Verification of External Specifications of Reactive Systems*

P. Bellini, M. A. Bruno, P. Nesi

Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze

via S. Marta 3, 50139 Firenze, Italy, tel: +39-055-4796523, fax: +39-055-4796363

email: nesi@ingfi1.ing.unifi.it, *www*: http://www.dsi.unifi.it/~nesi

SMC-097-08-B843

Abstract

The External Specification is currently approached by specification languages for describing and analyzing system requirements. The External Specification can be defined during the early stages of the system development and can be very useful for: checking the class/system/subsystem requirements; checking the system composition; evaluating costs of reuse; defining validated reference requirements, histories, and traces for the final validation. This paper presents a collection of criteria in order to formally verify the External Specification of reactive systems/subsystems. The verification criteria are grounded on the TROL specification model for real-time systems. In TROL, the External Specification is expressed in terms of ports and clauses with temporal constraints. The goal of the verification criteria presented is to check the completeness and consistency of the External Specification with special attention to temporal constraints. These criteria can be applied to other real-time specification models and have been enforced in the TOOMS tool. A practical example illustrates the verification process that embodies these criteria.

Index term: Verification, Requirements Engineering, External Specification, Real-time Systems, Object-oriented.

1 Introduction

Techniques for specifying reactive as well as real-time systems are presently the focus of interest of many researchers [1]. Due to the power required for industrial machines, the application area of reactive systems

*This work was partially supported by the Italian Research Council, CNR (Consiglio Nazionale delle Ricerche)

becomes larger every day. In such systems, relevant failures are caused by violations of conditions specified by means of the so-called temporal constraints (i.e., deadline, timeouts, etc.). It has often been demonstrated that, even for large applications, the adoption of formal methods since the early phases of the development life-cycle reduces the failure probability of the final product – e.g., [2], [3], [4], [5], [6], [1]. Moreover, formal models are effective if they are supported by verification and validation techniques to ensure the correctness of the system specification.

For these reasons, several languages and tools for modeling the system under specification have been proposed starting from its requirements – early examples are [7], [8]. More recently, the Object-Oriented Paradigm (OOP) has also impacted on several formal specification languages, in that these have been extended in order to support system structuring (i.e., composition/decomposition) and reuse – e.g., [1], [9]. The adoption of OOP also provides an improved means with which to identify the External Interface of each system component. The specification of the External Interface is usually given in terms of the External Specification which consists of a set of statements describing the high-level behavior of the component under specification and the structure of its interface.

The formalisms describing the External Specification of reactive systems can be classified in I/O-based or method-based (i.e., service based) models. In *I/O-based models*, an object/component is seen as a block which communicates through a number of I/O ports (via message passing) with the surrounding environment (i.e., other objects) (e.g., OSDL, ObjectChart, TOOMS). In *method-based models*, objects are still seen as blocks and their communications with other objects take place via procedure calls (e.g., Z++, Object-Z, VDM++) [1]. A more general model describes I/O ports or methods as class (entity, component) services.

The External Specification is typically given by means of predicates or traces, and can be considered as an abstract description of system (components, class) specifying its behavior in the most important operating conditions. In fact, this typical behavior of the system is frequently described on the basis of its status condition; in many cases, it is assumed that the status is externally visible using certain type of monitoring panel. For example, the External Specification is given by using: *History* and *invariants* in Z++ [10], *History Invariants* in Object-Z (linear time predicates) [11], *Aux-reasoning part* in VDM++ [12], *clauses* in TOOMS/TROL [13], *predicates* in TRIO+ [14], *Message Sequence Charts* describing scenarios (i.e., possible traces) in OSDL [9], ObjectCharts [15], ROOMCharts [16], etc. Other event-based languages (e.g., [17]) and assertion-based languages (e.g., [18], [19]) could also be adapted by collecting rules according to the External and the Internal point of view of the specification. Some of the above methods include special constructs for specifying temporal constraints, whereas others include temporal logics (e.g., MTL [20], TILCO [21], [22], TLA [23], [24], RTTL [25], [26], LEPAForM [27], TRIO [14], and for a survey [28]).

On the basis of the External Specification the phases of verification and validation can be performed from the early stages of the specification. A specification model based on a formalization of the External Specification and endowed with mechanisms for its verification and validation can be very useful for: (i) checking the class/system/subsystem requirements, (ii) checking the system composition, (iii) evaluating costs of reuse, (iv) defining validated reference requirements, histories, and traces for the final validation, and (v) early detection of inconsistencies.

Please note that the two terms verification and validation do not always have the same meaning [6], [29]. For instance, in [6] the most frequently used definitions for *verification* have been reported, while the term validation is used to mean "final verification". For *verification* we intend the phase in which the completeness and consistency of the specification are checked. A specification is complete to the extent that its parts are present and fully developed [29]. A specification is consistent to the extent that its provisions do not conflict with each other or with the general objectives [29]. The verification of correctness and completeness is typically performed statically by controlling the syntax and semantics of the model without executing the specification. For *validation* we intend the phase in which more general properties (that the system under specification must provide according to requirements) are demonstrated to be already enforced in the specification. The system validation consists in controlling the conditions of liveness (i.e., absence of deadlock), safety, and the meeting of timing constraints (e.g., deadline, timeout, etc.), etc., in the presence of critical conditions. It is usually performed statically in descriptive approaches (i.e., by proving properties) and dynamically in operational approaches (i.e., by simulation). Some of these aspects can also be detected in the phase of verification if a suitable formal model is used.

In mainly operational models and tools, the verification phase is frequently reduced to the syntax checking without controlling the completeness and the consistency of the behavioral parts of the specification. System behavior is frequently controlled only during the validation phase in which model-checking, and/or property proof techniques are adopted. During model-checking only a part of system behavior is validated against specific traces which have been provided by the user or, in some cases, automatically generated from the early phases of system specification. A different approach consists in the adoption of a theorem prover in order to prove properties against the specification (in this case the specification model must be supported by a specific theory). A middle approach may consist in the adoption of a symbolic model-checking which allows for validation of a larger part of system behavior with an effort comparable to that of performing model-checking. The above mechanisms must be obviously be supported by the semantics of the specification model adopted – e.g., first order logic, higher order logic, CSP, CCS, Petri Net, State Machines, set theory, etc.

Most of the previously mentioned techniques adopt the External Specification as a model for validating the specification by means of model-checking or by using other techniques. In those cases, it is assumed that the External Specification is correct and consistent, yet this is obviously a genarally false statement. For this reason, a mechanism for verifying completeness and consistency of the External Specification is needed. If the External Specification is complete and consistent the validity of its adoption for the general system validation has a greater significance. This is also true for the External Specification of each system component and subcomponent, thus, it is also useful for system composition/decomposition.

Obviously, the definition of verification and validation must be adapted when used to define criteria for verifying External Specifications. The verification mechanism must also guarantee that the histories constituting the External Specification are consistent and that the External Specification itself is complete. On the other hand, the External Specification is a non-exhaustive description of the component even if it is complete, but this can be very useful to better define the boundaries of the internal, detailed specification [30]. Please note that the use of verified and validated External Specifications is the first step towards building the Internal Specification. The External Specification and the Internal Specification must be consistent, coherent, etc. Therefore, the External Specification and its verification and validation techniques can be profitably integrated with techniques for system composition/decomposition – e.g., [31], [32], [33], as well as with object-oriented models and reuse-based techniques such as in our TOOMS/TROL approach [34], [13], [5].

In this paper, the authors present the criteria for the verification of the External Specification of TROL components. TROL (Tempo Reale Object-oriented Language) is an object-oriented language for the specification of real-time systems [13], [34], [35]. It adopts a dual model presenting both descriptive and operational aspects providing support for describing the system behavior, its functionality and structural aspects. TROL is supported by the CASE tool TOOMS which consists of a set of visual editors, a report generator, a database for collecting and recovering specifications for reuse, a compiler, an analyzer to perform the verification of completeness and consistency, a validator, and a simulator (that can simulate the system behavior by using both clauses and state machines) [34], [36], [5]. At each specification level, TOOMS can help the user to verify consistency, thus allowing the incremental specification and the execution of partially specified systems (i.e., prototyping) [34], [5]. The verification criteria have been set up to (i) detect inconsistency and incompleteness in the External Specification, (ii) create the bases for composition/decomposition process of development, (iii) formalize the external behavior of the component/system for its further adoption in order to validate the Internal Specification and/or implementation.

Each TROL entity presents an External Specification based on clauses and temporal constraints. The External Specification can be regarded as a description in terms of traces or first order predicates (such as in Object-Z or in OSDL), since clauses may be subsequently activated to form traces or histories. For these reasons, the criteria reported in this paper can be profitably used even on other specification models.

The paper is organized as follows. In Section 2, the TROL model is briefly reported in mathematical notation. In Section 3, a selection of criteria to verify completeness and consistency of External Specifications is formally presented with some small examples. The criteria proposed are illustrated in Section 4 through an example based on the specification of a cellular phone. Finally, conclusions are drawn in Section 5.

2 Overview of TROL Formal Model

In TROL, the system under specification is hierarchically decomposed into objects and sub-objects. In this paper, only the problems related to the verification of their External Specifications are addressed. The following mathematical model is used as a basic notation to formalize the criteria for the verification of External Specifications. The criteria reported in this paper have been selected from those which are currently enforced in the TOOMS CASE tool [5], [13]. These do not claim to represent an exhaustive collection of criteria for the detection of all problems that can be present in the External Specifications. We started to work on the criteria after the analysis of the problems encountered during the final validation of specification. We noticed that most of them could be avoided by using consistent and complete specifications. On these bases, criteria for early detection of problems of concistency and completeness were defined. Please note that these problems can arise for entire systems but also for the validation of each component, sub-component. The criteria have been elaborated upon by following the concepts of completeness and consistency with the aim of solving the problems detected during the application of non fully verified external specifications of several approaches – [1].

