **Detailed design** (of components and modules) follows ensuring that the design of the software product is correct. Individual modules are designed using a structured approach. Before the detailed design of components and modules proceeds, a check shall be made as to whether suitable designs and implementations already exist.

  Structured design techniques are used during the top level and detailed system design to produce the System Specifications based on techniques defined by *Yourdon, Jackson, SSADM theory*. This methodology results in a clear design representation of the system functionality and module interaction through the use of:
  - Dataflow diagrams
  - Entity Relationship Diagrams
  - State Transition Diagrams.

  The detailed designs are produced using either standard drawing packages or SDL, depending on the project size.
- After the system design phase the **implementation phase** follows. TELETEL has defined proprietary coding standards and naming conventions that all engineers should follow during coding. Frequent code reviews from team co-ordinators ensure the compliance to the company standards and the ease of maintenance, portability and productivity. High Level Languages such as C and low-level Assembler are used for the implementation of the high level or the Embedded systems Software.

Testing and validation procedures are available and are followed during the design of the test plans and the implementation of testing. The testing and validation approach that is followed in TELETEL software development lifecycle is presented in the section 3.2.1 below.

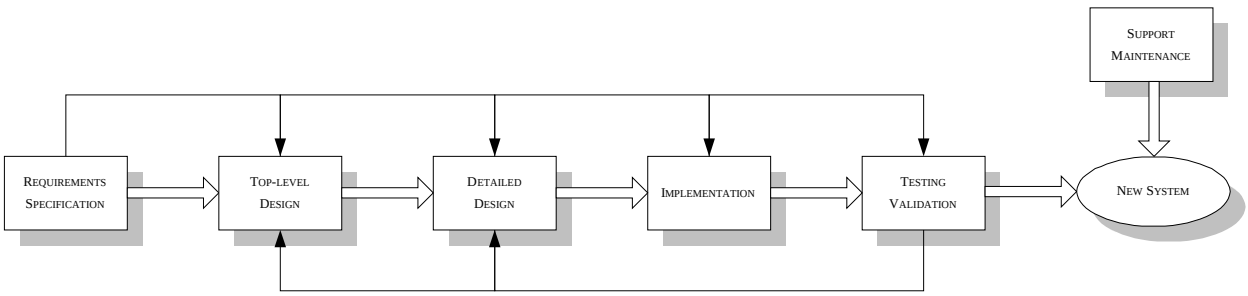


Figure 1: Current status of software development procedures

3.2.1 Testing and Validation

During validation the functional behaviour as specified in the requirements on one hand and the formal description of the service and/or protocol on the other hand are being compared. It is investigated whether the specified service or protocol is offering what was originally specified in the requirements. The testing approach that is followed in TELETEL is based on TTCN. In the Figure 2 below, the test environment for "in-house" protocol implementation validation and testing is illustrated.

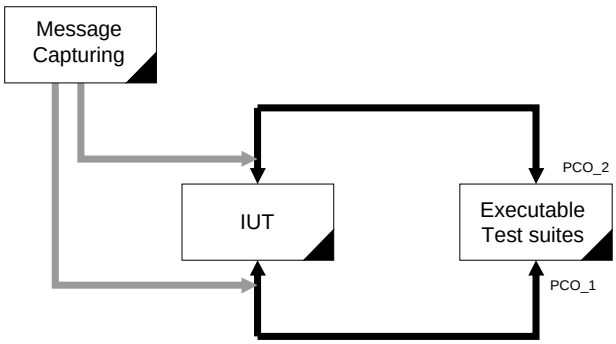


Figure 2: TELETEL test environment

Each implementation under test (IUT) module is integrated within the appropriate test environment according to the test steps and procedures that have been specified during the specification and design phases. The TTCN Execution module executes TTCN tests against the IUT, while message exchange is also being captured. The Tree and Tabular Combined Notation (TTCN) is a well-established notation for the specification of tests cases for OSI protocol conformance testing. Test cases are used for ensuring that different implementations of the same protocol specification are checked for the same set of requirements. A TTCN environment for defining, checking, compiling and executing test cases has been established allowing significant reduction of the testing phase effort, while in the same time ensures stability through standards compliance.

TELETEL is also licensing TTCN tests suites for various networks including SS7, ISDN, GSM, GPRS, ATM, etc. The test suites are also standalone products aiming at conformance testing of third party equipment (switching, gateways, terminals, etc). Depending on the protocol level, the TTCN test suite may be operating on top of protocol emulator layers (e.g. MTP-3 test suites are operating on top of MTP-2 emulator Figure 3).

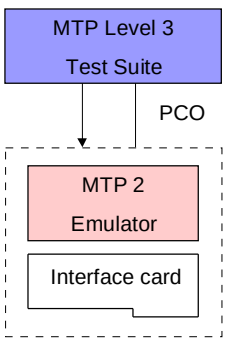


Figure 3: MTP-3 Test Architecture

3.2.2 Requirements for TTCN Testing

As described before, TELETEL uses a TTCN testing environment in order to ensure that the software implementation meets the functional requirements. Although TTCN provides a timer mechanism which allow to set timers and to check their status, the absolute and relative timing of events cannot be specified, particularly when the testing system is overloaded. The TTCN timer mechanism depends on the entire system load and performance and this fact might leverage the functional conformance of the Implementation Under Test (IUT).

The functional conformance of real time applications in many cases depends on their “in-time” reactions, especially when severe timed constraints have been considered. These requirements lead to the enhancement of conformance testing introducing particular extensions (real-time extensions) to the TTCN. Telecoms protocol implementations necessitate real-time TTCN enhancements in order to be able to be tested for conformance in any complex scenario.

A typical example of a low level telecommunications protocol layer is MTP-2. MTP-2 is a low-level protocol in the SS7 stack responsible for the reliable PDU transfer between two nodes. It builds on the HDLC method of PDU transfer by adding acknowledgement and re-send procedures. The SS7 stack was designed with dimensioning, performance and reliability in mind. The MTP-2 layer, as the lowest layer, is subject to the tightest time constraints. These constraints are much smaller than the time constraints of higher level protocols.

The test specification for an MTP-2 protocol implementation is described in the ITU-T Recommendation Q.781. In Figure 4 the MTP-2 test architecture that comprises a single PCO is illustrated. The MTP2 test suite includes about 100 primary tests for Validation and Compatibility conformance. It encompasses tests for Link State Control, Transmission failure, SU Delimitation, Alignment, Error detection and correction, the SUERM, AERM, the Congestion Control and the Transmission and Reception Control.

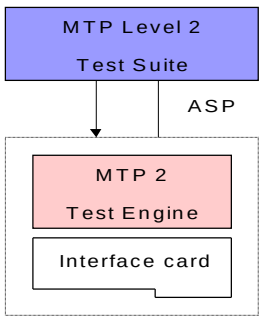


Figure 4: MTP2 Test Architecture

In the following arrow diagram (Figure 5) a graphical representation of an MTP2 test case is illustrated. The purpose of this test is to check the response to a link failure after corruption of two FIBs – detected by reception control – while the protocol is in “in service” state. The reception of two FISUs at A with corrupted FIBs when the data link is in-service state causes the link to be taken out of service.

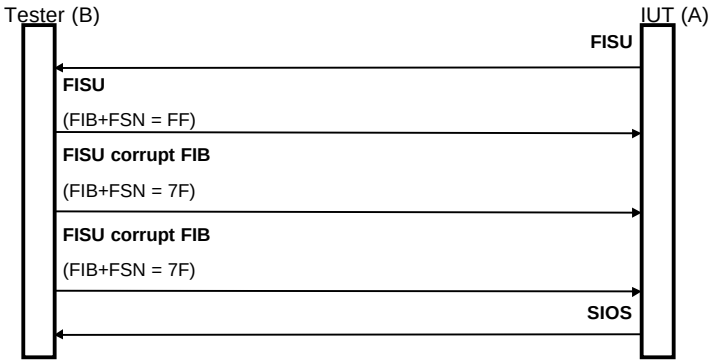


Figure 5: Arrow diagram of MTP2 test case

Whilst timers for connection establishment in the network layer may be in the range of 4-20 seconds, the MTP-2 timer T5, for example, defines a response time of 80-120 ms. For complex TTCN test scenarios (see Figure 6, with several PCOs where the ISDN, SS7 and IN interfaces are involved simultaneously) this response time is the same order of magnitude as the execution time needed to process a timeout in a simulator. These leads to scenarios incorrectly failing due to the inaccuracies introduced by the timer handling.

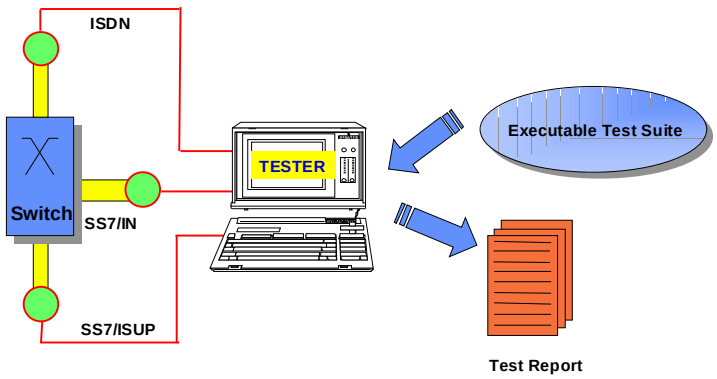


Figure 6: Complex TTCN test scenario

In this regard the test requirements for low level telecom protocols fall into the category of 'Tester is not fast enough to test the IUT' or the category 'It is not known if the tester is fast enough to test the IUT'. This is discussed in details in section 4.6.3. One requirement identified for this category of problem in the framework of Real-time TTCN is to prioritise timers used in the test suite.

