# Specifying, Validating, and Testing a Traffic Management System in the TRIO Environment

Angelo Gargantini^, Lilia Liberati^, Angelo Morzenti^, Cristiano Zacchetti+
^Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, ITALY;
+ ATM-Azienda Trasporti Municipale, Milano, ITALY.

**Abstract**

We report on an experience in applying a formal method to the specification and design of a system for monitoring and controlling surface vehicle traffic in a densely populated urban area. We illustrate the goals of the experience and describe the specification, validation, and verification activities. We also discuss the problems deriving from the particular, but, under several aspects, typical history of the application development, and from applying formal methods in an industrial setting. Finally, we assess the encouraging results obtained in the project.

**Key words and phrases**: Experiences with formal methods in project development, tool support for formal methods application, costs of formal specification and verification, control and automation systems, time- and safety-critical systems, temporal logic.

## 1 Introduction

Managing of large utility networks (railways, underground, highways, telecommunications, oil and gas transport, etc.), power production, transport and distribution, are today largely automated processes, and huge investments are made by several agencies and companies to procure and update the automation systems designed to support these processes. In many cases these systems are committed to external industries on the basis of requirement specifications. This fact justifies a widespread strategic interest for techniques and tools supporting requirement specification guaranteeing a clear separation from the subsequent design and implementation phases, and a high degree of maintainability of the specifications. Very often the developed systems have a high social impact, in that even a transitory malfunctioning may have important consequences in terms of economical damages, deterioration of the quality of social life, or even danger for human lives. Therefore specification techniques must support rigorous procedures for system validation and verification.

The obvious purpose of any specification is to express requirements and to serve as a reference for the successive phases of design, implementation, verification, and maintenance. Before these are undertaken, a very useful activity is often performed (especially when the specified systems are particularly complex or critical), namely the *validation* activity, which consists of establishing whether the actual requirements were indeed captured and correctly expresses by the specification. On the other hand, the *verification* activity is in charge of determining whether the constructed system correctly implements the requirements, as described (in a precise and possbly formal manner) by the specification.

Theoretical and applied research in computer science has been very active in the definition of formal methods providing a support to the specification, validation, verification, and maintenance of complex time and safety-critical computer-controlled systems [REX91]. However, up to the current state of the art there is no unified, consolidated, and widely accepted framework able to support all the development life-cycle, from specifying in an abstract, precise and unambiguous way the system requirements, to validating such requirements against user needs, to expressing design and implementation choices, to verifying the developed implementation (both by means of formal mathematical proofs and of empirical activities such as testing) against the stated requirements, to provide a support for design and system documentation and for system maintenance. Many formal methods, specification and design languages and methodologies exist, however, that provide an effective support for a subset of the above outlined activities.

In the past years the software engineering research group at Politecnico di Milano developed an environment for the specification, validation, and verification of time critical systems. This environment is based on TRIO, a linear time metric temporal logic, and includes a series of software tools providing (with various degrees of automation) a support to the crucial activities of the system development. TRIO is the result of a long term cooperation between academia and industry, therefore since the initial phases of its definition it has been applied to several case studies derived from industrial applications [CSt 90, CSt 92]; in recent years it was successfully employed in an ESPRIT ESSI[1] project concerning the specification and design of the control system of pondage power plants of ENEL, the Italian Energy Board [BC&95].

In the present paper, we report our recent experience in requirements formalization, design specification, system validation, and verification of an application for monitoring and controlling the surface vehicle traffic in a densely populated urban area. This activity was committed to the developers of the TRIO environment by Italtel, while the developed application will be installed in the metropolitan area of Milano and possibly, in the future, in urban areas having similar characteristics. The activity we report here constitutes the first "purely industrial" application of the TRIO based method, in that it was not motivated by any scientific goal (such as, e.g., obtaining new theoretical or methodological results or developing new support tools) but it simply aimed at obtaining technical, business, and organizational advantages by applying (under strict economical and temporal limitations) the TRIO language and environment to the development of the product, without relying on any support whatsoever from public or private research agencies.

The paper is structured as follows. Section 2 provides a brief overview of TRIO: to make the succeeding presentation reasonably self contained we illustrate the syntax of the language and the definition of the used derived operators. Section 3 illustrates the background, motivations and objectives of the activity describing in some detail the developed application, the history of the project and how the use of TRIO influenced the rather traditional development cycle of Italtel. Section 4 provides a report of the performed activity, including a few examples, simplified and abbreviated for reasons of brevity. In Section 5 we draw conclusions pointing out the most important technical results and the lessons learned.

[1] The ESSI program, supported by the European Union in the ESPRIT framework, is intended to favor the industrial use of novel technologies by partially covering the additional costs (in terms of education, training and experimenting) involved in their first applications.

## 2 A brief overview of TRIO

TRIO is a first order logical language, augmented with temporal operators that permit to assert properties at time instants different from the current one, which is left implicit in the formula. Unlike classical temporal logic, TRIO allows the specifier to express strict timing requirements by means of two basic operators—*Futr* and *Past*—which refer to time instants whose distance, in the future or in the past, is specified precisely and quantitatively. We now briefly sketch the syntax of TRIO and give an informal and intuitive account of its semantics; detailed and formal definitions can be found in [GMM90].