In TROL, the descriptive aspects of the language are used to help developers generate complete and congruent specifications, verifying all system aspects by means of clauses and reasoning on timing constraints. The descriptive aspects are mainly contained in the External Specification.

A TROL specification, Θ, consists of a set of objects, Ω, where each object is instantiated from its class according to the object-oriented paradigm [13]. A TROL **object** ω is a pair, $\omega = \langle N_o, b \rangle$ where N_o is the object name and b is the class from which object ω is instantiated. A **class** b is a 3-tuple:

$$b := \langle N_c, ES, IS \rangle,$$

where:

- N_c is the class name – i.e., class identifier;
- ES is the External Specification of the class;
- IS is the Internal Specification of the class.

2.1 Formalizing the External Specification

The External Specification is defined as:

$$ES := \langle I, O, \Psi \rangle,$$

where:

- $I := \{i_1, i_2, \ldots, i_{n_i}\}$ is a set of provided services (i.e., input ports). A provided service can be *normal* or *buffered*;
- $O := \{o_1, o_2, \ldots, o_{n_o}\}$ is a set of required services (i.e., output ports). A required service can be *normal* or *available*;
- $\Psi := \{\psi_1, \psi_2, \ldots, \psi_{n_\psi}\}$ is a set of clauses. These are used for specifying the behavior of the class in terms of input/output relationships.

A provided service – i.e., an input port $i \in I$ – is modeled as a 5-tuple:

$$i := \langle N_i, D, M, R_m, R_M \rangle,$$

where:

- N_i is the input name;

- $D := real \mid integer \mid signal \mid enum \mid bool \mid typedef \mid multitype \ldots$ is the input type. $typedef$ are used to define records, *multitype* to define variables/connections which can store/pass data belonging to different types, *enum* are enumerated collections, *signal* are token-messages containing no data (very useful for synchronization);

- M :=*normal* | *buffered* is the port modality;

- $R_m \in \mathcal{R}$ is the minimum distance between two consecutive inputs, in time units;

- $R_M \in \mathcal{R}$ is the maximum distance between two consecutive inputs, in time units.

Therefore, R_m and R_M are time bounds on the occurrence of input messages. Communications among objects are supported through *message passing*; messages are considered as tokens irrespective of their content – i.e., whether they contain data and/or commands (i.e., controls) [13]. The basic communication model is synchronous on 1-way unidirectional channels with a blocking sender and a blocking receiver, as in the model proposed by Shaw in [37]. Different communication mechanisms among ports are possible depending on their combination. These can be defined as *normal* (i.e., synchronous), *buffered* and *available* (i.e., shared variable) [13]. Only inputs can be buffered, while only outputs can be available. Normal ports correspond to the default case, thus producing a CSP-like communication model. A required service (an output port) $o \in O$, is structurally modeled as the input port schema with the following differences: R_m and R_M are time bounds on the production of output messages, and $o.M = normal \mid available$.

Dependencies among services/ports are defined by means of clauses with temporal constraints [13]. A TROL clause, ψ, is constrained to be formally structured as:

$$(FC \wedge OC) \rightarrow CC - - [T_m, T_M];$$

It specifies the production of an output due to the arrival of input messages, in particular: if the Action Condition ($AC = FC \wedge OC$) (Firing Condition and Object Condition) is true at time t, then the Consequent Condition CC will be true in an instant of the time interval $[(t + T_m), (t + T_M)]$. $T_m \in \mathcal{R}$ and $T_M \in \mathcal{R}$ are the minimum and maximum bounds on reaction time associated with the clause, respectively. The clause notation adopts an implicit model of time; time is considered linear and continuous.

FC is a condition describing an event due to input ports, while OC is a positive predicate consisting of a set of conditions on the state of output ports having modality *available* (thus reporting the object status in brief). Therefore, OC may be considered as the precondition for FC. In the cases in which FC and OC are both empty, the clause states that CC is always activated. CC is a predicate describing the message on the output port. CC is only specified as a conjunction of conditions ($CC = \bigwedge p$) which ensures the determinism on the object behavior, since the adoption of disjunctions of CCs can lead to non-deterministic specifications.

A clause $\psi \in \Psi$ can be regarded as a set of *Reduced-form Clauses* on the basis of the following mapping Φ:

$$\{\varphi_1, \varphi_2, \ldots, \varphi_{n_\varphi}\} := \Phi(\psi),$$

where each reduced-form clause φ_j is derived from the structure of clause ψ according to the following constraint:

$$(\psi.FC = \bigvee \varphi_j.FC) \wedge (\psi.OC = \bigvee \varphi_j.OC) \wedge (\psi.CC = \bigwedge \varphi_j.CC) \wedge (\psi.T_m = \varphi_j.T_m) \wedge (\psi.T_M = \varphi_j.T_M). \quad (1)$$

The FC of a reduced-form clause depends only on a single input port. The specific port can be recovered by using the polymorphic function $port()$, $i_{FC} = port(FC)$ where $i_{FC} \in I$, thus function $port() : FC \rightarrow input$. The OC and CC of a reduced-form clause can depend on one or more output ports. The set of these output ports is obtained by using $O_{OC} = port(OC)$ or $O_{CC} = port(CC)$, respectively, where $O_{OC} \subset O$ and $O_{CC} \subset O$, thus function $port() : CC \rightarrow \{output\}$ or $port() : OC \rightarrow \{output\}$.

In a reduced-form clause the AC and the OC can be only defined as the conjunction of simple conditions on a port value, while the CC can be only a simple condition on one output port; thus the AC, OC and CC can be represented as a set of simple conditions. For example, the following clause:

$$\psi: \quad \mathtt{A} > 5 \wedge \mathtt{B} = 3 \wedge \mathtt{C} > 1 \rightarrow \mathtt{C} = 0$$

is represented as:

$$\psi.FC = \{\mathtt{A} > 5\}$$
$$\psi.OC = \{\mathtt{B} = 3, \mathtt{C} > 1\}$$
$$\psi.AC = \psi.FC \cup \psi.OC = \{\mathtt{A} > 5, \mathtt{B} = 3, \mathtt{C} > 1\}$$
$$\psi.CC = \{\mathtt{C} = 0\}$$

TROL clauses may be *direct* (as described above) or *reverse* and they are modeled with a tuple:

$$\psi := \langle N_\psi, W, FC, OC, CC, T_m, T_M \rangle,$$

where N_ψ is the clause name, and $W = direct \mid reverse$. Please note that reverse clauses, $\psi.W = reverse$, are structured as

$$(CC \wedge OC) \rightarrow FC \quad -- \; [T_m, T_M];$$

This means that a reverse clause specifies the behavior of the external environment of the class. This is performed by describing how the occurrence of an output message can lead to receiving an input message on the object itself – i.e., how the external environment needs an output for producing an input on the entity. Reverse clauses are typically included in the External Specification. They can be easily identified in the specification since they are differently labeled.

2.2 Formalizing the Internal Specification

TROL classes are defined as comprised of a collection of communicating objects (which in turn belong to their classes). Objects which cannot be further decomposed are defined as extended state machines [13]. Please note that, in this paper, we address only the verification criteria related to the External Specification; the mathematical notation supporting composition/decomposition mechanism in the Internal Specification is reported since this is useful to better understand the criteria defined in the rest of this paper. In fact, the External Specification is comprised of input and output ports, which in turn are involved in the composition/decomposition mechanisms establishing communications.

The Internal Specification is defined as:

$$IS := \langle A, C \rangle,$$

where:

- $A := \{\omega_1, \omega_2, \ldots, \omega_{n_\omega}\}$ is a set of sub-objects. Each class is formally defined as a set of communicating sub-objects A, which in turn are instances of other classes;
- $C := \{c_1, c_2, \ldots, c_{n_c}\}$ is a set of connections. Internal class connections are established by means of *links* or communication *channels*. *Links* are defined as simple connections between inputs and outputs of the composite object with the corresponding inputs and outputs of its sub-objects – e.g.,

$\langle i, k \rangle$ is a link where $i \in \omega.I$, $k \in \omega_1.I$, and $\omega_1 \in \omega.A$. Communication *channels* are modeled as a tuple $\langle o, I \rangle$, which represents a connection from an output port o to a set of receiving input ports $I = \{i_1, i_2, \ldots, i_{N_i}\}$ (i.e., $1 : N$ communications).

3 Verification of External Specification

The verification process for the External Specification is comprised of two steps as summarized in Table 1. The former step consists in the verification of completeness and consistency of clauses and input/output ports. The latter step corresponds to the verification of the External Specification in modeling the component behavior, and is performed by considering the sequences of clauses. These steps are typically followed by a third step in which the validation of the External Specification is performed with respect to other properties provided by the analyst to verify whether the specification includes those aspects [13]. This phase is substantially different from the classical validation where the internal implementation is validated against traces and histories, which in turn can be considered External Specifications.

The following criteria are a selection of the general conditions that should be satisfied by External Specifications. These may be ported from a TROL model to other models and languages.

External Specification	**Step**
Verification of Completeness and Consistency	Completeness of External Specification Elements
	Completeness among Class Clauses
	Consistency of each External Specification Element
	Consistency among Class Clauses
	Completeness of Temporal Constraints of the Class
	Consistency among Temporal Constraints of the Class
Verification of High Level Behavior	Cyclic Sequences of Class Clauses
	Consistency among Sequences of Clauses

Table 1. Steps of Verification of the External Specification.