However in the practical application this will reduce the problem by defining the timeframe more accurately and by reducing the inaccuracies associated with the system handling of timeouts. In this sense it provides a qualitative improvement, but does not remove the source of the problem, which, within the scope of prioritised timers, can only be achieved by using an infinitely powerful processor. For the test requirements described above, it is anticipated that the timer handling overhead can be reduced sufficiently for it to be insignificant compared to the defined time constraints.

3.3 *Ericsson development process*

Several development processes are in place at ERICSSON but most of them are only lighter modifications of this one. This process is valid for medium and large projects. Smaller ones often combine or remove some steps.

The basic project architecture and work flow can be seen inFigure 7. Note that it is possible to have several levels of sub-systems in a project. It is also quite common that a project is divided between different ERICSSON companies around the world which adds complexity on the process.

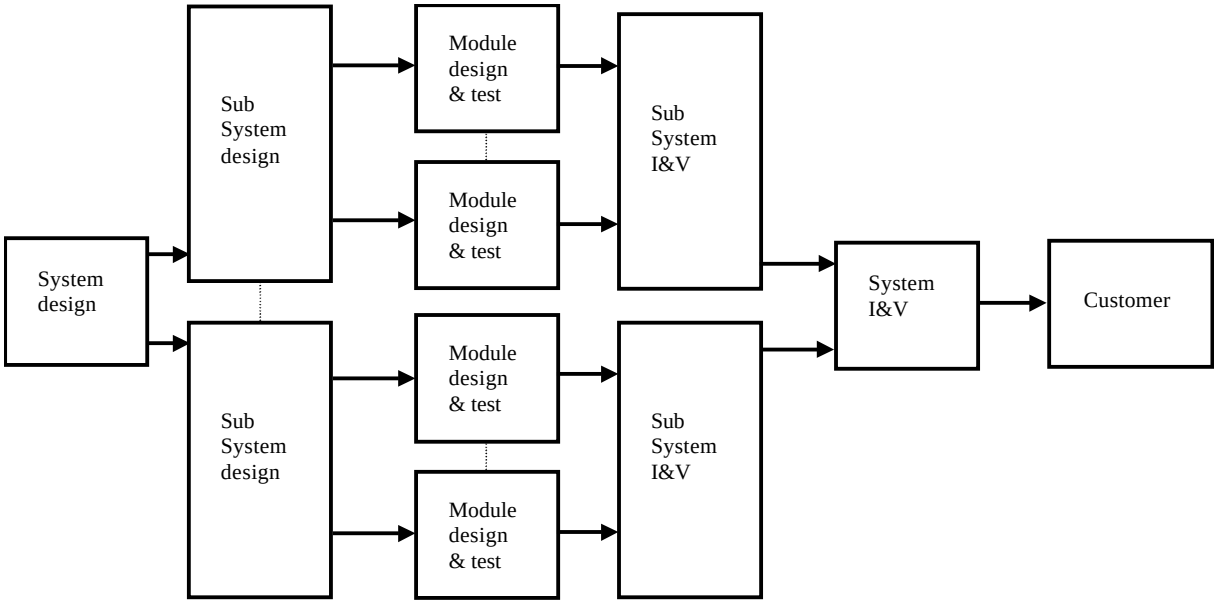


Figure 7: Project architecture and work flow

All work is done in increments and as parallel as possible (see Figure 8 below), so the described process is used several times during a project and there are also iterations inside the work flow. This puts heavy demands on good requirements and configuration handling and that is supported by tools.

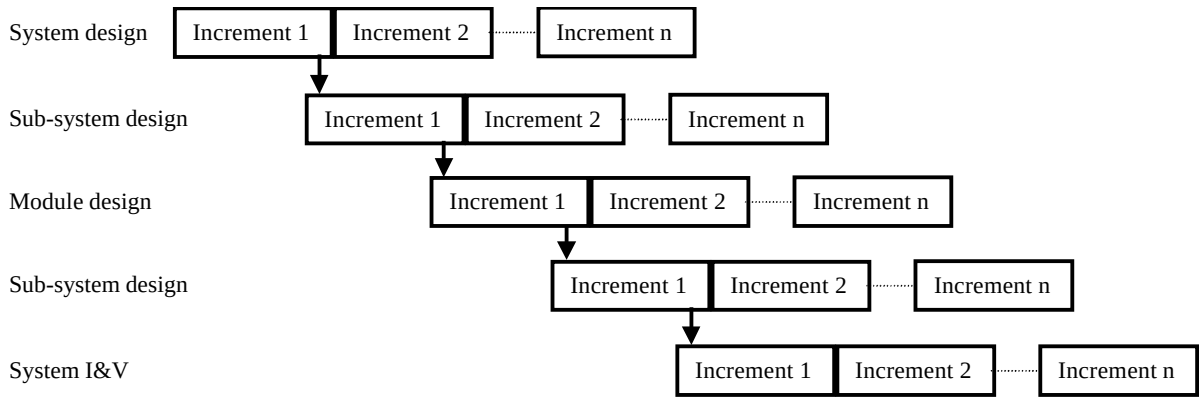


Figure 8: Incremental development process

The following sections describe each step in the development process in more detail. The languages within brackets is the most commonly used.

System design

A system group collects system requirements in a **requirements specification** (English) and makes a system design. It should be clear in this high level design (UML, SDL,MSC, English) which sub-systems it consists of and how they interface. The requirements capture is often a long process due to that it is rare that new communication protocol standards are in a stable state when a project starts.

Sub-system design

Each sub-design team refines the requirements collected from the earlier phase in a new **requirement specification** and describes their design in an **implementation proposal**. (UML, SDL, MSC, English). Some simulations are quite often done during this phase to verify that chosen algorithms works.

Module design & test

The sub-system design is divided into blocks, modules and functions. These are described in **implementation proposals** and implemented (UML,SDL,MSC,C,C++,PLEX). The interfaces between the separate software parts are also described in **inter-work descriptions**.

All implemented parts are tested by simulation or in a test environment with stubs for interfaced software. These stubs are replaced by real software during the progress of the project.

Sub-system I&V

A **test plan** and **test specification** (English, TTCN) are created based on sub-system requirements and time plans. Creation of the test lab and supporting the design departments in testability issues are also very important quite early in a project.

Integration of HW and SW starts as soon as possible and this phase includes several software releases due to found errors.

The verification against the test specification starts as soon as the sub-system is considered stable enough and the result is presented in a **test report** (English, TTCN).

System I&V

At this stage, most effort is put into verification of the total system against a **test specification** and **test plan** even though there can be some integration problems.

Also conformance tests, load tests, stress tests (TTCN, proprietary languages) and other system tests are done. Parts of these tests may be already done in the sub-system I&V phase.

Customer

It is quite common that several pre-releases are made to the customer before the official release. That makes it possible for the customer to get familiar with the system and educate own personal early.

3.4 *Standard engineering guidelines*

At ETSI, there are several kind of guidelines: the ETSI standardization process and the methodology activities framework.

The ETSI standardization process is quite formal concerning management oriented issues. When it comes to technical issues (steps 2,7,8 and 10), the ETSI process is very general [MTS00065]:

No	Step	Description
…	…	…
2	Discussion within TB/WG to determine overall/basic requirements of the standard(s)	A smaller group within a TB (usually the Sub-Technical Committee or Working Group where the proposal originated) will spend time discussing the feasibility of the proposal and will develop further the requirements for the standard(s). These requirements are rarely documented as part of the Work Item itself. If they are recorded at all, it is more usual for them to be presented in a Technical Report as the results of a pre-normative study. Once published, the link between these detailed requirements and the resultant standards can get lost. It would be very unusual for the study report to be used as the basis for subsequent validation of the standard(s). At this stage, there will be tacit agreement on the objectives of the standardization work but a consensus commitment is not normally sought.
…	…	…
7	Start of work	The serious work of developing the standard can now begin. This will either take place at ETSI within a STF or by volunteers working at their home offices.
8	Ongoing development of draft content	The draft content of a standard may contain text, tables, diagrams, formal specifications such as SDL, TTCN, MSC and ASN.1 or, more often, a combination of a number of these. Draft inputs are often produced by a number of contributors and the process may take many months to complete.
9	Regular reviews of drafts by WG meetings or by e-mail	An important part of the standard development process is to have the draft contents reviewed by interested and knowledgeable but reasonably independent individuals to ensure that the original requirements are being met and that the draft would be usable as a standard.

10	Formal validation if possible and desirable	Where formal notations are being used within a standard (even if they are not the normative part), tool-based validation of the specification model can be carried out. This can require the provision of time and other resources by the members of the responsible WG.
…	…	…

More detailed guidelines are available at the methodology activities framework [ETR298]: There exist six activities:

- **requirements collection:** to verify that all the requirements collected respect quality attributes. This is to check that these requirements are reasonably described;
- **classification:** to get a first understanding, structuring the informal specification, and to re-express it in terms of concepts of the application domain;
- **draft design:** to increase such understanding; the standard developer can analyze different perspectives of the system, using one or more models that describe the system partially;
- **formalization:** once he has obtained a thorough understanding of the problem, the standard developer actually writes down the formal specification using the formal technique of ITU-T Recommendation Z.100;
- **derivation of the validation model:** once the specification has been formalized, the standard developer provides a detailed and an executable (by the support of tools) version of the specification;
- **documentation:** from a formalized specification, the standard developer formats this according to standard rules used at ETSI (in CEN/CENELEC Internal Regulations).

As shown in the Figure 9 below, the result of performing an activity (requirements collection, classification, draft design, formalization, derivation of a validation model, or documentation) is a specification (collected requirements, classified specification, draft design specification, formal specification, validation model, ETS), consisting of a set of descriptions containing the knowledge acquired during the specification production.