The alphabet of TRIO is composed of variable, function, and predicate names, plus the usual primitive propositional connectors '$\neg$' and '$\rightarrow$', the derived ones '$\wedge$', '$\vee$', '$\leftrightarrow$', ..., and the quantifiers '$\exists$' and '$\forall$', and plus temporal operator symbols *Futr* and *Past*. The language is typed, in that a domain of legal values is associated with each variable, a domain/range pair is associated with every function, and a domain is associated with every argument of every predicate. Among variable domains there is a distinguished one, called the *Temporal Domain*, which is numerical in nature: it can be the set of integer, rational, or real numbers.

Variables, functions, and predicates are divided into *time dependent* and *time independent* ones. This allows for representing change in time. Time dependent variables represent physical quantities or configurations that are subject to change in time, and time independent ones represent values unrelated with time. Time dependent functions and predicates denote relations, properties, or events that may or may not hold at a given time instant, while time independent functions and predicates represent facts and properties which can be assumed not to change with time.

TRIO formulas are constructed in the classical inductive way. A term is defined as a variable, or a function applied to a suitable number of terms of the correct type; an atomic formula is a predicate applied to terms of the proper type. Besides the usual propositional operators and the quantifiers, one may compose TRIO formulas by using primitive and derived temporal operators. There are two temporal operators, *Futr* and *Past*, which allow the specifier to refer, respectively, to events occurring in the future or in the past with respect to the current, implicit time instant. If *A* is a TRIO formula and *t* is a term of the temporal type, then Futr(A, t) and Past (A, t) are TRIO formulas too, that are satisfied at the current time if and only if property *A* holds at the instant which is *t* time units ahead (resp., behind) the current time. On the basis of the primitive temporal operators *Futr* and *Past*, numerous derived operators can be defined for formulas, including the following list.

| **Operator** | **Definition** | **Explanation** |
|---|---|---|
| AlwF(A) | $\forall t\ (t > 0 \rightarrow \text{Futr}(A, t))$ | A will always hold |
| AlwP(A) | $\forall t\ (t > 0 \rightarrow \text{Past}(A, t))$ | A has always held |
| Alw(A) | $\text{AlwP}(A) \wedge A \wedge \text{AlwF}(A)$ | A always holds |
| Som(A) | $\neg\ \text{Alw}\ (\neg\ A\ )$ | Sometimes A holds |
| Lasts(A, d) | $\forall d'(0<d'<d \rightarrow \text{Futr}(A, d'))$ | A will hold over a period of length d |
| Lasted(A, d) | $\forall d'(0<d'<d \rightarrow \text{Past}(A, d'))$ | A held over a period of length d in the past |
| Until($A_1$, $A_2$) | $\exists t\ (t>0 \wedge \text{Futr}(A_2, t) \wedge \text{Lasts}(A_1, t)\ )$ | $A_1$ will hold until $A_2$ starts to hold |
| Since ($A_1$, $A_2$) | $\exists t\ (t>0 \wedge \text{Past}(A_2, t) \wedge \text{Lasted}(A_1, t)\ )$ | $A_1$ held since $A_2$ became true |

## 2.1 Validation and verification in TRIO

Formal methods, being based on a solid mathematical foundation, have a clear and unambiguous semantics, so that the validation and verification activities can be effectively supported by (semi)automatic software tools that can greatly enhance the effectiveness and the practical impact of such activities. This is the case with TRIO, where an environment of tools for editing specification, validating them and verifying design and implementation has been developed in recent years at Politecnico di Milano.

Broadly speaking, the validation activity can take in TRIO the form of *history checking*, *history generation* (i.e., *simulation*), and *property proving*.

When performing history checking [F&M94] the designer invents (with the aid of a suitable tool) histories of the modeled system (i.e., sequences of events, system configurations, and values for the significant quantities that represent a hypothetical trace of a system execution) that in his/her view correspond to a possible behavior of the specified system where the requirements and properties are apparent. Such histories are then checked, i.e., compared for consistency with the specification, by considering each history as the frame of an interpretation structure for the TRIO formulas. The results of history checking are useful both to the final user, who verifies that his/her expectations on the system behavior are sensible, and to the specifier, who controls that his/his understanding of the requirements are correct and have been effectively formalized by means of the formal notation.

A more sophisticated method of validation consists of simulating the modeled system by generating (with the support of suitable specialized interpreters, see [MMM95]) histories of the specified systems under particular constraints that may represent an initial system configuration or particular combination of input events coming from the environment and are assumed to stress particular system functionalities that the designer wants to explicitly visualize.

In the TRIO framework, verification can take the two main forms of formal proofs and testing. Performing the formal proof of the correctness of an implementation w.r.t. its implementation requires that all relevant features of the hardware system software employed are themselves formalized, as outlined in [M&M94]. The derivation of such proofs can be made manually using the axiomatic system presented in [FMM94] or with the support of a theorem prover, such as PVS, where the

TRIO semantics and axiomatic system have been suitably encoded, as it was done in [Jef95].