3.1 Verification of Completeness and Consistency

This section describes how the analysis of the External Specification is performed in order to verify its completeness and consistency, by also considering temporal constraints. As a first step, ports as well as clauses have to pass the syntax verification (including type checking) according to the TROL model and syntax [13]. After the syntax verification, class clauses are processed for deriving reduced-form clauses in order to simplify the following steps of verification. For example, by considering clause:

$$\texttt{nameclause} : (((\texttt{A} > \texttt{0} \wedge \texttt{A} < \texttt{9}) \vee (\texttt{B} = \texttt{5}) \vee (\texttt{C} = \texttt{4})) \wedge (\texttt{Z} = \texttt{33})) \rightarrow (\texttt{X} = \texttt{5} \wedge \texttt{Y} = \texttt{6});$$

the following reduced-form clauses have been obtained by considering the disjunctions in the FC and that OC must be always verified according to the model defined in equation (1):

φ_1 : $\texttt{(A>0} \wedge \texttt{A<9} \wedge \texttt{Z=33)} \rightarrow \texttt{X=5;}$

φ_2 : $\texttt{(A>0} \wedge \texttt{A<9} \wedge \texttt{Z=33)} \rightarrow \texttt{Y=6;}$

φ_3 : $\texttt{(B=5} \wedge \texttt{Z=33)} \rightarrow \texttt{X=5;}$

φ_4 : $\texttt{(B=5} \wedge \texttt{Z=33)} \rightarrow \texttt{Y=6;}$

φ_5 : $\texttt{(C=4} \wedge \texttt{Z=33)} \rightarrow \texttt{X=5;}$

φ_6 : $\texttt{(C=4} \wedge \texttt{Z=33)} \rightarrow \texttt{Y=6;}$

Please note that reduced-form clauses contain conditions which are only a conjunction ($\bigwedge p$) according to equation (1).

3.1.1 Verification of Completeness

The verification of completeness must be performed on (i) each class element and (ii) by considering class elements on the whole (i.e., ports and clauses). To this end, the most important criteria determined have been reported, which state the needs of completeness when the relationships among inputs and outputs ports of a class/subsystem are defined.

Criterion 1 – *Completeness of the External Specification elements – The External Specification must be complete in the sense that each input and output port and clause must be fully specified in its parts. In the case of missing parts, it is necessary to assume default values in order to specify them.*

This general criterion is translated into a set of more specific criteria. For example: (i) each input port must at least be used in the FC of a class clause; (ii) each output port must at least be used in the CC of a clause. Moreover, for ports typed as enumerate collections (i.e., *enum*), the External Specification should provide a clause for each possible value of the enumeration. In this way, it is ensured that each message which may arrive to an input port has a counterpart in the specification to describe the corresponding effects. □

As previously stated, according to the TROL model, outputs with modality *available* report the class status (frequently defined as enumerated collections). In the External Specification the class behavior should provide enough details to report the evolution of each possible status. In terms of TROL model, CC of a class clause may specify when an object reaches an internal status (i.e., when CC is defined in terms of an *available* output which presents the internal status or its subset). For this reason, in order to verify the compatibility between CCs and OCs (which represent the change of status and the precondition as a function of class status, respectively) the following criterion must be satisfied.

Criterion 2 – *Completeness among Class Clauses – For each class clause, φ_i, the External Specification of the class has to include at least another clause, φ_j, having its $\varphi_j.CC$ compatible with $\varphi_i.OC$. This can be formalized by:*

$$\forall \varphi_i \in \Psi \quad \forall P \in \varphi_i.OC \quad \exists \varphi_j \in \Psi \quad \exists Q \in \varphi_j.CC \ : compatible(P, Q);$$

where function "compatible()", Bool $\times$ Bool $\rightarrow$ Bool, is defined as:

$$compatible(P,Q): \quad (port(P) = \emptyset) \vee (port(Q) = \emptyset) \vee (port(P) = port(Q) \wedge \ \exists \bar{x} \in port(P).D: \ P(\bar{x}) \wedge Q(\bar{x})).$$

where: P and Q insist on the same port set; $port(P).D$ is the Cartesian product of the domains of ports of P (Q); $\bar{x}$ represents a tuple of values for ports in P (Q); and, $P(\bar{x})$ represents formula P with the specified substitution of values for the ports.

Please note that $compatible(P,Q)$ is true if the two formulae using the same ports can be satisfied with the same values for the ports, but is also true if one of two formulae is empty (not referring to any port). For example, $compatible(A > 3, X = 3)$ is false since the two formulae use different ports, while $compatible(A > 3, A > 5 \wedge A < 9)$ is true because: there exists a value for port A that satisfies both formulae (for example A equal to 6). When a port is typed as real or integer, in the worst case one or more boolean terms can be present in P and Q for the same port. The number of these terms for each port is supposed to be limited to H, thus, the number of domain segments is limited to $2H + 1$. The predicate compatibility is verified for each possible segment. For example if $\mathtt{A}$ is a port with real values, P is $(\mathtt{A} > \mathtt{3} \wedge \mathtt{A} < \mathtt{7})$ and Q is $(\mathtt{A} > \mathtt{5} \wedge \mathtt{A} < \mathtt{9})$ then the domain of $\mathtt{A}$ is subdivided in five regions: $(-\infty, 3]$, $(3, 5]$, $(5, 7)$, $[7, 9)$, $[9, +\infty)$. In each region the conditions P and Q can be evaluated to find if there exists a region where are both true.

The verification of criterion 2 ensures that the evolution of the state has been specified, at least for what concerns the class status. This is visible by means of the available outputs which are used in the OCs. This criterion may detect the presence (i) of unreachable states, and (ii) of states from which the system cannot evolve further. □

3.1.2 Verification of Consistency

The verification of consistency must be performed (i) on single elements and (ii) considering the relationships among all class elements (i.e., ports and clauses). In general, input and output ports do not present any issue to be verified, while each clause must be internally consistent and with respect to all the other class clauses. These concepts are formalized with the next criteria.

Criterion 3 – *Clause Internal Consistency – Each class clause must be activated in at least one case by considering the domain of clause variables. This can be checked by verifying the following condition:*

$$\forall \varphi \in \Psi: \; \exists \bar{x} \in port(FC).D \; \exists \bar{y} \in port(OC).D: \; FC(\bar{x}) \wedge OC(\bar{y}).$$

Criterion 3 verifies if there exists at least a set of values for inputs and outputs for which clause φ is activated by considering the activation of both FC and OC. This criterion must be verified by each clause of the class, otherwise that clause is meaningless in the class context. □

At class level, each clause must be proved to be decoupled with the other clauses of the class (all in the reduced-form), for guaranteeing the absence of multiple activations (i.e., to avoid that the same input produces different values on the same output port according to different clauses). To this end, in the literature, a set of criteria has been defined for identifying conflicts between clauses of activation and/or on message production. For example by transforming the specification in a Tableau and from this to identify nodes which are in conflict [38], [39].

Criterion 4 – *Class Clause Consistency – Each couple of class reduced-form clauses, φ_i and φ_j, must be decoupled (not in conflict) with each other:*

$$\forall \varphi_i, \varphi_j \in \Psi: \quad \varphi_i \neq \varphi_j \wedge clauses\_decoupled(\varphi_i, \varphi_j),$$

where:

$clauses\_decoupled(\varphi_i, \varphi_j):$

$$\neg\, compatibleAC(\varphi_i, \varphi_j) \,\vee\, compatible(\varphi_i.CC, \varphi_j.CC) \vee ((port(\varphi_i.CC) \cap port(\varphi_j.CC)) = \emptyset);$$

and:

$$compatibleAC(\varphi_i, \varphi_j) : \quad compatible(\varphi_i.FC, \varphi_j.FC) \,\wedge\, compatible(\varphi_i.OC, \varphi_j.OC).$$

This means that, in order to avoid conflicts for a simultaneous firing of different clauses producing different values on the same output port one of the following three cases must be satisfied: (i) ACs of different clauses are incompatible, or (ii) compatible ACs of different clauses must have compatible CCs (i.e., redundant clauses), or (iii) compatible ACs of different clauses have CCs operating on different output ports:

$$\begin{aligned}&\neg compatibleAC(\varphi_i, \varphi_j) \vee \\ &(compatibleAC(\varphi_i, \varphi_j) \wedge compatible(\varphi_i.CC, \varphi_j.CC)) \vee \\ &(compatibleAC(\varphi_i, \varphi_j) \wedge ((port(\varphi_i.CC) \cap port(\varphi_j.CC)) = \emptyset)).\end{aligned}$$

This was transformed in the above definition of *clauses_decoupled*() by using the theorem of absorption. The function *compatibleAC* is used to check if the ACs are compatible, have to be noted that if the FC or the OC is missing for a certain clause, the AC can be compatible, in this way this criterion also considers the conflicts that may happen if clauses are always activated (e.g., $\rightarrow CC$). □

For example, $clauses\_decoupled(\mathtt{A} > 3 \wedge \mathtt{B} = 3 \rightarrow \mathtt{C} = 5, \mathtt{A} > 5 \wedge \mathtt{B} = 5 \rightarrow \mathtt{C} = 3)$ is true because the ACs of the clauses are incompatible while $clauses\_decoupled(\mathtt{A} > 3 \rightarrow \mathtt{C} = 5, \mathtt{A} > 5 \rightarrow \mathtt{C} = 3)$ is false, because the two ACs are compatible while the CCs are incompatible (e.g., if `A=7` then `C` can be 3 or 5). Another case can be explained with $clauses\_decoupled(\mathtt{A} > 3 \rightarrow \mathtt{C} = 5, \mathtt{A} > 5 \rightarrow \mathtt{D} = 3)$ which is true because the ACs are compatible but the CCs operate on different ports. Finally consider the case in which A is an input and B is an available output port (is a state variable) then $clauses\_decoupled(\mathtt{A} > 3 \rightarrow \mathtt{C} = 5, \mathtt{B} = 5 \rightarrow \mathtt{C} = 3)$ is false since the the two clauses have compatible ACs and incompatible CCs. The ACs are compatible because the FC of the second clause and the OC of the first clause are empty and so the two FCs and the two OCs are compatible.