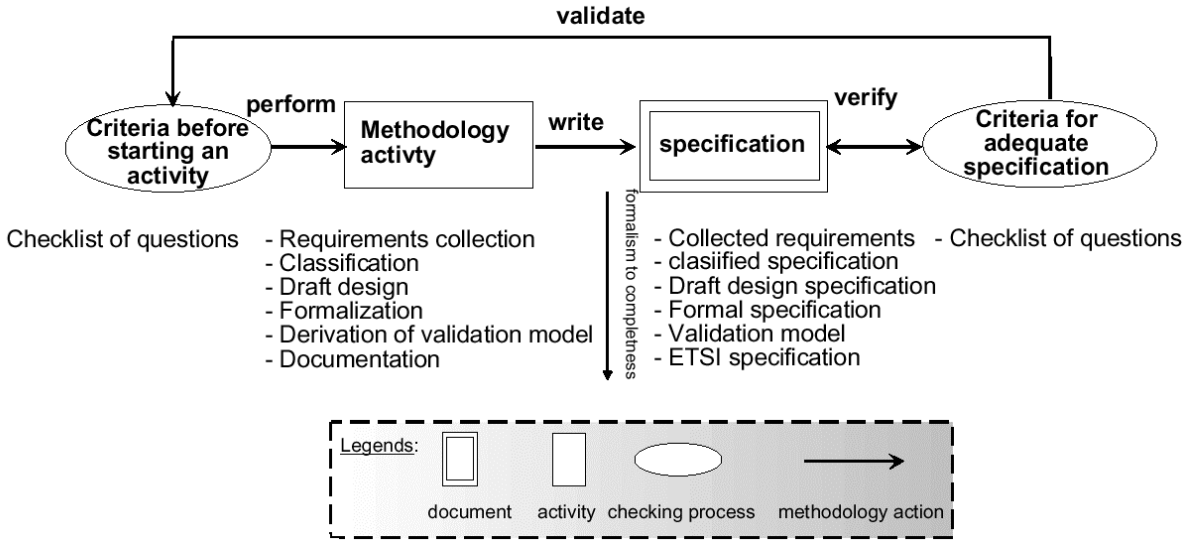


Figure 9: ETSI methodology activities framework

[ETR298] and other documents say nothing about how to handle time requirements during the methodology activities. An example, how an individual STF handles time requirements is the following:

"Generally speaking, the existing SET, RESET and timer input concepts in SDL are sufficient for most purposes. Response timers are specified using a guard (e.g. if CALL_PROCEEDING must be received within 100ms of SETUP being sent, the SETUP will be sent, a SET(NOW+100) will be executed and a new state entered to wait for either the CALL_PROCEEDING to arrive or the timeout to occur causing an error condition). Any other timing constraints that could not be expressed in current SDL would be specified in text outside the SDL model."

4. Requirements for real-time systems development

Three levels of abstraction can be identified in the construction of a real-time application:

- At the conceptual level, the application is analysed and specified, the different entities and their interactions as well as the constraints over these interactions. Constraints can be of type: coherence, synchronisation, tolerance, deadline, protection etc..
- At the architectural level, the entities in the application are distributed over different places and communication channels are established. Constraints can be of type: modularity, reusability, delays, scheduling, fault detection, reconfiguration etc.. Initial verification of the design should take place at this stage.
- At the operational level, the actual physical implementation is realised according to the architecture. The different functions must be generated and integrated on the processors, communications must be optimised, and timed constraints must be validated and tested.

The next sections review for each development stage the needs for modelling and processing timed constraints in real-time applications.

4.1 *Analysis requirements*

Existing real-time analysis methodologies use several types of models for representing timing requirements and timing information for systems analysis: timing (timeline) diagrams, time annotated MSCs, informal English, formal timing annotations on functional models (e.g. duration, frequency, delay jitters, skew).

Regarding the underlying semantics of the formal extensions proposed for SDL, the basis can be taken from the timed automata theory [ACD90] and from timed automata with urgencies [BS97], [BST97], a high-level formalism for modelling temporal properties of reactive systems. This research provides a framework for studying timed extensions to SDL, as they are investigated in the IF technology project [IF99].

Finally another direction for integrating real-time requirements within an SDL-MSC specification is expressing them in terms of some extended temporal logic [SL95], [MMG92]; the method can be applied to analyse delay bounds, delay jitters, and various QoS constraints. In the context of our project, this approach will not be further studied as we concentrate on timed extensions directly operated in the formalisms SDL, MSC and TTCN.

As far as the target technologies of the project are concerned, in the analysis phase, MSC and SDL with some extensions should offer sufficient support for capturing most timing requirements and expressing timing information available for a real-time system.

4.1.1 MSC

MSC'2000 gives the possibility to express timing requirements in an MSC. Requirements are given by intervals of time elapsing between any two events in an MSC. This facility offers a first means to express timed constraints, and must be assessed before proposing further extensions to the MSC language.

4.1.2 SDL

SDL is rich in programming constructs for manipulating time (timers, the system clock **now**), but expressing high-level timing requirements or timing information with these constructs is difficult. Examples of information that cannot be captured in SDL are:

- "it is required that the system responds to signal A within 500 to 700ns"
- "performing the action *a* takes between 200 and 500 ns"
- "signal A comes from the environment with a nominal frequency of 6,4µs, and a jitter of ±20%".

Such information may be essential:

- for the completeness of the specification of the system per se,
- for verification of certain properties,
- for the generation of sound test cases with respect to the timely behaviour of the system.

Currently, there are two reasons for which the SDL language cannot capture such information:

- language constructs (e.g. assertions, guards) for expressing certain conditions are missing,
- the semantics of existing SDL constructs is too relaxed with respect to time.

The subsequent sections discuss these two problems, and settle guidelines for the solutions which should be developed within the project.

4.1.2.1 *Assumptions on moments and duration*

Motivation

Currently, all high-level time specifications have to be modelled in SDL using programming features like timers. However, specifications with timers are imperative, and they do not match the intended semantics of assertions on moments and duration (which are just logical constraints). The drawbacks of using timers for expressing constraints are twofold:

- The description is cluttered.

  For instance, for expressing the duration of an action by means of a timer, several elements must be introduced in the model artificially: a timer, a timeout transition for expressing the minimal action execution time as a timeout-waiting time, another timeout transition leading to an invalid state for expressing the maximal action execution time.

  All these additional elements are spread throughout the model, and it may be difficult to see their relation to the initial action whose execution time they are expressing. It may also be difficult to distinguish between normal usage of SDL timers and this special usage.
- Tools subsequently using the SDL specification must distinguish between normal and special usage of timers. E.g. code generators must not generate code for the timer and the timeout transitions describing action duration. Simulation and verification tools must handle differently the invalid state used to express the maximal action duration.

For action duration in particular, several extensions to SDL were already proposed. The *Object*GEODE Simulator [OBG99] uses a syntactic extension by which one can associate an execution time (interval) to an action. Queuing SDL [MTMC99] uses a more elaborated

approach in which execution times are dynamically calculated with the help of queuing machines, so that they are depending on the amount of work and on the charge of the system.

Requirements

An extension to SDL encompassing both solutions mentioned above is needed and should be studied in the project. The extension must be sufficiently general so that a large class of timing constraints could be expressed with it, without hard-coding a pre-selected number of constraint types. The types of timing constraints that must be expressible with the proposed SDL extensions should include: action/communication duration, frequency of an action or of a signal (output/input), jitter conditions, inter-signal skew, etc..

4.1.2.2 *Semantics with controllable time*

Motivation

The current SDL semantics does not prescribe any limitations with respect to the way time passes during the execution of an SDL system (except that the value of **now** is not decreasing throughout the execution). The formal semantics contained in Z.100 maintains that an undefined amount of time may pass while a process is in a state, before the process starts executing the next fireable transition. Z.100 also specifies that "an undefined amount of time may pass while an action is interpreted".

Such a semantics is valid in the sense that actual implementations of the system conform to it. However, for simulation and verification purposes, this semantics is impractical for at least two reasons:

- Reasonable assumptions about the verified system, such as that a timer set for 10ms will be consumed before a timer set for 100ms (timers set at the same moment, by concurrent agents which are idle in the meantime), are not enforced by the formal semantics. As a consequence, execution paths which are valid regarding formal semantics but have no interest for the user will be explored by simulation and verification tools.
- The introduction of several non-realistic behaviours (execution paths) will lead to false negative verification results.

The semantics implemented in commercial SDL simulation and verification tools is different from the standard semantics in the way it handles time. Tools always make simplifying assumptions such as: action duration is 0, transitions are triggered as soon as they are enabled, etc..

Requirements

The project should investigate a modified semantics for SDL, which would handle time in a way that:

- some flexibility is allowed for modelling the responsiveness of the system and duration of actions.

- powerful analysis methods exist for the modified SDL semantics. Simulation and model checking (and, in particular, model checking of temporal properties) should be available for the modified semantics.

The second goal is essential, since one of the main advantages of SDL over other real-time modelling techniques is the availability of powerful simulation and verification tools.

4.2 *Design requirements*

The low-level design and programming constructs often employed in the construction of real-time systems represent a fairly stable and sufficient set of mechanisms. This section revisits existing concepts from the point of view of SDL, emphasising some lacks of SDL.

SDL has a good coverage of the programming mechanisms often used in real-time systems construction. Hierarchical structuring of systems and sub-systems, asynchronous communication, and timers are some of the positive real-time design and programming features of SDL. There are however several recurring real-time programming patterns that cannot be described natively in SDL.

4.2.1 Interruptive Timers in SDL

Motivation

Time dependent behaviour may be expressed in SDL using: the system clock (the implicit variable **now**), and timers (which can be set, waited upon, or received asynchronously).