Several years of experience of the authors in cooperation between academia and industry have however shown that formal proofs are not considered as a practically convenient technology, whereas various forms of testing are much more appreciated and widely employed. Formal specifications in TRIO can be used to support the generation of functional test cases, that include both data to be input as stimuli to the system to be tested, and expected results of the experiments, to be compared with the actual reactions of the tested system. Functional testing (often referred to in handbooks as *black box* testing) verifies the requirements of the system under test without any reference to the actual hardware/software structure of the implementation; it should be considered a complement, not an alternative, to structural, white box testing, which is based on the structural properties of the implementation (e.g., execution flow in software code). The activity of test case generation from TRIO specification is supported by a tool based on interpretation algorithms similar to those for system simulation and, moreover, support the annotation of test cases with information useful for the testing experiment [MMM95].

## 3 Background, motivations and objectives of the project

Italtel is an Italian manufacturing company, leader in the field of electronic devices and systems for telecommunication. In recent years it extended its activity to the areas of automation and control, including the production of hardware/software control and communication systems for the traffic regulation.

For organizational and economic reasons, in the present project most of the system development, starting from the specification requirement down to the coding of software, was committed to an external, relatively small, software house, called NUS; thus during all the project, the personnel of Italtel played mainly an organizational role: they established deadlines, coordinated the various phases of the project, organized meetings and information exchange among all the participants.

### 3.1 The developed system

The application we describe here is a centralized system for monitoring and controlling surface traffic in a high-traffic urban area, by means of a network of sensors and traffic signals. The final purpose of the system is to regulate the traffic in real-time depending on the current flow of vehicles and on the condition of the streets in the controlled area.

In the following we describe the system to be developed and the objectives of its regulating action.

The regulation goal is the minimization of the stop times of the vehicles at the traffic signals and the minimization of the number of such stops, for the main traffic flows in a urban area.

The urban area to be regulated is defined as a *basin*; the basin encloses several *crosses*, where a cross can contain several *groups*; a group is a part of the traffic signal, composed of the set of colors that are met by a traffic flow. As an example, a group can be the set red-yellow-green, another group can be a set of red-yellow-green right arrows. A cross can be composed of two groups in the typical situation where we have two intersecting traffic flows, or it can be composed of several groups.

The system architecture was designed around the four main kinds of performed operations, and therefore included the following four components.

- Traffic Monitoring Station (TMS), in charge of observing and estimating about the current vehicle flow in a given cross.
- Semaphore Regulator (SR), in charge of managing the green times of the traffic signals for a given cross.
- Basin Regulator (BR), which coordinates several SR's. The geographical extension of a basin depends on the traffic characteristics of the considered area and on the desired level of regulation.
- Control Center (CC): which coordinates several BR's and manages a system to provide the citizens with information on the current state of the traffic.

The system is centralized, but its lowest parts can operate also on a stand-alone mode, so that the optimization and regulation functions for a cross are carried out by the local SR. The possibility of a stand-alone functioning permits its applications in urban areas with a limited traffic flow; it reduces the information flow in the communication network, and it limits the damages in case of a crash in the CC.

The reported activity was centered on the TMS and the SR, so in the following we describe in more detail the structure and the functions of these two subsystems.

The TMS includes the following components:

- software procedures;
- several sensors installed on the street ground, to count the vehicles passing through the cross, for every traffic flow (the transmission latency between a sensor and the computer system is negligible);
- a hardware unit, shared with the SR, built over an Intel 80386 microprocessor.

The function of the TMS is the collection, analysis, and storage of the traffic data, based on inputs coming from sensors in several measure points. A measure point is composed of one or two sensors. The TMS can compute, on the basis of the inputs coming from the single measure points, the number of vehicles and the mean occupation time, while for the points provided with a double measure device it can compute a vehicles classification on the basis of the vehicle dimensions, the vehicle mean speeds, and the vehicles density.

The TMS can manage several measure points (up to a number of 64), by monitoring the state of the sensors, with a sampling time of 10 msec. As a result, the TMS gives to SR the following values:

- vehicle flow and mean speed, grouped by classes of vehicle dimensions;
- vehicles flow and mean speed, transformed into an equivalent conventional measure referred to a standard vehicle type;
- density of vehicles;
- saturation index of the street.

The TMS component, moreover, provides statistics and estimations of current and future traffic demand.

The SR includes the following components:

- software procedures;
- a hardware unit, shared with TMS;
- a configuration file, provided by an operator;
- a peripheral unit, composed of a keyboard and a display, to be used by an operator to define the SR configuration file or to regulate the traffic signals of the cross manually.

The configuration file determines the desired behavior of the traffic signals of the cross; this behavior depends on the geometric features of the cross and on the adopted traffic regulation policies. The most important information in the configuration file is the *traffic plan*. The traffic plan is the sequence of several *phases*, where a phase is characterized by the color of all the groups, and by a phase-time. Some phases can be extended, on the basis of calls coming from the way (pedestrians, public transport means, ambulances, ...); some phases can be executed only upon request.

The following list presents a sample of the functions of the SR.

*General functions.*

- Tests about the integrity of the resident information and the devices working.
- Execution of the traffic plan, with controls over the possible conflicts between traffic flows.
- Start-up procedures.
- Manual commands management.
- Data transferring to external devices.
- Degraded working, in emergency situations.

*Micro-regulation functions.*

- Execution of phases called by pedestrians.
- Execution/extension of phases called by vehicles.
- Execution of phases called by emergency means.