3.1.3 Verification of Temporal Constraints of the Class

The verification of temporal constraints of a class must be performed by controlling their completeness and consistency, considering input and output time bounds and temporal constraints of clauses.

To achieve a complete External Specification, the time bounds for inputs and outputs and bounds on reaction time of clauses must be defined. The detection of missing temporal constraints is trivial. On the other hand, the time bounds of an output depend on the time bounds on the corresponding input according to the temporal constraints of the clause which relate them. The relationships among temporal constraints are a consistency problem, which is discussed in Criterion 6.

Criterion 5 – *Completeness of class Temporal Constraints – Each input, output and clause must provide a temporal constraint, otherwise suitable default values must be assumed.*

A time bound of $[0, \infty]$ is assigned as default (thus they are considered fully aperiodic signals) to input and output ports without time bounds.

In Fig.1, the specification of temporal constraints is depicted through a graphical representation, in which the temporal domains of input and output ports are reported on x and y axes, respectively. More specifically, in Fig.1, two particular cases are reported. On the left, the case in which both input, i, and output, o, have a temporal constraint equal to $[0, \infty]$, while a constraint equal to $[0, \infty]$ is also set for the clause describing their relationships. In this figure, the gray area represents the space of possible combinations of input, output, and reaction occurrences. Please note that the case in which an instantaneous reaction time is set for the clause corresponds to the diagonal line. The graph on the right presents the case in which an input has a time bound of $[0, \infty]$, while its corresponding output has $[\varphi.T_m, \infty]$ and the related clause has $[\varphi.T_m, \varphi.T_M]$. □

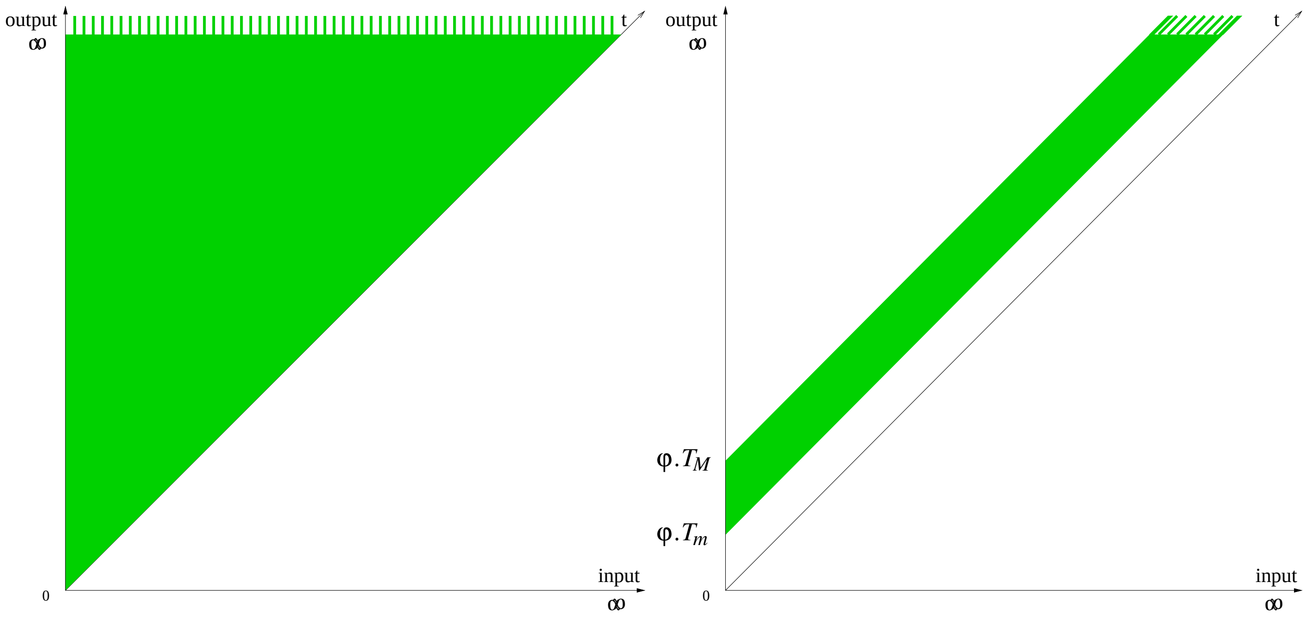


Figure 1. Notation to represent relationships among temporal constraints.

If the temporal constraint of a clause or that of a port is missing, then bounds on its corresponding values can be evaluated on the basis of the time bounds of related input and output ports and clauses. This is due to the fact that relationships of consistency can be defined among these temporal constraints.

Criterion 6 – *Consistency among Temporal Constraints of input and output ports and related clause – Temporal constraints of a reduced-form clause and related input and output ports must be consistent with each other according to specific formulae.*

A set of relationships among these temporal constraints must be satisfied as described in the sequel. We consider now an input, i, and output, o, having as time bounds $[i.R_m, i.R_M]$ and $[o.R_m, o.R_M]$, respectively; these are related by means of clause φ with temporal constraint $[\varphi.T_m, \varphi.T_M]$. Given the above defined time bounds, when an input event occurs the object will provide an output message within interval $[\varphi.T_m, \varphi.T_M]$. According to the perviously introduced graphical representation, in Fig.2 (on the left), the relationships between possible occurrences of input and output events on the basis of the clause are depicted. If an input event has occurred ("last i" in the figure) producing its corresponding output (last o), then the interval during which the next input (output) event may occur is represented by a heavier line along the input (output) axis.

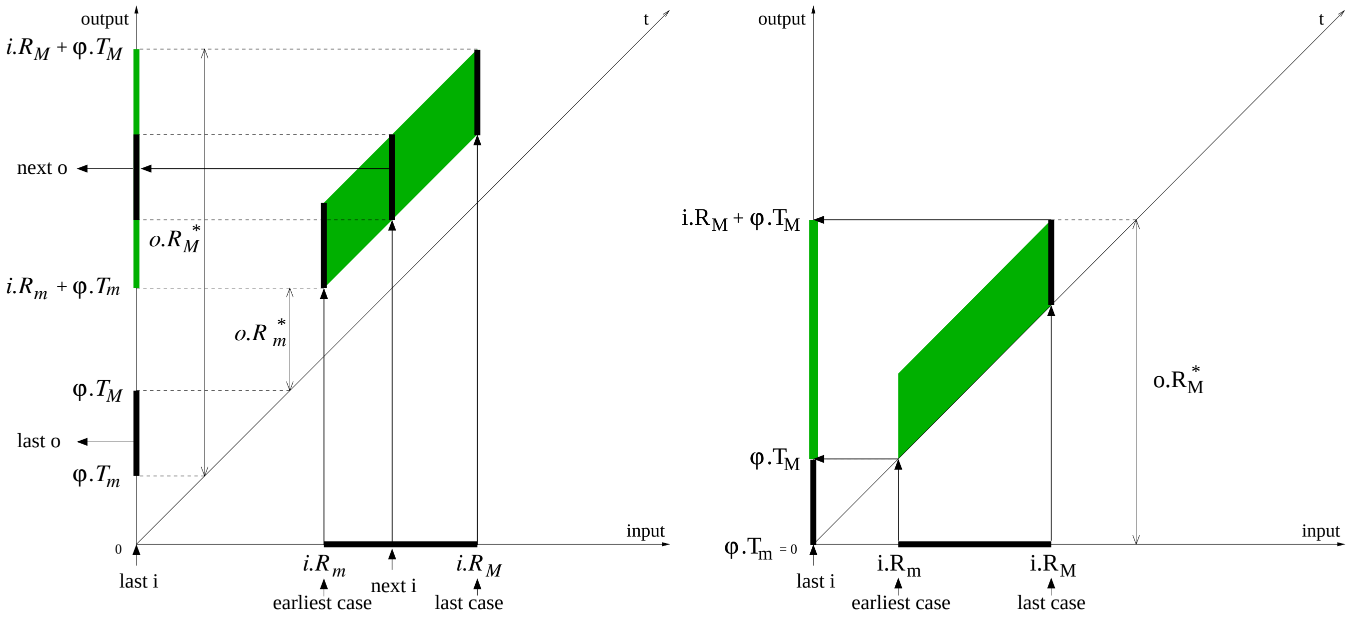


Figure 2. Relationships among temporal constraints associated with input, output and related clause: (i) general case (on the left), (ii) limit case (on the right).

According to TROL model, the next input will arrive when at least $i.R_m$ time units are elapsed since its last occurrence. Since the next input event is constrained to occur within $i.R_M$ time units, then conditions on the time bounds of outputs can be evaluated. According to the "earliest" case, when the next input event exactly occurs after $i.R_m$ time units, the minimum time bound for the output is evaluated to be:

$$o.R_m^* = i.R_m + \varphi.T_m - \varphi.T_M. \tag{2}$$

On the contrary, for the "last" case, when the next input event exactly reaches the object after $i.R_M$ time units, the maximum time bound for the output is evaluated to be:

$$o.R_M^* = i.R_M + \varphi.T_M - \varphi.T_m. \tag{3}$$

Equations (2) and (3) state the relationships among the temporal constraints of input, output and clause. These can be used for evaluating a missing temporal constraint when the other two are known.