SDL timer timeouts are always received as asynchronous messages. For general-purpose time dependent code this is usually fine, but it is difficult to write real timeout emergency procedures using timers. To ensure that a piece of code is executed immediately as a consequence of a timeout, the SDL programmer must first make sure that the agent handling the timer is idle when the timer is received. If this is not the case, then the process may consume the asynchronous timer message from the message queue only when it finishes its current job, which may be too late.

Requirements

The extension of SDL with a notion of emergency timer is to be studied in this project. An emergency timer is a timer whose expiration is taken immediately into account by the receiving agent. This notion would create a link between the exception mechanism from SDL'2000 and the system time, so that time passage may raise an exception.

4.2.2 Timing of Signals and Processes

There are usually less processors in an execution platform than SDL processes running in parallel. In this case, the right timing of signal processing (respectively process execution) is very important for real time systems, but there are not any possible timing methods standardised in the SDL notation. Two alternative timing methods would be useful to solve this problem:

- The first one could be to have the possibility to order "time labels" to the signals. Using these labels it would be possible to determine the order of signal processing from the queues of the different processes.
- The other solution could be a method to define the execution order of the different processes, which means that a process runs until it has any signals to be processed from it's queue and after finishing the execution, the next process to be executed is unambiguously defined.

4.2.3 Non-buffered Signal

Sometimes it is needed to send a signal to another process with "top priority" which should be immediately processed. The solution using priority levels is not suitable in this case, because other signals could have the same priority level in the queues, so the requirement of immediate processing is not met. The only solution is to use signals that do not enter the buffer of the processes i.e. it is immediately processed.

4.2.4 Other timers problems

NORTEL, a member of the Interest Group, has encountered 2 other timer problems on a current project using SDL in an IP router context. The code from theSDLs will be put on a real platform to work with existing software in the next months.

1. The timers are one-shot timers. There is currently no way of describing temporal timers that will go off every 'n' seconds, or every 'n' seconds for 'm' time. It is needed that every time a timer goes off it has to be set again for the next interval.
2. Passing timers as a data structure to a different process. Currently timers are a special type, but it is not declared using the DCL construct. So it is not really a variable.

4.2.5 Grouping Block/Process Instances

It would be useful to define groups of blocks (in any hierarchic level) so that creating or releasing an instance of these blocks happens simultaneously. And if we refer an instance of block B in an instance of block A, it means the instance of block B created on the same time.

It is also needed some 'casting' operator to overcome strict type handling on a controlled way. This would allow:

- to select from the individuals of a process the subset or an arbitrary instance where a logical expression built from the process parameters holds;
- to select from the individuals of a process the subset or an arbitrary instance where a numerical expression built from the process parameters has its extremity (min, max).

4.2.6 Atomic Transactions and Synchronisation

Atomicity and mutual exclusion may be achieved in SDL by directly inserting system calls in the SDL code. However, inserting system calls in SDL for achieving atomicity and mutual exclusion has severe drawbacks:

1. By inserting system calls in SDL, the SDL description becomes configuration dependent. Normally, one advantage of SDL is that it is sufficiently high level so that an SDL description may be mapped to different physical software configurations. Commercial code generators make use of this feature of the language.
2. External code is generally not handled well by simulation and verification tools, because such tools need a formal description of the actions to take, description which is not available for system calls.

With native SDL constructs for atomicity and mutual exclusion, a code generator could generate the right synchronisation, rollback or deadlock protection code in every possible mapping. Moreover, atomicity and mutual exclusion would be taken into account in simulation (which is not the case when using system calls), and deadlocks or other kind of errors that they may introduce could be detected earlier.

The same discussion stays valid for general purpose synchronisation code. General forms of synchronisation between SDL agents may be achieved only by using external system calls (e.g. to OS semaphore operations). In this case also, native SDL constructs would be beneficial for the same reasons enumerated above.

Requirements

The project should propose extensions to SDL which offer native SDL synchronisation and mutual exclusions. The extensions should:

- capture common synchronisation patterns (e.g. critical sections) in a simple way.
- be general enough to capture complicated synchronisation patterns (e.g. multiple-condition waiting).

4.3 *Verification and validation requirements*

The main requirement regarding verification and validation is concerned with system behaviour: model simulation and verification of model properties are needed as early as possible in the development cycle. Verification usually checks for model consistency and model correctness (safety, liveness, robustness etc.), while validation checks that the system provides the expected outputs (service, function, performance etc.). Verification/Validation results should be either a counter-example if a property is not satisfied, or some examples of correct feasible scenarios. In any case, it should be possible to replay a scenario by simulation. Considering the target technologies covered in the project, the verification and validation phases concern MSC, SDL, and possibly other requirements specification languages.

Requirements on MSC

As mentioned before, the project will rely onto the new MSC'2000 standard. However, some of the features of MSC'2000, which are interesting for real-time development, are not yet implemented by tools.

One goal of the project is to study the impact of the time-related features of MSC'2000 (time marks, time intervals, etc.) on verification and validation.

As far as possible, the tools produced or extended within the project should use time annotations from MSCs:

- analysis techniques should be developed so that SDL simulation and verification tools be capable of generating time annotated MSCs.
- model checking of properties specified by time annotated MSCs should be possible.

Requirements on SDL

The extensions proposed to SDL in order to improve the analysis and design phases (Sections 4.1.2 and 4.2.1) should be taken into account by the verification and validation tools developed or extended within the project.

Verification of general forms of temporal properties should be possible. Properties should be expressible through MSCs, through logic invariants, and possibly through other property specification languages (see next paragraph).

Other requirements specification languages

Document [INT8] compares several requirements specification languages used in connection with SDL: MSC, *Object*GEODE's GOAL language, TELELOGIC TAU's observer process language. Automata-based requirements languages, such as GOAL and TAU observer processes, are superior to MSCs in a number of respects.

The study begun by [INT8] should be pursued, in order to establish the need for such an automata-based requirements specification language in the real-time systems development process. If proven useful, such a language should be proposed and extended to cover timing aspects.

4.4 *Performance and dependability evaluation requirements*

Performance and/or dependability evaluation is an important step in the development of most real-time systems. Traditionally, performance and dependability evaluation has used a separate set of models, disregarding the functional models that serve as basis for the other steps of system development. Traditional models in performance analysis are the queuing networks, and plain stochastic processes.

Recently, academic studies have concentrated on *model-based performance evaluation*, which propose the use of the same models as for functional description, possibly with stochastic extensions [MTMC99]. Existing studies on model-based performance evaluation concern formalisms like: Petri nets (stochastic Petri nets), process algebra (stochastic extensions of CCS, CSP, LOTOS, etc.)

Requirements

The opportunity of extending SDL with stochastic modelling concepts could be studied within this project. Existing stochastic analysis methods would have to be adapted to work on SDL descriptions. Possibility of using lower-level stochastic analysis tools by transformation of SDL models is a direction for study. The combination of stochastic extensions with the other SDL extensions proposed in the project (specially timed extensions) is to be considered.

4.5 *Targeting requirements*

Automated code generation is a major advantage of systems specified with SDL and this should be preserved as much as possible, in particular for industrial purpose. The automatic derivation of applications from models is faster and less error-prone than manual coding, while keeping a close link to the original SDL model. When it is necessary to consider performance aspects at the implementation stage because this has not been done earlier in the engineering process, it is often the case that the solutions retained do not respect the system architecture, and moreover they increase the complexity of the application [MTMC99].

Regarding performance requirements and more generally non-functional properties, most should be expressed at the specification level by using the timed extensions to SDL and MSC. Furthermore, performance aspects linked to implementation such as resource requirements should be identified and formalised. These latter requirements depend on the implementation alternatives such as how a message queue is implemented. In addition to the capacity and the intrinsic performance of the resource, its service strategy is also important regarding the overall system performance. For all these aspects related with final performance at implementation, it would then be appropriate to have the possibility to quickly change and evaluate a new alternative directly at the modelling level e.g. similarly to what tends to be done in HW-SW partitioning; the best trade-off between different parameters could then be selected against the cost, the performance, the reliability, etc., of the application.

In INTERVAL, the formal specification of performance requirements will be done both in SDL itself (while describing the behaviour of blocks, channels, transitions, etc..) or in MSC diagrams (while describing execution scenarios from a user point of view). In addition timed TTCN specifications should complement the performance engineering process by providing executable timed tests derived from the formal specification and that can be run on a given implementation.

Targeting consists in transforming first the formal SDL specification into an implementation-oriented SDL description where design decisions have been made, and then generating the final executable code. In [MTMC99] the authors have identified a number of such decisions that may influence the performance of the implementation: the granularity of SDL entities (e.g. processes), the selection of the most appropriate SDL construct e.g. for an operation, dynamic creation of processes, the algorithms to implement SDL services, the specification and implementation of the data manipulated, etc.. Description of these characteristics should be investigated as part of timed extensions to SDL.

In [MTMC99], the most important implementation decisions regarding system performance have been classified in three categories:

- Software architecture
  - Application specific code
  - SDL run-time system
  - Operating system
- Hardware architecture
- Mapping of the load on the hardware

The software architecture, generally realised through a 3-layers software stack, implements the functionality and the performance of the application. The application specific code is the one generated from the SDL model; it interacts with the run-time system primitives to implement the processing described in the SDL model. The run-time system layer provides the environment needed to map the primitives for inter-process communication, process and timer management on the corresponding primitives of the given operating system.

The hardware architecture is concerned with processors, memory, and interfaces. Performance factors at this level include the generic services provided to implement scheduling, timers, communication, and interrupts handling. At this level there is another important aspect to consider regarding performance: how to map optimally the load on the hardware? This problem is not addressed in the frame of the project and is left for further study.