*Macro-regulation functions.*

- Execution of a sequence of traffic plans, timed by a daily plan.
- Generation of local traffic plans: SR tasks, using "historical" traffic data, can generate and execute traffic plans, to adapt dynamically its behavior to the state of the traffic flows.
- Generation of centralized traffic plans: SR tasks generate a traffic plan on the basis of "historical" data and of corrections imposed by the BR.

Finally, to guarantee the safety of the overall system, all the above described components contain procedures for checking data integrity and for maintaining some safety-related constraints (as a typical example two traffic signals on intersecting flows can never be simultaneously green).

## 3.2 Motivations and Objectives

The product developed in the described project is intended to be proposed to administrations of urban areas that intend to monitor and control the surface traffic. As it is assumed that local administrations would acquire such systems through a public bid where the competitors would be evaluated on the basis of cost-effectiveness of the proposed products, Italtel was at the same time very concerned about conceptual aspects such as logical correctness and safety, and about more commercial aspects such as its overall development and operation costs. As a consequence, our experience was by no means a *research* project, but an actual industrial development where the application of the formal method should provably lead to tangible advantages, both technical and economic.

During the development of the above described semaphore system, Italtel decided to acquire an independent assessment of the correctness and effectiveness of the product specified and designed by NUS. After some informal contacts and a study of the current state of the art in the field they were convinced of the usefulness of an approach based on formal methods, adopted TRIO as a suitable formalism, and asked the group of developers of the TRIO environment inside the Software Engineering group at Politecnico di Milano to organize the specification, validation, and verification activity. At the time when this activity started, the project had come to the following point: the TMS component had been specified and implemented (not yet tested), while the SR component had been specified and was under implementation.

Our job would be to formalize in TRIO the specification, analyze and validate it, and provide a plan for functional testing obtained from the TRIO specification as a set of test cases. The fact that this activity was not planned from the beginning of the development, and that it took place when the specification phase had already been completed and the coding was under way, was the main source of inconveniences in applying the TRIO-based method.

The work started with the study of the informal specification of TMS and SR [NUS95]. Not unexpectedly, the available documentation was incomplete and inconsistent. Many sections expressed design solutions rather than appropriately abstract requirements and constraints. It was a real challenge to avoid taking wrong steps under strict project deadlines and limited resources.

A further difficulty derived from the fact that the informal specification, although stated as "functional", was in fact written in a very operational style: for instance it contained several flow-charts with conditions and actions described in terms of software variables (or in a completely informal manner), and all temporal aspects were specified in terms of counters periodically updated by the software. Therefore, some of our work of formalizing the specification was in fact a "reverse engineering" operation, where we tried, to the extent compatible with the available resources, to extract an abstract version of the requirements from the accessible documentation. To some degree, we also adapted the TRIO formalism, which is descriptive in nature, to describe in a more operational fashion some operations and procedures that it would have been too long and costly to rephrase in a completely different manner.

As our work proceeded we met still another difficulty, deriving from the fact that the initial, informal specification had not been updated to keep it consistent with the modifications introduced during design and implementation.

## 4 Activity Report

Our activity was centered on the first two components of the system: the Traffic Monitoring System (TMS) and the Semaphore Regulator (SR). Our job was fundamentally to formalize the requirements and to plan the testing activity by generating functional test cases. These activities naturally led to a validation of the requirements: for instance, some test cases showed a few undesirable behaviors, that were however compatible with the specification.

A complete and detailed report of the activity and its results is clearly out of the scope of the present paper; in the following we just provide two illustrative examples of requirements formalization, regarding respectively the TMS and the SR, together with the results of requirements validation and a description of the generated cases.

### 4.1 TMS: requirements formalization

TMS analyzes the state (on or off) in time of two magnetic loops laying below the asphalt surface, in the same direction as the traffic flow. In what follows we will call "loop A" and "loop B" the first and the second loop in the direction of motion of the vehicles: the distance among the two loops is chosen in such a way that any vehicle traveling on the right direction will first enter loop A then enter loop B, then leave loop A and finally leave loop B. Also, the distance among the loops is such that any vehicle traveling at a reasonable speed will be detected when crossing one or both loops. Notice that for streets with many lanes there is a pair of loops for each lane. The system samples the state of the loops periodically (every 10 msec). Every 100 msec the collected data are analyzed and the results are memorized in a set of counters that indicate for how many ticks (each tick corresponds to a sampling time of 10 msec) a loop is on or off.

In this part of the specification we chose to maintain the counters, describing them by means of the following TRIO variables:

|  |  |
|---|---|
| TonA: | number of ticks the loop A was on |
| ToffA: | number of ticks the loop A was off |
| TonB: | number of ticks the loop B was on |
| ToffB: | number of ticks the loop B was off |

If a state has a duration less than a given value T_MIN, this state is interpreted as a transient interference and therefore ignored; in this case the system increases the value of the counter, as if state did not change. Therefore the system detects a vehicle entry on loop (or leaving) only after T_MIN ticks. When a transition (from *on* to *off* in case of vehicle entry or from *off* to *on* in the case of exit), is validated (which happens after the loop state is constant for T_MIN ticks), the value of the counters are memorized through suitable program variables. These variables are formalized in TRIO as follows.