For safeness, the $o.R_m$ declared by the analyst has to be lower or equal than $o.R_m^*$. This constraint is needed since: if the bound suggested by the user is grater than $o.R_m^*$, the component could produce a value out of the time bounds imposed. In the same manner, the $o.R_M$ declared has to be grater or equal than $o.R_M^*$ to avoid the specification of unsuitable temporal constraints. Therefore, the following conditions hold:

$$\begin{aligned} o.R_m &\leq i.R_m + \varphi.T_m - \varphi.T_M, & (4)\\ o.R_M &\geq i.R_M + \varphi.T_M - \varphi.T_m. & (5) \end{aligned}$$

According to Fig.2, to avoid overlapping between two consecutive input events before the first is served, the temporal constraints of clause φ and related input must satisfy the following condition:

$$0 \leq \varphi.T_m \leq \varphi.T_M \leq i.R_m. \tag{6}$$

As a limit case, according to equation (6) the temporal bounds of a clause can assume the following values $\varphi.T_M = i.R_m$ and $\varphi.T_m = 0$ (see Fig. 2 on the right). By replacing these values in equations (2) and (3), the following equations: $o.R_m^* = 0$ and $o.R_M^* = i.R_M + i.R_m$, are obtained. This means that, in the earliest case, the object instantaneously reacts, while in the last case, it produces the output simultaneously with the arrival of the next input event. If $\varphi.T_M > i.R_m$, the successive events on the outputs can be overlapped.

Therefore, in order to have more realistic and safer relationships for the class, condition (6) should be modified, by assuming a reaction time always greater than zero and $\varphi.T_M < i.R_m$. Then condition (6) becomes:

$$0 < \varphi.T_m \leq \varphi.T_M < i.R_m, \tag{7}$$

As a consequence, **Criterion 6** states that: *for each reduced-form clause (direct or reverse) φ, the temporal constraints of the clause and those associated with its related input and output ports must satisfy conditions (7), (4) and (5):*

$$\forall \varphi \in \Psi : \quad i \in port(FC) \wedge o \in port(CC) \quad \wedge \quad (\varphi.W = direct) \wedge (0 < \varphi.T_m \leq \varphi.T_M < i.R_m) \wedge$$
$$(o.R_m \leq i.R_m + \varphi.T_m - \varphi.T_M) \wedge (o.R_M \geq i.R_M + \varphi.T_M - \varphi.T_m);$$

while for reverse clauses:

$$\forall \varphi \in \Psi : \quad i \in port(FC) \wedge o \in port(CC) \quad \wedge \quad (\varphi.W = reverse) \wedge (0 < \varphi.T_m \leq \varphi.T_M < o.R_m) \wedge$$
$$(i.R_m \leq o.R_m + \varphi.T_m - \varphi.T_M) \wedge (i.R_M \geq o.R_M + \varphi.T_M - \varphi.T_m);$$

□

Criterion 6 must be verified by each class clause. This means that, on its basis, some changes to the temporal constraints of ports of clauses may be needed in order to obtain its verification (thus constraining the specification to be consistent and more effective). These changes can lead to diffuse changes along all class temporal constraints; thus, an iterative verification of all class clauses is needed.

When a direct clause and a reverse clause are specified over the same ports a special condition have to be considered. In this case, conditions of Criterion 6 are not sufficient to verify the consistency of time bounds for input and output. In fact, the condition for direct clauses ensures the output bound if the input time bounds are guaranteed; respectively, the condition for reverse clauses ensures input time bounds if the output time bounds are guaranteed. Therefore, this bi-directional condition between input and output bounds does not allow the application of the above criteria. In this condition, $o.R^*_m$ is the sum of the minimum reaction time bounds of the reverse and direct clauses, and $o.R^*_M$ is the sum of the maximum reaction time of the two clauses. For the input time bounds the same condition hold:

$$o.R_m = i.R_m \leq \varphi.T_m + \varphi_r.T_m, \tag{8}$$

$$o.R_M = i.R_M \geq \varphi.T_M + \varphi_r.T_M, \tag{9}$$

where, φ is a direct clause from input i to output o, and φ_r is a reverse clause from output o to input i.

The above and other less relevant criteria are used in TOOMS to verify in detail the completeness and consistency of the External Specification of all class clauses.

3.2 Verification of High Level Behavior

This section presents the analysis of the high-level description enforced in the External Specification of the class. A high-level specification is given by means of the so-called High-Level Clauses, HLCs. Each HLC implicitly specifies a sequence of events on inputs/outputs and the related relationships described by means of a set of reduced-form clauses. Moreover, each HLC can be regarded as a sort of trace, history-trace, or message sequence chart of scenarios such as in ROOMChart or in OSDL [16], [9], [5]. An HLC can be obtained for the conjunction of reduced-form clauses or can be directly given on the basis of the system requirements. In this last case, their verification against the External Specification can be regarded as a validation.

Before identifying the HLCs of the class and applying the related criteria, the class reduced-form clauses have to pass the criteria presented in the previous sub-sections.

An HLC can be formally expressed as an ordered set of class reduced-form clauses, $\{\varphi_j | j = 1, 2, \ldots, n_\varphi\}$. Since a direct clause can activate a reverse clause and vice versa, the ordered set is built by direct and reverse clauses in alternated manner. Direct clauses describe the class behavior, while reverse clauses describe how the environment (in which the object of a class is used) reacts to class requests. Thus, each couple of consecutive clauses belonging to an HLC must satisfy the following condition:

$$\forall i \in [1, n_\varphi - 1] : \ \varphi_i \in HLC \wedge \varphi_i.W \neq \varphi_{i+1}.W \wedge (\varphi_i \Rightarrow \varphi_{i+1}).$$

The first part specifies that consecutive clauses must be different in type (direct/reverse), while "$\Rightarrow$" states that φ_i activates φ_{i+1}. A clause activates another clause when its CC makes true the AC of the second clause (also considering the class status in the OC of both clauses). Please note that no class clause activates the first clause, since the activation of the sequence only depends on the external events. Formally, the set of HLCs of a class is modeled as $\mathcal{HLC} = \{hlc_1, hlc_2, \ldots, hlc_{n_{hlc}}\}$. As a limit case, the simplest HLC is a stand alone reduced-form clause.

An HLC can have a subset of clauses which may result in a cyclic loop of activations. Although suitable messages may activate other actions, this condition could lead to a live-lock and thus should be carefully analyzed. In certain contexts, the presence of such a condition is functional for the application (e.g., a protocol of hand-shaking), for this reason the non-satisfaction of this criterion results in a warning alarm for the analyst. This fact is specified by the following criterion.

Criterion 7 – *Cyclic sequence of class clauses – A clause, φ_j, belonging to a HLC, should not activate any other previous clause in the HLC itself:*

$$\forall \varphi_i, \varphi_j \in \{\varphi_1, \ldots, \varphi_{n_\varphi}\} : (j > i) \wedge (\varphi_i.W \neq \varphi_j.W) \wedge \neg(\varphi_j \Rightarrow \varphi_i).$$

In some cases, Criterion 7 can be false without causing any problem in the specification – e.g., a cyclic sequence used for specifying communication protocols as shown in Section 4. □

Moreover, two HLCs: $hlc_i, hlc_j \in \mathcal{HLC}$, are in conflict with each other if their sub-sequences of clauses are simultaneously activated and produce compatible results. This can occur even if their reduced-form clauses are decoupled (not in conflict). For example, let us now consider a couple of simple sequences of clauses formed by a sequence of reduced-form clauses (φ_1, φ_2, φ_2 and φ_a, φ_b, φ_c) as follows:

φ_1	:	`A=5` $\rightarrow$ `E=1;`	φ_a	:	`A=5` $\rightarrow$ `F=8;`
$\varphi_2(reverse)$	:	`E=1` $\rightarrow$ `C=10;`	$\varphi_b(reverse)$	:	`F=8` $\rightarrow$ `D=81;`
φ_3	:	`C=10` $\rightarrow$ `B=5;`	φ_c	:	`D=81` $\rightarrow$ `B=6;`

Both sequences of clauses are activated as soon as a message having a value equal to 5 reaches input A; hence, φ_1 and φ_a are decoupled (not in conflict) since these generate different actions. Please note that φ_1 and φ_a are capable of activating φ_2 and φ_b, and these in turn activate φ_3 and φ_c, respectively. In this case, φ_3 and φ_c are decoupled (not in conflict) since they have different ACs. By considering sequences $\varphi_1, \varphi_2, \varphi_3$ and $\varphi_a, \varphi_b, \varphi_c$, the derived HLCs are in conflict since the same firing condition leads to produce CCs which are in conflict (i.e., `B=5` and `B=6`) and, thus, non-acceptable because they are inconsistent.

For the above reasons the following criterion has been introduced for detecting conflicts in each couple of HLCs of the class when this is not satisfied.

Criterion 8 – *Consistency between subsequences of HLCs – In $\mathcal{HLC}$ each couple of HLCs: hlc_m, hlc_n, must not be in conflict by considering also their sub-sequences – i.e., decoupled clauses:*

$$\forall hlc_m, hlc_n \in \mathcal{HLC} : \forall \varphi_i, \varphi_j \in hlc_m : \ (j > i) \wedge \forall \varphi_k, \varphi_l \in hlc_n : \ (l > k) \wedge$$
$$\varphi_j.W = \varphi_l.W \wedge \varphi_i.W = \varphi_k.W \wedge clauses\_decoupled(\Lambda(\{\varphi_i, ..., \varphi_j\}), \Lambda(\{\varphi_k, ..., \varphi_l\}));$$

where: $\{\varphi_i, ..., \varphi_j\}$ is a subsequence of ordered clauses in hlc_m, $\{\varphi_k, ..., \varphi_l\}$ is a subsequence of ordered clauses in hlc_n, and $\Lambda()$ is a function for obtaining a reduced-form clause φ from a HLC ($HLC \rightarrow \varphi$):

$$\Lambda(\{\varphi_i, ..., \varphi_j\}): \qquad (\Lambda.AC = \varphi_i.AC) \ \wedge \ (\Lambda.CC = \varphi_j.CC).$$

□

From the computational point of view, it should be noted that if N is the number clauses of a *hlc*, the number of possible clause subsequences is $N(N-1)/2$ and the number of subsequences starting and ending with a certain type of clause (normal or reverse) is approximately $N^2/8$. Therefore, by using force brute algorithm to check the above criterion between two HLCs the asymptotical complexity is an $O(N^4)$ and for checking all the possible pairs of HLCs the asymptotical complexity is an $O(M^2N^4)$ where M is the number of HLCs in $\mathcal{HLC}$ and N is their length in terms of clauses.