In conclusion, the choice of the software architecture greatly influences the performance of the system. A critical choice is to decide where the system functionality should be realised i.e. either in the application specific part, or in the SDL run-time part, or at the operating system level. An additional factor to take into account is whether the implementation will be distributed or not.

4.6 *Testing requirements*

An overall requirement analysis from the point of view of testing can be found in [HDGN00]. The main aspects are given below.

4.6.1 Definition of the testing process

We are assuming the following scenario: There is an SDL specification, which serves as a basis for an implementation. We want to use the SDL also as a basis for the development of test cases, if possible, to derive test cases automatically. The assumption is that the system contains some time critical features, e.g. some maximum response time is specified and has to be met by the implementation under test.

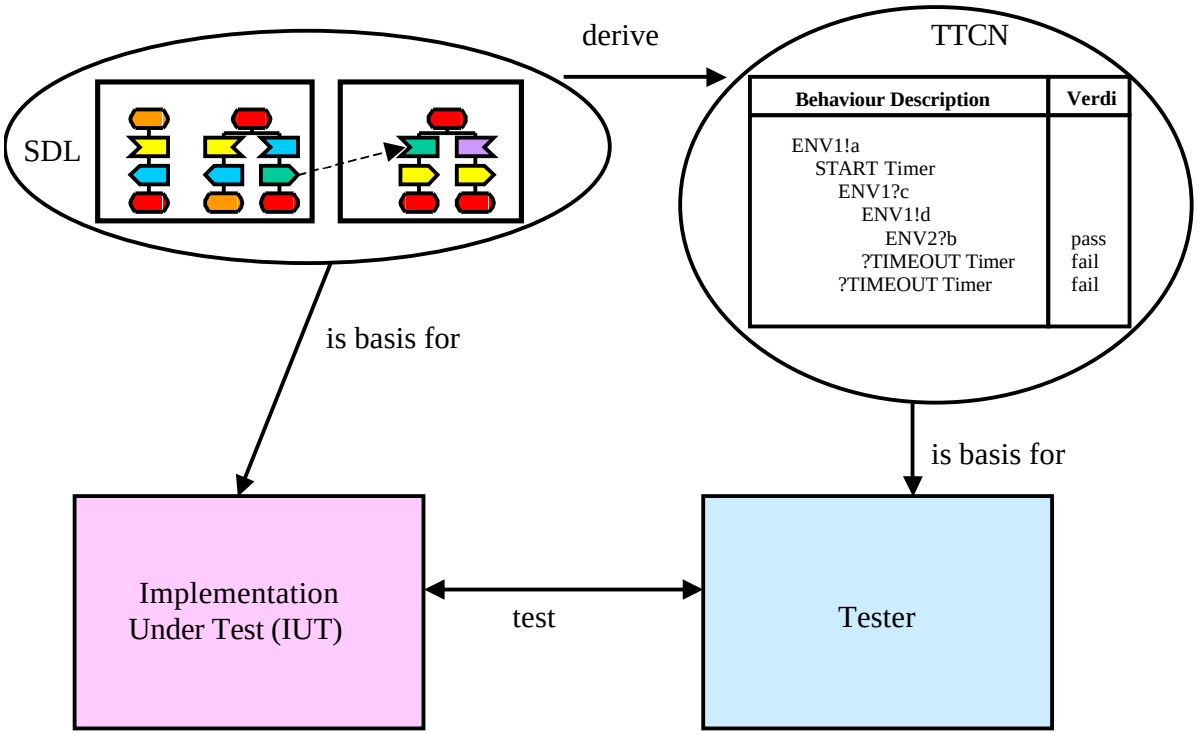


Figure 10: Testing methodology integrating SDL and TTCN

4.6.2 General requirements for the extension of TTCN

Time synchronisation
A tester may consist of many Parallel Test Components (PTCs) and a main test component (MTC). In order to be able to test for timed behaviour spread over different PTCs, a time synchronisation mechanism is necessary, i.e. each PTC has to be able to ask for the exact absolute time at any moment in order to relate test events to this time.

Qualitative verdicts
In current TTCN a verdict can only have three values PASS, FAIL and INCONCLUSIVE without further qualification. However, when it comes to time critical measurements, it may be necessary to exploit more information. For example, if an event has to happen within a time interval, it may be interesting to know how well it is within the interval, e.g. the time when an event happens may be recorded in the test report. Time measurements are normally not reproducible. Therefore, the same test case run at a different time may result in a different verdict. In general, it is necessary to run tests repeatedly and gather some statistical results.

Architecture and PCO specification
A PCO (Point of Control and Observation) in TTCN is an asynchronous communication path, or in TTCN-3 optionally a synchronous one. Beyond this there are no further details known or specified for a PCO in a test suite. In particular, it is not an issue by which means the tester accesses the IUT and how much time this connection consumes at an average. For time critical

measurements we need such information. A PCO therefore has to be qualified so that the test cases can be designed accordingly.

4.6.3 Requirements related to the speed of the tester

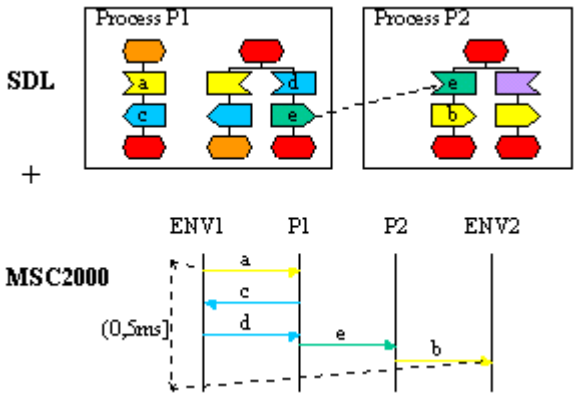
TTCN assumes that the tester is always fast enough to test the IUT. While this assumption is legitimate if only time non-critical functional behaviour is tested, for real-time applications it may not be. The processing speed of the tester may not be able to keep track with the test events that happen at the PCOs.

We want to distinguish 3 cases:

- Assumption: the tester is fast enough to test the IUT
- No assumption can be made about the speed of the tester
- Assumption: the tester is not fast enough to test the IUT

4.6.3.1 *Assumption: the tester is fast enough to test the implementation*

In this case we may not need any extensions to TTCN. The normal timer mechanism should be sufficient. The time requirements are specified with the normal MSC2000 "time interval" concept. In the following example the time elapsed between event "a" and event "b" should be less than 5 ms. Figure 11 shows the SDL specification, the MSC2000 time requirement specification and the test case that results from it in TTCN-2 and TTCN-3.



TTCN-2

Behaviour Description	Verdict
ENV1!a	
START Timer	
ENV1?c	
ENV1!d	
ENV2?b	pass
?TIMEOUT Timer	fail
?TIMEOUT Timer	fail

TTCN-3

```
Timer T:= 5ms;

ENV1.send(a);
T.start();
alt{   []   ENV1.receive(c);
            ENV.send(d);
            alt{   []   ENV2.receive(b);{
                        verdict.set(pass);stop()}
                   []   T.timeout();{
                        verdict.set(fail);stop()}
                }
       []   T.timeout(); {verdict.set(fail);stop()}
}
```

Figure 11: Deriving TTCN from time constrained SDL/MSC

We are assuming here that the tester is fast enough. We are assuming also implicitly that the connection between the tester and the IUT is infinitely fast. While the first assumption may be true, the second cannot, i.e. when the test case is designed and implemented, we have to make sure that the timeouts are defined in such a way that message travel times are taken into account.

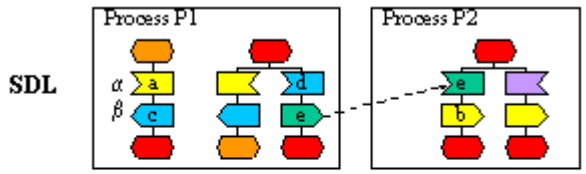
4.6.3.2 *No assumption can be made about the speed of the tester*

In this case we have to define some time constraints for the tester, i.e. we have to require the tester to execute the test events within a certain time interval, so that the IUT can be tested successfully. A proposal for this has been published in [WG 97] and is called real-time TTCN, i.e. it is an extension of TTCN by time intervals for test events.

How can these time intervals be determined?

We need to know what to test for, which implies to have a very detailed idea about the time behaviour of the IUT. Traditionally this time information, i.e. at which times can we stimulate and observe events, is not part of the SDL specification. If we add this time information to the SDL specification, we have a basis for determining the time intervals needed for testing.

Various concepts for time extension have been proposed for SDL, e.g. [MTMC99], [GMB+00a], [GMB+00b], [DBL99]. Whichever time extensions are chosen, it has to be made sure that execution time constraints for the test events can be derived with sufficient accuracy.



TTCN-2

Time	Behaviour Description	Verdict
0	ENV1!a	
	START Timer	
(1,2)	ENV1?c	
etc.	ENV1!d	
	ENV2?b	pass
	?TIMEOUT Timer	fail
	?TIMEOUT Timer	fail

Figure 12: Deriving real-time TTCN from performance SDL model

4.6.3.3 *Assumption: the tester is not fast enough to test the IUT*

In this case we know a priori that, in certain situations, the tester may be not fast enough for the IUT. We have to make sure that the time critical events are handled by the tester with higher priority than non-critical ones, without changing the test result. We have to introduce a priority mechanism into TTCN.

Let us consider the example of Figure 13. We want to test for the same time requirement as above, i.e. whether the IUT is responding to an “a” within 5 ms with a “b”. In addition we want to test, whether process P1 will respond with a “g” after it has been stimulated with “a” and “f”. We assume here that the tester runs two parallel test components PTC1 and PTC2. PTC1 performs the same test as in Figure 11. The PTC2 starts it’s operation after the IUT has been triggered with an “a” (by PTC1) and responded with a “c”.