TonAd: saved value of TonA

ToffAd: saved value of ToffA

TonBd: saved value of TonB

ToffBd: saved value of ToffB

Therefore, the program can compute for how many ticks a loop was free or occupied by a vehicle. Moreover the system monitors the time elapsed between the vehicle entry (or leaving) on the first loop and the entry (or leaving) of the same vehicle on the second loop, and therefore it can compute the velocity, acceleration and direction (on the right or worse direction) of the crossing car. The system also estimates the following quantities: length of the current vehicle, distance between a vehicle and the preceding one, number of vehicles (classified by length) up to present time.

The system also compares the states of the two loops, to find possible faults: for example if a loop is constantly *off* or *on* while the other one changes its state (this is called loop cross check).

## 4.2 TMS: requirements validation

During the formalization and test case generation activity, an error was found in the algorithm that updates counters upon a change in loop state and estimates vehicle direction. We detail here the part regarding estimation of vehicle direction.

If the vehicle is crossing the semaphore in the wrong direction, it first leaves the second loop B and then loop A as shown in Figure 1.

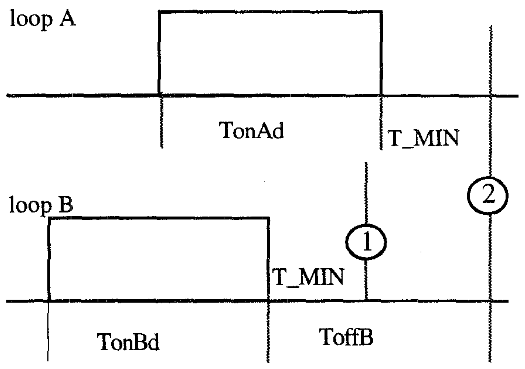


Figure 1. Sequence of loop states for a vehicle in wrong direction.

Let us consider how loop B detects vehicle direction. Loop B checks the direction in the instant marked 1 in Figure 1 (i.e., T_MIN ticks after the on/off transition). If the vehicle is crossing the loops in wrong direction, loop A still has not completed its detection. When loop A counts a vehicle (which takes place at time marked 2, T_MIN after the on/off transition for loop A), it saves TonA in TonAd and sets TonA to zero. The criterion for estimation of vehicle direction for loop B is expressed by the TRIO formula:

$$ToffB = T\_MIN \wedge TonA \neq 0 \wedge TonAd = 0 \rightarrow WrongDirectionB$$

Then loop A activates direction estimation at instant 2 of Figure 1. It determines that the vehicle is going in the wrong direction if it left loop B while it still was on loop A, i.e., if ToffB is more than T_MIN and less than

T_MIN+TonAd. In TRIO we express this condition as follows.

$$\text{ToffA} = \text{T\_MIN} \wedge \text{T\_MIN} < \text{ToffB} < \text{T\_MIN} + \text{TonAd} \rightarrow \text{WrongDirectionA}$$

Now the abstract system requirement is that a vehicle traveling in the wrong direction should be ignored, therefore its passage should have no consequence on the value of the counters. To ignore a vehicle passage, the system adds to the off counter (equal to T_MIN) the value of the *on* counter and the value of the *off* counter saved when this vehicle entered on the loop.

The system undesired behavior derived from the fact that if a vehicle crossed loop B in the wrong direction, this change in the values of the counters was done too early, i.e., at time instant 1. This change of values for the loop B at time 1 was expressed as follows.

$$\text{WrongDirectionB} \rightarrow \text{ToffB} = \text{T\_MIN} + \text{ToffBd} + \text{TonBd}$$

As a consequence the successive direction check on loop A (at time 2) would find the counter of loop B changed as if no vehicle crossed and could not recognize the vehicle in wrong direction. This error was corrected as follows: loop B counters are modified not at the time of direction check for loop B but at the time of direction check for loop A.

## 4.3 TMS: system verification

We outline here only the kinds of test cases we produced. Our guideline in the generation was the quality of the generated test cases, not their number: we tested every critical aspect of the system. The generated test cases included:

- vehicles at various speeds (lowest speed is in case of a queue) and accelerations;
- noise of the loop state, of *on-off-on* kind and *off-on-off* kind;
- loop failures such as loop in constant *on* or *off* state and loop in intermittent state (in such cases the cross check is positive and one of the loop is momentarily ignored);
- vehicles crossing in wrong direction;
- different traffic conditions: traffic queue, intense traffic, normal traffic and sparse traffic.

## 4.4 SR: requirements formalization

We specified the whole semaphore regulator, that in the informal specification documents and in the implementation was poorly modularized. So we tried to achieve the modularization of the system in the formal specification, grouping distinct concepts and processes. Modularity of the formalization facilitates reasoning on the requirements, which was essential for test case generation. The system was divided into four components.

**Diagnosis:** this monitors electric voltage and current on the lights. In case of unexpected values, the system starts yellow flashing. If the failure continues, all lights are turned off.

**Decision mode:** starting from keyboard requests and from central commands, this part decides how semaphore must work: automatic (SR alternates red lights and green lights in a full automatic way), all lights red (in the emergency or startup phase), manual (SR changes phases only upon command from the keyboard), coordinated mode (when single SR is synchronized with other SRs to obtain a green wave).