4 The External Specification of a Cellular Phone

In this section, the External Specification of a cellular phone system is used to illustrate the application of the criteria presented in the previous sections.

The following functionalities of the cellular phone have been extracted from the textual specifications: *"The mobile phone receives and sends calls by means of a telecommunication channel (with the possibility of storing a set of call-numbers through an agenda mechanism) (see Fig.3). The connection with the mother station of the current cell is obtained and maintained according to the standard protocol ETACS*[1]*. According to that protocol, the frequency can be automatically changed and two main systems for controlling the communication quality are provided (the measure of: the main information, and that of the tone called* supertone *at 6 KHz). The user interface of the cellular phone is comprised of (i) an LCD display on which the information about the phone status (call, ring, etc.) and the number called are shown (messages as battery down and the low-frequency status are shown on special areas), (ii) a keyboard with buttons and switches, to dial the number, to pass from normal to agenda status and vice-versa, to adjust the volume, to switch off/on the cellular phone, etc. The phone provides a beeper for keys, ringing and alarms."* From the detailed requirements the most important clauses describing the behavior of the cellular phone can be identified.

[1]ETACS (Extended Total Access Communication System, 900 MHz): adopted in Italy and derived from the American standard AMPS (Advanced Mobile Phone Service). This system has been specified in the context of a joint collaboration between the University of Florence and OTE S.p.A. [35], [40].

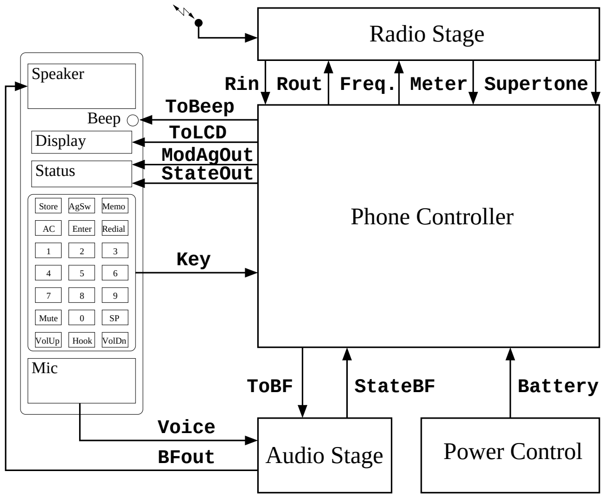


Figure 3. The cellular phone with its components.

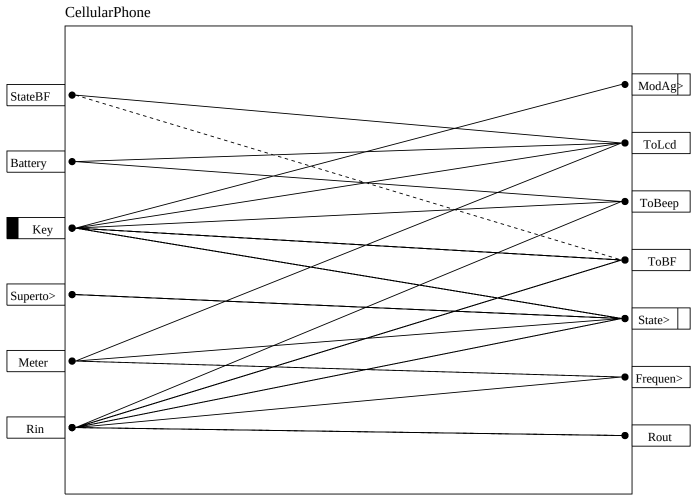
4.1 Cellular Phone: TOOMS/TROL External Specification

The External Specification of the Phone Controller is assigned to the main system/class as reported in TOOMS/TROL notation in Figs. 4 and 5 (the whole TROL specification can be found in [40], while a partial description has been reported in [35]). Please note that we have specified subsystem Phone Controller as class `CellularPhone`. The specification reported in Fig. 4 presents some problems of consistency and completeness which have been detected during the verification phases, as shown in the rest of the section. These cannot be easily detected by merely observing the specification.

As the External Specification of class `CellularPhone` passes the syntax verifications, clauses specified by the analyst are collected as reported in Fig. 5. According to TROL model, dashed lines represent clauses called `reverse`, by specifying that the corresponding output sends a message in order to receive another message from the related input.

Please note that according to TROL model and language, `New()` and `Ready()` are special functions. These specify the conditions in which a **new** message is received by an input port and a message is **ready** to be transmitted by an output port, respectively. This allows the writing of more abstract clauses by merely specifying the presence of messages without the need of identifying their specific or possible values. For example, clause `Link` specifies that a message received on `Rin` port always produces a message on `StateOut`. A more detailed clause of this general property is `NewCall2`.

Let us now describe some of the clauses presented in Fig.5. During a call, we have `StateOut=Talking`, thus:




```
multitype TypeLcd = { Message: String; BFLcd:TypeStateBF; BattLcd:TypeBattery; LinkLcd: Real; TKey: Character};

typedef TypeStateBF = {Vol:Integer; Mute:Boolean; Voice:Boolean};

typedef TypeFrequencies = {UpLink:Real; DownLink:Real};

typedef {TypeBattery = Integer};

enum TypeCtrlBF {VolUp, VolDown, Mute, SP_Phone, Busy, Free};

enum TypeState {Init, Idle, Ringing, Talking, Waiting};

enum TypeBeep {Beep, Ring, RingBatt, Off};


Provided_services: // Inputs:

    Rin [3000,4000]: String;             // Frame from Radio Receiver (DownLink),ETACS

    Meter [3000,4000]: Real;             // Level of Power in High Frequency

    Supertone [3000,4000]: Real;         // Quality of Connection at Low-Freq (6Khz)

    Key buffered [3000,4000]: Character; // (Buffered) Keys from Keyboard, Number, HOOK, etc

    Battery [3000,4000]: Integer;        // Power-Level of Battery

    StateBF [3000,4000]: TypeStateBF;    // Status of Hardware Modules at Low Frequency (Audio Stage)

Required_services: // Outputs:

    Rout [3000,4000]: String;                 // Frame for Radio Transmitter (UpLink), ETACS

    Frequencies [500,5000]: TypeFrequencies;  // Possible values for UpLink and DownLink Frequencies

    StateOut available [500,5000]: TypeState; // (available) Observable Phone Status

    ToBF [500,5000]: TypeCtrlBF;              // Controls Hardware Modules at Low Frequency (Audio Stage)

    ToBeep [400,5100]: TypeBeep;              // Controls of Beeper: On, Off

    ToLcd [1000,6000]: TypeLcd;               // Messages for the LCD Display

    ModAgOut available [1000,6000]: Boolean;  // (available) Phone (FALSE) or Agenda (TRUE)
```

Figure 4. Early version of the TROL External Specification of class `CellularPhone`: plot generated by TOOMS (in the plot, long names cut by TOOMS tool are marked with >).

```
Dropvox                 :  Supertone < LOWER_TONE_QUALITY ∧ StateOut=Talking → StateOut=Init -- [200,300];
NormalTalking           :  Supertone ≥ LOWER_TONE_QUALITY ∧ StateOut=Talking → StateOut=Talking -- [200,300];
NormalInit              :  Supertone ≥ LOWER_TONE_QUALITY ∧ StateOut=Init → StateOut=Init -- [200,300] ;
Dropdata                :  Meter < LOWER_QUALITY ∧ StateOut ≠ Talking → StateOut=Init --[200,300];
Echo                    :  Key ≥ '0' ∧ Key ≤ '9' ∧ StateOut=Idle → ToLcd=TKey -- [10,50];
Echokey                 :  New(Key) ∧ StateOut ≠ Init → ToBeep=Beep -- [10,50];
NewCall                 :  Rin=NEWCALL ∧ StateOut=Idle → Rout=ACKCALL -- [350,500];
NewCall3 (reverse)      :  Rout=ACKCALL ∧ StateOut=Idle → Rin=OKNEWCALL -- [350,500];
NewCall2                :  Rin=OKNEWCALL ∧ StateOut=Idle → StateOut=Ringing -- [300,400];
Ringing                 :  StateOut=Ringing → ToBeep=Ring -- [200,300];
Modeswitch              :  Key=AgSw ∧ StateOut=Idle ∧ ¬ ModAgOut → ModAgOut -- [10,50];
Modeswitch2             :  Key=AgSw ∧ ModAgOut → ¬ ModAgOut -- [10,50];
Display                 :  New(StateBF) → ToLcd=BFLcd -- [10,50];
Update (reverse)        :  Ready(ToBF) → New(StateBF) -- [10,50] ;
Hooking1                :  Key=Hook ∧ (StateOut=Talking ∨ StateOut=Waiting) → StateOut=Idle -- [200,300];
Hooking3                :  Key=Hook ∧ StateOut=Ringing → StateOut=Talking -- [100,200];
Link                    :  New(Rin) → Ready(StateOut) -- [20,60];
Acknowledgment          :  New(Rin) → Ready(Rout) -- [350,500];
Start (reverse)         :  Ready(Rout) → New(Rin) -- [350,500];
Handover                :  Rin=NEWFREQ → Ready(Frequencies) -- [10,50];
BatteryOff              :  Battery < LOWER_LEVELBAT → ToLcd=BattLcd -- [100,150];
BatteryDown             :  Battery < LOWER_LEVELBAT → ToBeep=RingBatt -- [100,150];
BeepOff                 :  StateOut=Idle → ToBeep=Off -- [200,300];
VolumeUp                :  Key=Volup ∧ StateOut=Talking → ToBF=VolUp -- [40,100];
VolumeDown              :  Key=Voldown ∧ StateOut=Talking → ToBF=VolDown -- [40,100];
Mute                    :  Key=Mute ∧ StateOut=Talking → ToBF=Mute -- [40,100];
SP_Phone                :  Key=SP ∧ StateOut=Talking → ToBF=SP_Phone -- [40,100];
CallerWait              :  Key=Enter ∧ StateOut=Idle → StateOut=Waiting -- [40,100];
CalleeBusy              :  Rin=Busy ∧ StateOut=Waiting → ToBF=Busy -- [40,100];
CalleeFree              :  Rin=Free ∧ StateOut=Waiting → ToBF=Free -- [40,100];
Calling                 :  Key=Enter ∧ Key=Hook ∧ StateOut=Waiting → Ready(StateOut) -- [40,100];
Answering               :  Key=Hook ∧ StateOut=Ringing ∧ StateOut=Talking → Ready(StateOut)-- [40,100];
Wait                    :  New(Rin) ∧ StateOut=Waiting → Ready(ToBF) -- [60,200];
BFControl               :  New(Key) ∧ StateOut=Talking → Ready(ToBF) -- [60,200];
Talking                 :  New(Supertone) ∧ StateOut=Talking → Ready(StateOut) -- [40,100];
DisplayMeter            :  New(Meter) → ToLcd=LinkLcd -- [200,300];
NewFrequencies          :  Meter < LOWER_QUALITY ∧ StateOut=Talking → Ready(Frequencies) -- [400,500];
UpdateMeter (reverse)   :  Ready(Frequencies) → New(Meter) -- [100,300];
...........             :  ...........
```