We assume that both PTCs are running on the same tester. However, PTC2s operations are not time critical compared with the one’s of PTC1. Therefore we may give them a lower priority without changing the test result.

Particularly timer events, START and TIMEOUT, have to be executed at the tester with great care for their timing.

TTCN has to be extended by priority scheduling mechanisms.Since prioritisation changes the order of test events, we have to make sure that the test results are not changed. This problem is similar to the partial order problem, which is tackled with partial order methods, see e.g. [GPS96]. What this means for CATG is for further study.

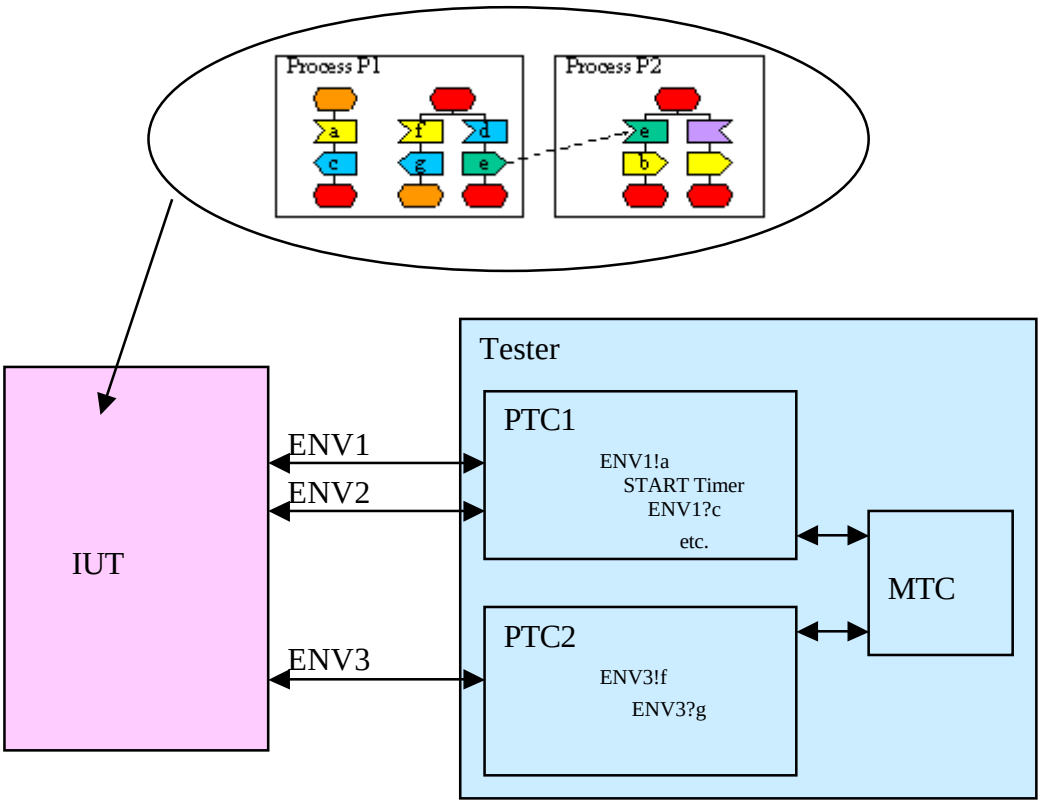


Figure 13: A test architecture with parallel test components (PTC1 has priority)

The problem to be solved here resembles the partial order simulation problem, i.e. whether and under which circumstances we can change the order of events without changing the result of

an execution. The problem is for further study, in particular whether it can be supported by tools and CATG.

4.6.3.4 *Conclusion regarding tester speed*

With the assumption that the test equipment running the TTCN test case is always much faster than the IUT, current TTCN can be used to specify real-time tests. However, the assumption about the speed of test equipment is an implicit requirement, and it may lead to wrong testing results if not guaranteed.

As long as the test equipment is faster than the application, the need of a TTCN extension for real-time testing is not mandatory. However, simple extensions such as suggested in RT-TTCN [WG97] outline possible solutions. In particular, an annotation mechanism with time labels might be a more obvious or more understandable way to express the already existing timing requirements in pure conformance testing.

4.6.4 Other extensions to TTCN (for further study)

[SBR97] argue in their paper that further extensions to TTCN are necessary to be able to specify meaningful performance tests. These extensions should allow to specify traffic load for the IUT (see Figure 14 below). The paper proposes the appropriate TTCN concepts. However, this is a different dimension of TTCN extensions and is for the moment not considered in the INTERVAL project.

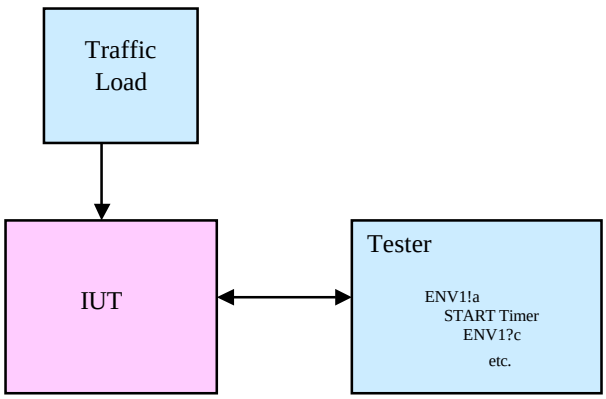


Figure 14: A test architecture with load generator (PerfTTCN example in [SBR97])

4.7 *Tools requirements*

4.7.1 General requirements

The prototypes and tools developed and extended within the project will be based on the existing tools:

- TELELOGIC's TAU and ObjectGEODE,
- SOLINET's Contessa TTCN Suite,
- VERIMAG's IF framework.

The tools developed in the project should implement, at a maximum extent possible, the language extensions anticipated by the previous sections of this document.

The possibilities of existing tools such as modularity, interoperability, flexible language implementation, etc. should be preserved whenever possible.

The new tools should be capable of handling both timed specifications (new extensions) and standard specifications (specifications using only the present features of the languages). When handling a timed specification, switching between a timed mode, where the extensions are taken into account, and an "untimed" mode, where the extended information is disregarded, should be possible.

The subsequent sections contain additional, tool-specific requirements, which complement the requirements on tools.

4.7.2 Edition tools

The concepts and mechanisms identified as necessary extensions to the languages (SDL, MSC, TTCN) will be submitted to the respective standardization bodies. However, in a first step, the editors for the concerned languages may implement only lightweight forms of these extensions. This is in order to minimize the effort spent on interface extensions, and to concentrate resources on more important tasks such as the development of back-end tools and the work on extensions to the standards.

Therefore, the editors will have the choice between:

1. *implementing the extended versions of the languages natively*. This is unlikely, due to the reasons mentioned above, and to the fact that standardized versions of the languages will be available late into the project lifetime.
2. *using special comments which reflect timing information*. All the languages support in some way informal user comments. These comments could contain the necessary timing annotations, and could be interpreted by the back-end tools.
3. *Using a separate description of timing information*. Separate description of timed aspects requires an appropriate notation, and tools to edit, compile and integrate it with the system's description.

It is possible to have a combination of these three approaches as a result of the choices made for timed extensions.

4.7.3 Simulation tools

The existing simulation tools encounter a series of problems due to the discrete-time semantics they use, and the way they handle temporal non-determinism (esp. non-determinism of the system environment). As a consequence, in some cases, certain execution scenarios that can happen in a real implementation of a system do not appear during simulation.

Compared to the current simulation tools, the extended simulation tools should provide a better coverage of the execution scenarios which can occur in a real implementation of the simulated system. The limits of the extended simulation tools should be studied and stated explicitly.

4.7.4 Verification and validation tools

The extended tools should be capable of verifying basic or more complex temporal properties, including:

1. *absence of deadlocks*. Existing SDL verification tools are capable of checking this property. However, with temporal extensions, special cases of deadlocks due to the non-satisfaction of temporal conditions may be introduced. These types of deadlocks must be detected by the extended verification tools.
2. *invariance properties*. Existing SDL verification tools are capable of checking such properties. For timed specifications, however, verification of temporal invariants may be necessary and should be implemented by the tools. Examples of temporal invariants include:
   - a certain event a is always handled within δ time units,
   - the time span between two particular states of the system always falls within some interval.
3. *non-zenoness.* For a timed specification, a zeno path is an *infinite* execution path during which time does not progress beyond a finite limit. Zeno execution paths indicate an incompletely or incorrectly modeled system, since the system is allowed to act infinitely fast and handle an infinite amount of events within a finite duration. Zeno execution paths should be detectable with the extended verification tools.
4. *linear properties.* The verification of more complicated temporal properties of systems usually relies on a complementary formalism such as temporal logic or some form of timed Büchi automata. The extended verification tools should be capable of verifying timed linear properties as complex as those expressed with timed linear temporal logic formulas. There is no restriction over the property specification language to be used. However, considering the requirements below, the most likely candidate ObjectGEODE's GOAL language, suitably extended for expressing timed properties.

The verification and validation tools should also support:

- time annotations existing in MSC'2000. The tools should be capable of both:
  - generating time annotated MSCs,
  - using time annotated MSCs as input for verification.
- an extended version of the GOAL language, capable of capturing temporal properties of SDL systems.