**Duration generation:** depending on current traffic density, SR changes green duration of some direction, to improve vehicle flow.

**Phases timing:** this part executes traffic plan, calling phases in the right sequence (considering phase requests and special phases for emergency and public vehicles) and giving to each phase the right duration. We report here on the formalization of phase temporization algorithm, when a phase is with call, extension and reservation.

The system executes a phase with call only if there was a vehicle detection by an magnetic loop or a button pressing by a pedestrian. A phase with extension has a variable duration, increased generally by a vehicle passage on a loop, whereby its duration can reach a maximum time. In case this maximum time is reached and the phase is with reservation, then it will be executed in the successive cycle.

The diagram in Figure 2 shows the working of a phase with call, extension and reservation as informally specified in the initial document: $t_0$ counts the ticks since the phase is active, and $t_e$ counts the extension time still to give. The temporal meaning of diagram is:

a phase ends if it is active since a time equal to the maximum one (PhaseTime) or if no extension request arrived for a time equal to ExtensionTime.

The task that updates counters runs every 100 msec, so there is an obvious correspondence between values of counters and elapsed time. However timing specifications expressed in terms of values of the counters would be rather cumbersome, intricate, and, most important, strongly dependent on implementation features. Therefore, we chose not to use counters to express timings but rather we used TRIO operators thus obtaining more readable and transparent specifications.

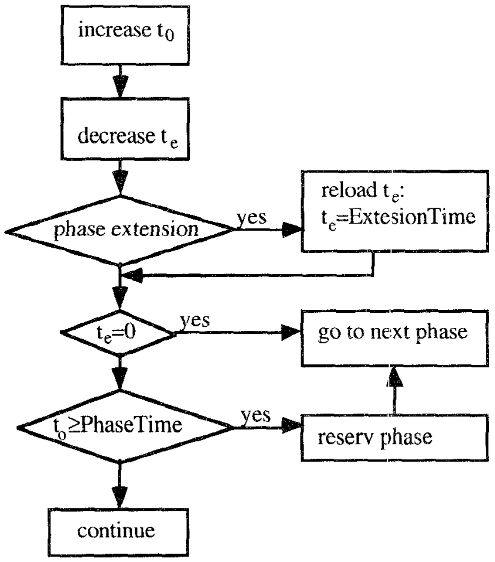


Figure 2. Flow-chart describing a phase with call, extension, and reservation.

We show here the meaning of some predicates used in the formalization:

*Phase(i)*: i-th phase of the traffic plan is now active. If a phase is with reservation, predicate *Reservable(i)* is true too.

*PhaseTime(x):* x is the maximum actual phase duration.

*ExtensionTime(x)*: x is the time to add to duration if there is an extension request.

Predicate *PhaseChange()* is true if actual phase ends, *Extend()* is true if an extension request arrives.

We can thus express the change of the i-th phase with the following TRIO axiom.

$$\left(\begin{array}{c} Phase(i)\wedge \\ Reservable(i)\wedge \\ PhaseTime(pt)\wedge \\ ExtensionTime(et)\wedge \\ Lasted(\neg Extend(), et)\wedge \\ Lasted(Phase(i), pt) \end{array}\right) \leftrightarrow PhaseChange()$$

*Reserved(i)* in our specification is true if the i-th phase is reserved. In TRIO reservation condition is:

$$\left(\begin{array}{c} PhaseChange()\wedge \\ Phase(i)\wedge \\ Reservable(i)\wedge \\ ExtensionTime(et)\wedge \\ Lasted(\neg Extend(), et) \end{array}\right) \rightarrow Until(Reserved(i), Phase(i))$$

### 4.5 SR: requirements validation

Test cases showed an incorrect behavior of SR only for phases with reservation. SR executed regularly a phase reserved, but light configuration was wrong.

### 4.6 SR: system verification

We produced many test cases for every functionality of SR. In every group of test cases we focused on a single specification component, and we examined a single critical aspect.

**Light diagnosis:** system working with very short transient power failures; cases with unexpected presence or lack of current and/or voltage on the lights; cases with red (or green or yellow) lamps erroneously enlightened; failures persistence and the consequent release of teleswitches.

**Mode decision:** automatic mode (local or centralized), manual, with every red lamp, lighting of the yellow lamps.

**Phase duration generation:** different traffic situations, different cross topologies and different traffic plans.

**Phase timing:** traffic plans composed of more phases with or without call, extension, reservation.

## 5 Conclusions: evaluation and assessment

In evaluating the results of the activity and assessing the usefulness of this industrial application of TRIO, we should keep in mind that the situation where we operated was far from ideal, both from the technical and the organizational viewpoint. First, the specification document we started with was mostly informal, without any modular structure, extremely detailed (in fact many system features had been over-specified by expressing them in terms of implementation-oriented solutions such as flowcharts, pseudo-algorithms. etc), while some features were described ambiguously or in several, mutually inconsistent ways. The available budget was extremely tight, both in terms of economic resources and available time (deadline of the project). The validation and test planning activity were performed *during* (or even after) the design and coding phases, starting from a specification document that had not been modified according to the changes in the system functions occurred during design. Because of all the above outlined limitations and inconveniences the activity constituted a real challenge, and we were forced to keep a very pragmatic attitude, aimed at solving the numerous problems encountered in the fastest and most economical way.