Figure 5. A part of the clauses of class `CellularPhone` (second part of the previous figure).

- clauses VolumeUp, VolumeDown, Mute, SP_Phone express that different messages are sent to module Audio Stage for controlling low-frequency;
- clause Dropvox states that the communication is interrupted, when the signal power Supertone is under threshold LOWER_TONE_QUALITY;
- clause NormalTalking states that if Supertone is not under threshold LOWER_TONE_QUALITY, the phone status is maintained.

The last condition also holds when StateOut=Init as specified by clause NormalInit.

For clause Dropdata no communication is carried out (when StateOut $\neq$ Talking) and the quality of the connection is evaluated by means of a VU-meter which is maintained under control for maintaining the phone ready for an answer.

When the phone is not in Init status, clause Echokey specifies that pressing a key on keyboard produces a beep sound. Other clauses specify that the effects on output ToBeep are BeepOff, Ringing and BatteryDown, even if different sounds could be required.

When a CellularPhone is in Idle state the following clauses have been specified:

- clause Echo expresses that the phone makes the echo on the LCD display of each number pressed with a given fixed range of delay;
- clause NewCall establishes that when the phone receives a call, an acknowledgment is sent to the main station by means of Rout;
- clause NewCall2 states that when message OKNEWCALL is received the phone accepts the call by passing to the state of Ringing;
- clause NewCall3 defines the behavior of the main station for accepting a new call;
- clause Modeswitch states that the agenda mechanism is activated when the related button is pressed;
- clause BeepOff expresses that the phone beeper is usually off.

Please note that the above list of clauses is only a part of the complete External Specification. In the next sections, the proposed criteria are applied in order to detect problems of consistency and completeness on the External Specification of class CellularPhone.

4.2 Cellular Phone: Verification of External Specification

As can be easily verified, some of the above clauses are not yet in reduced form. For example, clause `Hooking1` is comprised of the following reduced-form clauses:

```
Hooking1a   :  Key=Hook ∧ StateOut=Talking → StateOut=Idle -- [200,300];
Hooking1b   :  Key=Hook ∧ StateOut=Waiting → StateOut=Idle -- [200,300];
```

The above clauses state that, when the caller hangs up, clause `Hooking1` (when `StateOut=Talking` or `StateOut=Waiting`) specifies that the phone returns to the `Idle` State. Please note that clause `Hooking3` (when the caller hangs up and `StateOut=Ringing`) expresses that the communication is established.

Once the syntax verification and the generation of reduced-form clauses are performed, the verification process can start.

4.2.1 Cellular Phone: Completeness of the External Specification

According to Criterion 1, each single element of the External Specification of class `CellularPhone` must be complete. For example, some verifications have been performed: all possible values of output `ToBeep` are set in the CCs of clauses `Echokey`, `BeepOff`, `BatteryDown` and `Ringing`. These clauses use different values/messages on the same port – i.e., `Ring` (for ringing), `RingBatt` (alarm when the battery level is down), `Beep` (when a key is pressed), and `Off` for turning off the beeper after `RingBatt` or `Ring`.

According to Criterion 2, the External Specification must be verified to be complete with respect to all class elements. In fact, all OCs of the class clauses must be present in at least one CC of class clauses. In class `CellularPhone`, outputs of type *available* are `StateOut` and `ModAgOut`. These have been defined as enumerate collections, and thus their possible values must be present at least once in both CCs and OCs of a clause of the External Specification.

The possible values of `StateOut`: `Init`, `Idle`, `Ringing`, `Talking`, and `Waiting` can be reached by the phone status (see Fig. 4) as it has been stated by CCs of clauses: `Dropvox`, `NormalInit`, `Dropdata`, `Hooking1a`, `Hooking1b`, `NewCall2`, `NormalTalking`, `Hooking3` and `CallerWait`. These are compatible with OCs of clauses described above. For example, $compatible(\texttt{StateOut} = \texttt{Idle}, \texttt{StateOut} = \texttt{Idle})$ is true for OC of clauses `NewCall` and `NewCall2` with CC of clauses `Hooking1a` and `Hooking1b`:

```
NewCall     :  Rin=NEWCALL ∧ StateOut=Idle → Rout=ACKCALL -- [350,500];
NewCall2    :  Rin=OKNEWCALL ∧ StateOut=Idle → StateOut=Ringing -- [300,400];
Hooking1a   :  Key=Hook ∧ StateOut=Talking → StateOut=Idle -- [200,300];
Hooking1b   :  Key=Hook ∧ StateOut=Waiting → StateOut=Idle -- [200,300];
```

On the other hand, Criterion 2 is also verified for clause `Modeswitch2` which can fire depending on a condition on the available output port `ModAgOut`; since its OC is compatible with CC of clause `Modeswitch`.

4.2.2 Cellular Phone: Consistency of the External Specification

By applying Criterion 3, it is verified that any clause of class `CellularPhone` is not in conflict with respect to the domain of its variables. For example, clause `Calling` is inconsistent, since the components of its condition for firing are not compatible (which key is pressed? `Hook` or `Enter`?), while clause `Answering` is not consistent since the components of its OC are not compatible with each other (in the meantime, the phone should be ringing and talking).

As regards Criterion 4, although some clauses may have compatible ACs, the cross-consistency depends on their CCs. For example, clauses `BatteryOff` and `BatteryDown` are decoupled (not in conflict), since when the battery power is under the battery level (`Battery` < `LOWER_LEVELBAT`), clause `BatteryOff` states that a message is sent to the LCD display, while clause `BatteryDown` specifies that a message is sent to the beeper of the phone. Predicate $clauses\_decoupled$(`BatteryOff`, `BatteryDown`) is true and, thus, Criterion 4 is satisfied.

Clauses `Acknowledgment` and `Link` present the same AC. The specific message from the radio stage (input `Rin`) can constrain the phone to react in different ways, but this is not specified in these abstract clauses. In particular, clause `Acknowledgment` sends an acknowledgement message and clause `Link` updates the phone status (output `StateOut`). All these clauses are decoupled.

Clauses `Echokey` and `BatteryDown` are inconsistent with respect to clause `BeepOff`, since they have compatible ACs and CCs and work on the same ports producing different results (Criterion 4 detects a problem). The beeper cannot remain quiet (off) when it is activated for the pressing of a key or for the presence of an alarm. The above clauses must be removed or modified. The second choice leads to:

```
BeepOff  :  StateOut=Init → ToBeep=Off -- [200,300];
```

At this point, this clause is still in conflict with clause `BatteryDown` that specifies that a sound is produced as soon as the battery level is too low disregarding the phone status. To this end, this last clause has been changed in:

```
BatteryDown  :  Battery < LOWER_LEVELBAT ∧ StateOut≠Init → ToBeep=RingBatt -- [100,150];
```

Please note that clause `Ringing` was not in conflict with both the previous and the updated versions of the clauses, since it has a compatible CC with clauses `BatteryDown` and `EchoKey` even though these present compatible ACs. This is allowed by Criterion 4.

4.2.3 Cellular Phone: Temporal Constraints of the Class

If Criterion 5 is not verified, the External Specification must be considered incomplete. In the case of class `CellularPhone`, all the temporal constraints have been specified. According to Criterion 6, these must be consistent with each other.