4.7.5 CATG tools

Current SDL analysis and simulation tools are able to generate test cases automatically from an SDL specification. The main advantage in addition to faster and cheaper processing, is that automatic test generation or Computer Aided Test Generation (CATG) guarantees consistency between the formal specification and the test cases run on the implementation [SEG00]. For that reason it seems important as part of the timed extensions proposed for SDL, MSC and TTCN that CATG be still operational, certainly with some adaptations. Therefore a strong requirement with respect to those extensions is that the three formalisms should be extended in a related and coherent way, so that no inconsistency happens between the different

specifications i.e. the formal SDL model of the system, theMSC model of test purposes and the TTCN test cases, when they are processed together as in CATG.

It should be noted that a CATG adapted for timed extensions should be compatible with previous timers mechanisms in the languages. In particular a number of test timers are already used in some test generator tools such as: a timer to guard expected SUT output events, a timer to check situations where no output is expected, a timer to check a maximum time to execute an implicit send, or a timer to wait for the expiration of timers in the SUT.

Within current CATG techniques, it is also possible to use observer processes to help test generation. Because an observer process has direct access to internal elements of the SDL specification e.g. timers, it will be necessary to study the impact of timed extensions for the observers with respect to simulation and test generation e.g. change of a timed constraint or pruning a timed path in the state space.

Tools for computer aided test generation (CATG) have to be able to cope with real-time requirements which are specified using time annotated SDL andMSC. Test generation may both be based on direct translation of an MSC describing real-time requirements into TTCN and on derivation of TTCN from an time extended SDL model. In the latter case, CATG is based on the state space exploration techniques performed by the SDL simulation tools. Therefore, the underlying simulator of the CATG platform has to be able to provide a time annotated state space for subsequent representation of conforming paths using TTCN.

Output of CATG should be either TTCN-2, TTCN-3 or time extended TTCN-3 depending on the state of the other tools which use TTCN as input.

4.7.6 TTCN executor benchmarking

The TTCN language (both versions 2 and 3) itself is suitable for describing real-time test cases. The usage of timers and verdicts enables the checking of real-time constraints.

However, if the test executor equipment (the Means of Testing) is not fast enough, it can not communicate with the IUT properly so the testing becomes meaningless. The most important advantage of reactive testing in contrast to log analysis is that the tester can explicitly control the behaviour and state transitions of IUT in order to achieve better coverage. The performance bottleneck, for example, can be the delay between two successive send events or the reaction time (the time between receiving a message and sending the response).

4.7.6.1 *Protocol-specific benchmarking*

The optimisation of the test executor's speed is out of scope of the TTCN standard. These performance requirements depend strongly on the tested protocol, so the real-time test suites must formalise them somehow.

In a real-time test suite a couple of test cases can check the test executor rather than the correct behaviour of the IUT. A fail verdict of these 'benchmarking' test cases means that the corresponding test executor is not able to run the real test cases correctly, i.e. their verdict won't be correct. These test cases must be run with the protocol adaptation layer and with a hardware or software loop-back (if possible). We think these preliminary measurements can be done using standard TTCN constructs, so there is no need for language extensions.

An example is given below, which checks whether the test executor is fast enough to send two consecutive messages within a duration specified by "requirement_time", in TTCN-3:

```
testcase TC001() runs on mytc
{
    timer t1 = 1 s;
    timer t2 = 1 s - requirement_time;
    t1.start;
    PCO.send( message1 );
    PCO.send( message2 );
    t2.start;
    alt{
    [] t2.timeout { verdict.set( PASS ); }
    [] t1.timeout { verdict.set( FAIL ); }
    }
}
```

4.7.6.2 *Protocol-independent benchmarking*

Since the most of the program code of TTCN executors are generic and protocol-independent, it won't be a good practice to measure the low level performance characteristics of the test executor in every real-time test case. The steps of the test execution can be broken down into atomic and independent operations (e.g. the assignment or matching of a single message field, a timer or verdict operation, etc.). The execution time of such an operation depends on the test executor only. If these atomic times are known, the total execution time of a complex test case operation can be calculated.

These atomic operations must be specified and standardised. They can be measured by the tool vendors or with special test cases. Finally, the speed characteristics of the test executor in a given HW/SW environment can be described in a simple table. From the real-time requirements of the test suite an another table can be derived. The comparison of two tables will decide whether the executor can run the test suite or not.

There are still two problems with this atomic benchmarking:

- The way of performance estimation and calculation is different for compiler based and interpreter based test executors.
- The processing time of the protocol adaptation layer was not considered.

4.7.7 Tools integration platform

The envisioned platform for timed extensions will be based partly on existing tools, partly on tools under development. The existing tools are: TAU Suite and ObjectGEODE from TELELOGIC, CONTESSA Suite from SOLINET and VERIMAG's IF framework.

The tools under development are part of Next Generation of TELELOGIC SDL and TTCN tools, developed for SDL-2000 and TTCN v3.

Experiments for timed SDL simulation will at first be performed on two prototypes:

- IF framework developed by VERIMAG,
- Timed extension of TELELOGIC ObjectGEODE's simulator for SDL.

A complete picture of the prototype platform is given below:

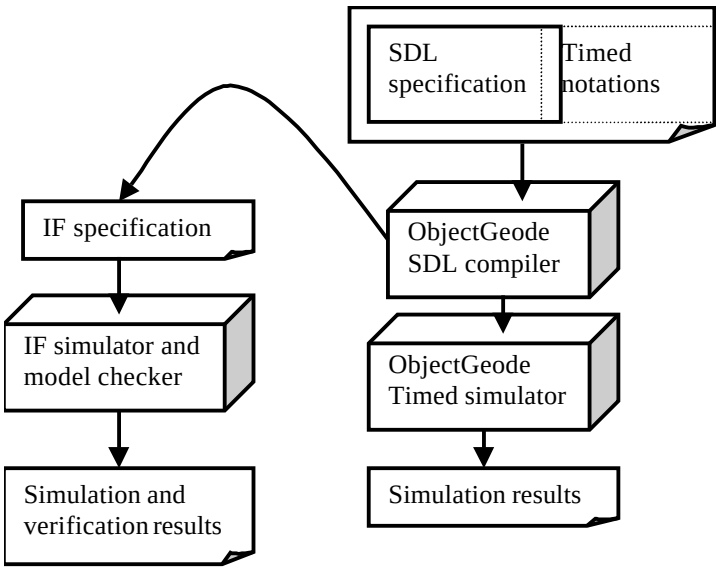


Figure 15: Timed SDL simulation and verification

Computer Aided test Generation will be developed gradually using TELELOGIC Next Generation toolset. This toolset will also be open to external tools such as VERIMAG's IF.

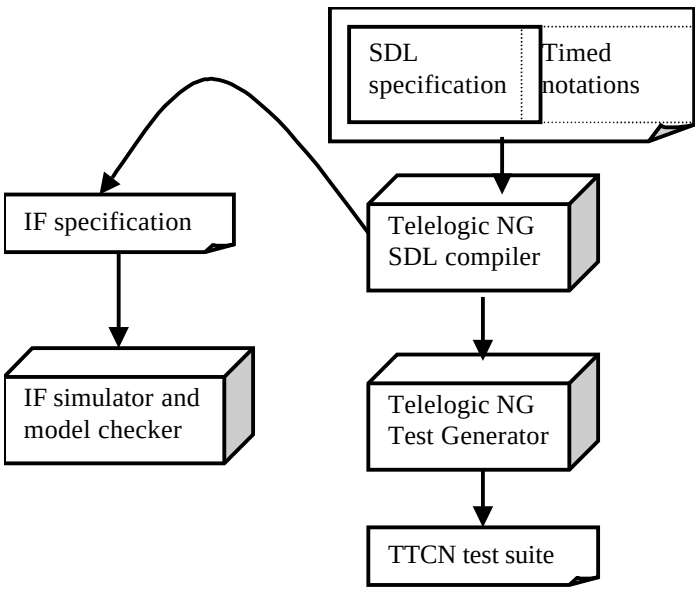


Figure 16: Computer Aided Test Generation (CATG) platform

Experiments for Timed SDL-TTCN co-simulation will be first performed with the following tools:

- TELELOGIC ObjectGEODE's SDL simulator augmented with support for timed extensions
- TELELOGIC TAU TTCN Suite coupled in a co-simulation backplane

The final co-simulator will be based on TELELOGIC Next Generationtoolset and SOLINET CONTESSA Campaigner for TTCN (Figure 17).

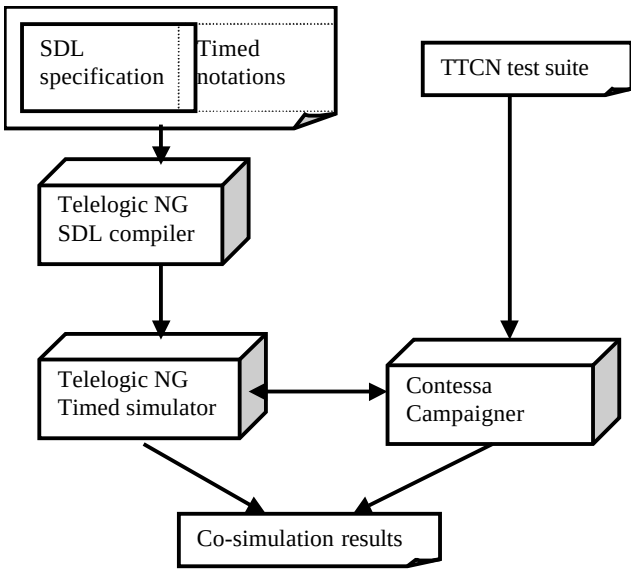


Figure 17: Timed SDL-TTCN co-simulation platform

5. Classification of requirements

Extensions for SDL	*Classification*
Analysis requirements	**Desirable**
Assumptions on moments and duration	Desirable
Semantics with controllable time	Mandatory
Design requirements	**Desirable**
Interruptive Timers in SDL	Mandatory
Timing of Signals and Processes	Desirable
Non-buffered Signal	Desirable
Other timer problems	Desirable
Grouping Block/Process Instances	Optional
Atomic Transactions and Synchronisation	Desirable
Verification and validation requirements	**Desirable**
Time-related properties expressed in SDL	Desirable
Join verification with timedMSCs	Desirable
Performance and dependability evaluation requirements	**Desirable**
Extending SDL with stochastic modelling concepts	Desirable
Targeting requirements	**Optional**
Mapping functionality to software architecture	Optional
Mapping load to hardware architecture	Optional