Despite these adversities, we can now claim that the activity was successful. Such a claim is mostly based on the fact that Italtel considers the obtained results very useful and intends to apply the same methods to the development of the next scheduled system component, i.e., the Basin Regulator. Also NUS expressed a very positive appreciation, and is interested to applying the validation and verification method on a routine basis in future development projects.

The validation activity allowed us to find several inaccuracies in the original specification document, while the testing experiments (which have not yet been terminated but have been completed for the vast majority) did not find significant errors in the software code. Adjusting the specification (and, correspondingly, the implementation) according to the errors and inaccuracies found during the validation phase, was a relatively expensive process; this is, in our view, the major negative consequence of having started the validation activity only when the design and implementation were in an advanced state or even completed. In fact this is just yet another demonstration of the usefulness of performing validation in parallel with specification, before any costly development takes place, as advocated by [Kem85]. On the other hand, it must be pointed out that identifying and correcting such errors would have been even more costly if the system was already in operation!

We also note that the discovered errors influenced only the accuracy of the developed system in measuring the traffic flow or its effectiveness in optimizing the vehicle circulation, not the safety requirements that are obviously essential for a semaphore system. This is because the safety-related checks were assigned (adopting a rather judicious and prudent approach) to separate modules that proved to be specified in a very simple and transparent way and correctly implemented.

The fact that the testing activity was, up to now, "unsuccessful" (i.e., it did not discover errors) was not surprising to us, for two main reasons. First, the personnel from NUS demonstrated during the whole activity a high technical competence for anything concerning the application domain and the programming activity, so we expected them to code the software with great attention and accuracy, and to test it thoroughly according to traditional structural testing methodology. Second, inaccuracies in the specification give rise to errors that can be considered as "common mode", in that they cannot be detected by a formal *verification* activity, which consists of comparing the implementation with the specification; such inaccuracies can be detected only through a *validation* activity, which allows the designer to compare the requirements, as (formally) stated in the specification, with the actual user needs. Finally, we notice that the activity of test plan production is useful even in the extreme case where the testing experiments does not reveal any error in software code because, as we point out in [MMM95], quite often test case generation allows the designer to simulate the system behavior, and therefore provides additional powerful validation means.

The tools for validating histories of the system and for generating test cases were used effectively: we produced quite a large set of test cases and analyzed several histories of the system for consistency with the specification. We do not conceal, however, that the use of the tools was at times both difficult and time consuming. In particular, as we pointed out in [F&M94, MMM95], the interpretation algorithms on which the tools are based, are at least exponential in the size of the formulas (i.e., in the number of nested connectives and quantifiers) so that, for large formulas, the performance of the tools can be unacceptable. In an ideal scenario (having sufficient time and resources) we would have structured the specification using the modular language TRIO$^+$ [M&S94], but in the present case we chose to rewrite some subformulas to improve efficiency in their interpretation. A typical applied transformation changed a TRIO formula of the kind

$$\forall x \forall y ... \forall z(A(x) \wedge B(y) \wedge ... \wedge C(z) \rightarrow D(x, y, ..., z))$$

(where each of the conjuncts in the premise of the implication holds for just one or a few values of the argument variable) into the following, equivalent formula

$$\forall x(A(x) \rightarrow \forall y(B(y) \rightarrow \ ... \rightarrow \forall z(C(z) \rightarrow D(x, y, ..., z))...))$$

that avoids the combinatorial explosion deriving from the nested quantifications.

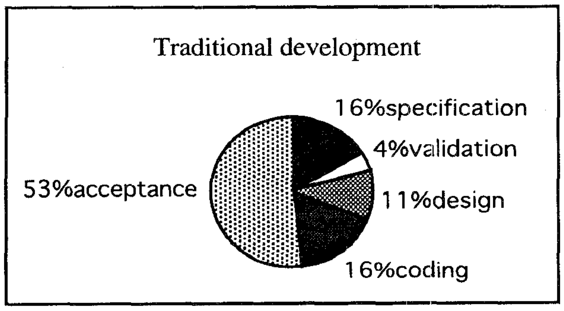
Another source of difficulties in the use of the tool for test case generation, derived from the fact that often the user must provide suggestions to guide the interpretation process, otherwise the tool would spend an undue amount of time in useless attempts to find models for subformulas. In some extreme cases the user found it more convenient to write manually test cases (which are rather similar to histories) and then simply check them with the history checker component of the tool. From a mathematical viewpoint the test cases generated in this manner were perfectly correct, but we must admit that this situation is rather far from the ideal procedure where the designer extracts test cases directly from the specification.

The above remarks lead us to conclude that test case generation must be performed by relatively highly skilled personnel, having a good knowledge both of the application domain and of the TRIO formal notation. System analysis and verification is still a labor-intensive activity that cannot be fully automated: it can only be supported with semiautomatic tools whose operation requires some ingenuity.

In the near future our efforts will be devoted to improving the effectiveness of the tools supporting the system simulation and test planning by

- removing the few remaining bugs;
- providing further functions;
- improving the user interface (including graphical features, better interaction with the file system, tools for managing first-order formulas);
- provide guidelines for a proficient use of the tool (suggestions regarding typical formula structure and described behavior).

From an organizational viewpoint, our principal remark is that, in any system development of a significant complexity, an increase in the effort spent in the initial phases of requirements specification and validation constitutes a profitable investment, since it reduces the overall development cost. Italtel staff recognizes that the activity centered on TRIO led to the discovery of errors or inaccuracies whose detection and correction would have been much more expensive if carried out during maintenance. In the ESSI project mentioned in the introduction [BC&96] the systematic application of the TRIO-based method led to a shift in resource allocation (with respect to more traditional development methods) from the design and acceptance phases to the requirements specification and validation phases, according to what shown in Figure 3, yielding a significant reduction of the overall costs, and improved project management and control.



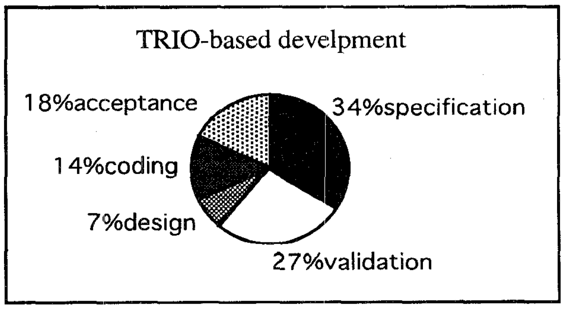


Figure 3. Shift in resource allocation from traditional to TRIO-based development.

Probably for organizational reasons (strict deadlines, resistance to profound change), Italtel is however not willing to modify its development cycle introducing a pervasive use of formal methods; it is simply willing, in the development of the Basin Regulator component, to perform requirements validation and test planning in parallel with design and coding. Hopefully, they will apply a new methodology and life-cycle to a brand new project in the future!

**Acknowledgments**

We would like to thank Dino Mandrioli and Fabio Schreiber for several illuminating discussions in the initial phases of the project.

## References

**[BC&95]** M.Basso, E.Ciapessoni, E.Crivelli, D.Mandrioli, A.Morzenti, E.Ratto, P.San Pietro, "Experimenting a Logic-Based Approach to the Specification and Design of the Control System of a Pondage Power Plant", ICSE-17 Workshop on Industrial Application of Formal Methods, Seattle, WA, April 1995.

**[BC&96]** M.Basso, E.Ciapessoni, E.Crivelli, D.Mandrioli, A.Morzenti, E.Ratto, P.San Pietro, "A logic-based approach to the specification and design of the control system of a pondage power plant", to appear in Colin Tully, editor, "Improving Software Practice" John Whiley Series on Software Based Systems, John Wiley & sons, New York, 1996.

**[CSt 90]** *Specification environments for real time systems based on a logic language,* Technical annex to research contract 27/90, December 1990, Case studies (in Italian) on a regulator in a pondage power plant and on high-voltage substations.

**[CSt 92]** *Specification environments for real time systems based on a logic language,* Technical annex to research contract 49/92, December 1992, Case studies (in Italian) on a programmable digital energy and power meters and on data collection and elaboration for dam security.

**[F&M94]** M.Felder, A.Morzenti, "Validating real-time systems by history-checking TRIO specifications", ACM TOSEM-Transactions On Software Engineering and Methodologies, vol.3, n.4, October 1994.

**[FMM94]** M.Felder, D.Mandrioli, A.Morzenti, "Proving properties of real-time systems through logical specifications and Petri net models", IEEE TSE-Transactions of Software Engineering, vol.20, no.2, Feb.1994, pp.127-141.

**[GMM90]** C.Ghezzi, D.Mandrioli, A.Morzenti, "TRIO, a logic language for executable specifications of real-time systems", The Journal of Systems and Software, Elsevier Science Publishing, vol.12, no.2, May 1990.

**[Jef95]** R.D.Jeffords, "An Approach to Encoding the TRIO Logic in PVS", Technical Report, Naval Research Laboratory, Wash.,D.C.,1995.

**[Kem85]** Kemmerer, R.A., "Testing Formal Specifications to Detect Design Errors", *IEEE Transactions on Software Engineering 11*, 1 (January 1985), 32-43.

**[M&M94]** L.Mezzalira. A.Morzenti. "Relating specified time tolerances to implementation performances", 6th IEEE Euromicro Workshop on real-time systems, Vaesteraas, Sweden, June 1994.

**[M&S94]** A. Morzenti, P. San Pietro, "Object-Oriented Logic Specifications of Time Critical Systems", ACM TOSEM - Transactions on Software Engineering and Methodologies, vol.3, n.1, January 1994, pp. 56-98.

**[MMM95]** D.Mandrioli, S.Morasca, A.Morzenti, "Generating Test Cases for Real-Time Systems from Logic Specifications", ACM TOCS-Transactions On Computer Systems, November 1995.

**[NUS95]** NUS (Sistemi per la Pianificazione Urbana e territoriale), "Detailed specification of a traffic monitor and a semaphore regulator", Project Documentation (in Italian), October 1995.

**[REX91]** Proceedings of the REX Workshop "Real-Time Theory in Practice", Plasmolen-Mook, The Nederlands, June 1991, "Lecture Notes on Computer Science", n.600, Berlin, 1991, Springer Verlag.