Extensions for MSC	*Classification*
Analysis requirements	**Desirable**
Implement MSC'2000 timed features	Mandatory
Automata-based requirements language	Optional
Verification and validation requirements	**Desirable**
Properties expressed as timed-annotatedMSCs	Desirable
Generation of timed-annotatedMSCs	Desirable
Using time-annotatedMSCs as input to timed SDL verification	Desirable

Extensions for TTCN	*Classification*
General requirements for the extension of TTCN	**Mandatory**
Time synchronisation	Mandatory
Qualitative verdicts	Mandatory
Architecture and PCO specification	Mandatory
Requirements related to the speed of the tester	**Desirable**
Extension of TTCN by time intervals for test events	Desirable
Introduction of priority scheduling mechanisms into TTCN	Desirable
TTCN executor benchmarking	Desirable
Other extensions to TTCN for specifying performance tests	**Optional**

Tools Requirements	*Classification*
General requirements	**Desirable**
Timed / Untimed specification mode	Desirable
Edition tools (support of timed extensions)	**Mandatory**
Simulation tools	**Desirable**
Verification and validation tools	**Desirable**
Temporal invariant verification	Optional
Infinite execution path detection (no time progress)	Optional
CATG tools	Optional
TTCN executor	**Desirable**
Protocol-specific benchmarking	Optional
Protocol-independent benchmarking	Optional
Tools integration platform	**Desirable**
Connection to external tools	Desirable
Timed SDL-TTCN co-simulation	Desirable

6. Conclusions

In this report, the project has examined the characteristics of real-time systems, in particular in the Telecom domain, in order to identify the major requirements of their development process. The emphasis has been put on time requirements, their representation and their processing. In addition, the current development process at each user company and the ETSI standardisation process have been described showing special needs that must be fulfilled by a new solution. Such a solution in INTERVAL will be based on Formal Description Techniques (FDTs) that will be improved with timed extensions.

The second part of the report expresses the requirements for real-time development in relation with the FDTs retained in the project: SDL, MSC and TTCN. For every development stage: analysis, design, verification and validation, performance evaluation, testing, the needs are detailed and the lacks/limits of current notations are identified, leading to suggestions for extensions or improvements of the notations. Extensions defined in INTERVAL should give the user the possibility to realise a functional design that includes performance assessment based on the non-functional parameters specified in the model.

In addition, requirements have also been studied regarding the tools that support these notations and that will be prototyped/extended in the project. Regarding those tools requirements, it may be noticed that there is not currently one complete tool that cover all the functional and timing aspects addressed in INTERVAL, for the development of real-time systems. Current tools provide separately parts of the industrial needs. It is important that tools support standard formats for data and model exchange. On other respect, users may also want a toolset that is customisable in order to be used in other industrial domains which have different needs such as for automotive real-time software.

Finally a classification of the requirements is given that will serve as input to the next project phase: specification of timed extensions and definition of tool support. All the requirements regarding the SDL, MSC and TTCN notations as well as the tools have been reviewed in respect of their level: mandatory, desirable or optional, taking also into account the time scale and the resources of the project to be aware of what is attainable.

7. References

[SDL96] ITU-T: *Recommendation Z.100 – Specification and Description Language (SDL)*, ITU, Geneva, 1996

[SDL2000] ITU-T*: Recommendation Z.100 – Specification and Description Language (SDL)*, ITU, Geneva, November 1999

[MSC96] ITU-T: *Recommendation Z.120 – Message Sequence Chart (MSC)*, ITU, Geneva, September 1996

[MSC2000] ITU-T: *Recommendation Z.120 – MSC2000*, ITU, Geneva, November 1999

[TTCN-v2] ISO/IEC: *Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework – Part 3 (second edition): The Tree and Tabular Combined Notation. International Standard 9646-3*, ISO/IEC, 1997

[TTCN-v3] ETSI TC MTS: *Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation – TTCN-3 Core Language EN000063-1 (provisional)*, ETSI, Sophia Antipolis, 2000

[DHIP88] D.J. Hatley, I.A. Pirbhai: *Strategies for Real-Time System Specification*, Dorset House Publishing Co., New York, 1988.

[MTS00065] ETSI: *Methods for Testing and Specification (MTS); Methodological approach to the use of object-orientation within the standards making process; Initial Study*, DTR/MTS-00065 V1.0.9, Sophia Antipolis, 1999

[ETR298] ETSI: *Methods for Testing and Specification (MTS); Specification of protocols and services; Handbook for SDL, ASN.1 and MSC development*, ETR 298, Sophia Antipolis, 1996

[EG 202 103] ETSI: *Methods for Testing and Specification (MTS); Guide for the use of the second edition of TTCN*, EG 202 103 Sophia Antipolis, 1999

[INT8] INTERVAL Technical Report, I. Ober: *Requirements Specification Languages. A comparative Study of MSC, GOAL and TAU Observer Processes.* Reference IST/11557/WP5/07.00/VER/008, July 2000

[WG97] Th. Walter, J. Grabowski: *Real-time TTCN for testing real-time and multimedia systems.* In: Testing of Communicating Systems (Editors: M. Kim, S. Kang, K. Hong), Volume 10, Chapman & Hall, 1997. (available at http://www.itm.mu-luebeck.de/publications/IWTCS98RT/IWTCS98-RT-TTCN.ps.gz)

[HDGN00] D. Hogrefe, Z. R. Dai, J. Grabowski, H. Neukirchen*: Requirements on the time extensions of SDL and TTCN from the point of view of testing*, Technical Report to the INTERVAL mailing-list, June 18, 2000

[MTMC99] A. Mitschele-Thiel, B. Müller-Clostermann: *Performance Engineering of SDL/MSC Systems*, tutorial paper at 9th SDL-Forum, Montreal, 1999. (available at http://www.cs.uni-essen.de/Fachgebiete/SysMod/Papers/QSDL/TutSlidesForum99.pdf)

[GMB+00a] S. Graf, L. Mounier, M. Bozga, I. Ober, A. Kerbrat, D. Vincent: *Mini-Description of SDL & MSC Time Extensions*, presented at the INTERVAL meeting in Paris, May 02, 2000

[GMB+00b] S. Graf, L. Mounier, M. Bozga, I. Ober, A. Kerbrat, D. Vincent: *SDL for real time: What is missing?*, presented at SAM 2000 in Grenoble, June 26, 2000. (available at http://www.irisa.fr/manifestations/2000/sam2000/papers.html)

[DBL99] R. Dssouli, G.V. Bochmann, Y. Lahav (Eds.): *SDL'99 – Proceedings of the* 9th SDL-Forum, Elsevier, Montreal, 1999

[SBR97] I. Schieferdecker, B. Stepien, A. Rennoch: *PerfTTCN, a TTCN language extension for performance testing.* In: Testing of Communicating Systems (Editors: M. Kim, S. Kang, K. Hong), Volume 10, Chapman & Hall, 1997. (available at http://www.csi.uottawa.ca/~bernard/PerfTTCN.ps.gz)

[GPS96] P. Godefroid, D. Peled and M. Staskauskas: *Using Partial-Order Methods in the Formal Validation of Industrial Concurrent Programs*, IEEE Transactions on SE., Vol.22, No.7, pages 496-507, July 1996. Preliminary version in Proceedings of ACM SIGSOFT ISSTA'96 (Int. Symp. on Software Testing and Analysis), pages 261-269, San Diego, January 1996. (available at http://www.bell-labs.com/user/god/public_psfiles/ieee-tse96.ps)

[OBG99] ObjectGEODE 4.1 *Toolset Documentation*, VERILOG (now TELELOGIC) Products, 1999, http://www.csverilog.com

[IF99] M. Bozga, J.C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis : *IF – An intermediate representation for SDL and its applications*. In R. Dssouli, G.V. Bochmann, and Y. Lahav (Eds.), Proceedings of SDL Forum'99 (see [DBL99] above), Montreal, Canada, 1999

[ACD90] R. Alur, C. Courcoubetis and D. Dill, *Model checking for real-time systems*, 5th annual IEEE Symposium on Logic in Computer Science, Philadelphia, 1990

[BS97] S. Bornot and J. Sifakis: *Relating time progress and deadlines in hybrid systems*, International Workshop HART'97

[BST97] S. Bornot, J. Sifakis and S. Tripakis: *Modeling Urgency in Timed Systems*, International Symposium: Compositionality - The Significant Difference (Holstein, Germany), Vol. 1536 of LNCS, Springer, 1997

[SL95] S. Leue: Specifying Real-Time Requirements for SDL Specifications – A Temporal Logic Based Approach. In Proceedings of the Fifteenth International Symposium on Protocol Specification, Testing, and Verification PSTV'95, Chapmann & Hall, 1995 (http://swen.uwaterloo.ca/~sleue/publications.files/pstv95.ps.Z)

[MMG92] A. Morzenti, D. Mandrioli and C. Ghezzi: *A model parametric real-time logic*, in ACM transactions on Programming Languages and Systems, 14(4), 521-573, 1992

[SEG00] M. Schmitt, M. Ebner, J. Grabowski: *Test generation with AutoLink and TestComposer*, in Proceedings of SAM 2000 Workshop inGrenoble, June 26, 2000.