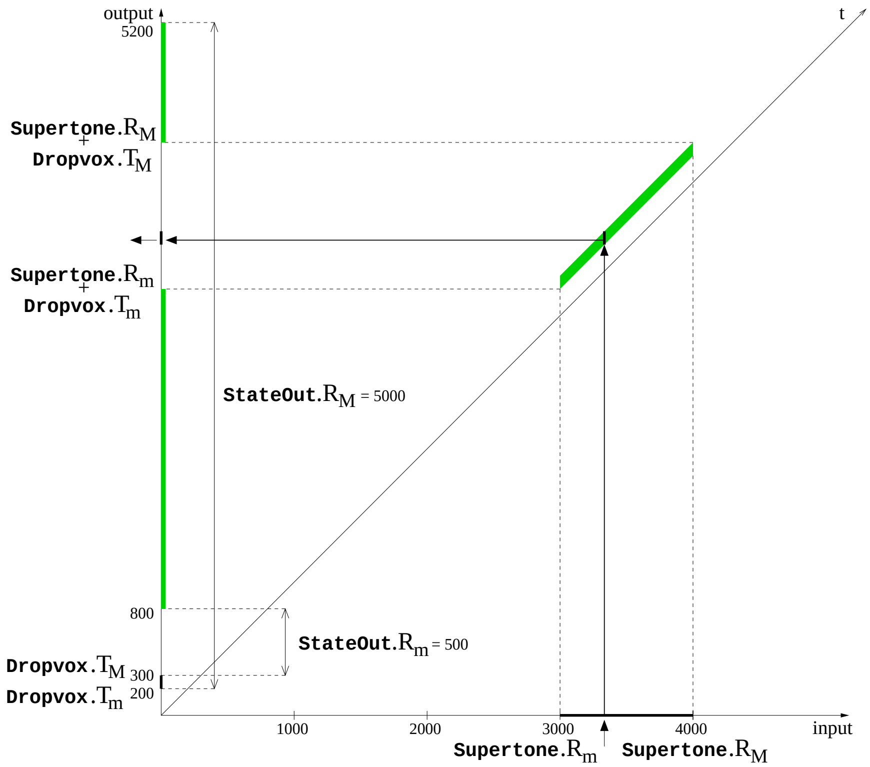


Figure 6. Temporal constraints of clause `Dropvox` of class `CellularPhone`.

For example, in Fig. 6, the temporal constraints of clause `Dropvox` and related input and output ports are depicted through the graphical representation introduced in Section 3.1.3. These must be verified according to conditions of Criterion 6. In clause `Dropvox`, when input `Supertone` is under the threshold (during a talk), the phone passes to the state `Idle` after at least 200 time units and no later than 300 time units. Therefore, Criterion 6 is satisfied with these values since:

$$0 < \texttt{Dropvox}.T_m \leq \texttt{Dropvox}.T_M < \texttt{Supertone}.R_m, \qquad 0 < 200 \leq 300 < 3000,$$
$$\texttt{StateOut}.R_m \leq \texttt{Supertone}.R_m + \texttt{Dropvox}.T_m - \texttt{Dropvox}.T_M, \qquad 500 \leq 2900,$$
$$\texttt{StateOut}.R_M \geq \texttt{Supertone}.R_M + \texttt{Dropvox}.T_M - \texttt{Dropvox}.T_m, \qquad 5000 \geq 4100.$$

The constraints of Criterion 6 to be satisfied are reported on the left, while the actual values are on the right. The values of the temporal constraints must be coherent with those obtained by verifying all the other clauses in which the same input and/or output ports are involved. In this case, clauses `NormalTalking`, `DropData`, `NormalInit`, `NewCall2`, `Hooking1a`, `Hooking1b`, `Hooking3`, etc., must be verified. For example, according to clause `Link` the phone status (output `StateOut`) is updated not earlier than 20 time units and not later than 60 time units. In this case, Criterion 6 is satisfied:

$$0 < \texttt{Link}.T_m \leq \texttt{Link}.T_M < \texttt{Rin}.R_m, \qquad 0 < 20 \leq 60 < 3000,$$
$$\texttt{StateOut}.R_m \leq \texttt{Rin}.R_m + \texttt{Link}.T_m - \texttt{Link}.T_M, \qquad 500 \leq 2960,$$
$$\texttt{StateOut}.R_M \geq \texttt{Rin}.R_M + \texttt{Link}.R_M - \texttt{Link}.T_m, \qquad 5000 \geq 4040.$$

After having performed a similar verification on the rest of class clauses for controlling their consistency, it resulted that Criterion 6 is not satisfied by clause `Update`. In particular we have:

$$0 < \texttt{Update}.T_m \leq \texttt{Update}.T_M < \texttt{ToBF}.R_m, \qquad 0 < 10 \leq 50 < 500,$$
$$\texttt{StateBF}.R_m \leq \texttt{ToBF}.R_m + \texttt{Update}.T_m - \texttt{Update}.T_M, \qquad 3000 \not\leq 460,$$
$$\texttt{StateBF}.R_M \geq \texttt{ToBF}.R_M + \texttt{Update}.T_M - \texttt{Update}.T_m, \qquad 4000 \not\geq 5040.$$

Therefore, the last conditions are not satisfied, and a solution can be to modify the temporal constraints of input `StateBF`. A modification of temporal constraints for input/output ports (and for reverse clauses) can be regarded as a revision of the external requirements of the system. A modification of the temporal constraint associated with a direct clause is a revision of the internal specification of the system (in fact, the temporal constraints of the clause specify bounds on the internal implementation of the class itself). For example, $\texttt{StateBF}.R_m = 450 \leq 460$ and $\texttt{StateBF}.R_M = 5100 \geq 5040$ satisfy the above constraints. Clause `Display` also depends on input `StateBF` and, with the new values for the temporal constraint, Criterion 6 is satisfied for `ToLcd` having `[400,5500]`. With these values, clauses `Echo`, `DisplayMeter` and `BatteryOff` are also verified.

Please note that in the External Specification there exists two pairs of clauses which describe the communication protocol between the Cellular Phone and the mother station – e.g., `Acknowledgment`, `Start`, and NewCall, NewCall3. These two pairs of clauses reference the same input and output ports, in that each couple presents both a direct and a reverse clause on the same input/output ports – i.e., Rin and Rout. This fact may produce inconsistency on temporal constraints which must be corrected in order to guarantee the right system behavior.

According to the early version of the specification, Criterion 6 is not satisfied for both pairs of clauses. For example, for the Acknowledgment and Start clauses the following relationships should hold:

$$0 < \texttt{Acknowledgment}.T_m \leq \texttt{Acknowledgment}.T_M < \texttt{Rin}.R_m, \qquad 0 < 350 \leq 500 < 3000,$$
$$0 < \texttt{Start}.T_m \leq \texttt{Start}.T_M < \texttt{Rout}.R_m, \qquad 0 < 350 \leq 500 < 3000,$$
$$\texttt{Rin}.R_m = \texttt{Rout}.R_m \leq \texttt{Acknowledgment}.T_m + \texttt{Start}.T_m, \qquad 3000 = 3000 \not\leq 700,$$
$$\texttt{Rin}.R_M = \texttt{Rout}.R_M \geq \texttt{Acknowledgment}.T_M + \texttt{Start}.T_M, \qquad 4000 = 4000 \geq 1000.$$

As can be noted on the right a constraint is violated. This can be solved by revising the specification, for example by specifying a temporal constraint of [700,4000] for both Rin and Rout. These values also solve similar problems related to clauses NewCall and NewCall3.

Clauses UpdateMeter and NewFrequencies also present similar problems:

$$0 < \texttt{UpdateMeter}.T_m \leq \texttt{UpdateMeter}.T_M < \texttt{Frequencies}.R_m, \qquad 0 < 200 \leq 300 < 500,$$
$$0 < \texttt{NewFrequencies}.T_m \leq \texttt{NewFrequencies}.T_M < \texttt{Meter}.R_m, \qquad 0 < 400 \leq 500 < 3000,$$
$$\texttt{Meter}.R_m = \texttt{Frequencies}.R_m \leq \texttt{NewFrequencies}.T_m + \texttt{UpdateMeter}.T_m, \qquad 3000 \neq 500, 3000 \not\leq 600, 500 \leq 600,$$
$$\texttt{Meter}.R_M = \texttt{Frequencies}.R_M \geq \texttt{NewFrequencies}.T_M + \texttt{UpdateMeter}.T_M, \qquad 4000 \neq 5000, 4000 \geq 800, 5000 \geq 800.$$

Two conditions are not satisfied. In this case, a solution can also be to modify some temporal constraints – e.g., by imposing [600,4500] for both Frequencies and Meter which satisfy the above conditions. Criterion 6 is also satisfied by clauses DisplayMeter, and Dropdata with the new values for the temporal constraint.

Even if these conditions have been corrected, the mechanism of producer/consumer established between the cellular phone and the outside environment must be better defined in the system implementation. In fact, problems may occur for the reception of more messages than those the consumer is able to cope with. A solution can be obtained by using one of the validation techniques mentioned in the introduction.

4.3 Cellular Phone: Verification of High Level Behavior

For the following steps, the high-level behavior of the class/system is captured by deriving HLCs from the External Specification of class CellularPhone.

Some sequences of clauses which subsequently activate each other have been identified. In particular, clauses Acknowledgment activates the reverse clause Start, etc. This means that as soon as the phone sends an acknowledgement to the Radio Stage, it is again constrained to set a new value in Rin within a time interval. In turn, clause Start activates other direct clauses (Link and Acknowledgment). Thus, the phone can send an acknowledgement to the radio stage or report the status (output StateOut) as well. Similarly, both clauses BFControl and Wait activate the reverse clause Update which in turn activates clause Display. In this way, when the low-frequency part of the phone is controlled or get a busy/free signal from the Audio Stage, the phone status is signalled in the LCD display. For each of these sequences, an HLC can be derived. Hence, some of the HLCs of CellularPhone are reported in the following:

$$
\begin{aligned}
\mathcal{HLC}_{\texttt{CellularPhone}} = \; & \{\{\texttt{Acknowledgment, Start, Link}\},\\
& \{\texttt{Handover, UpdateMeter, DisplayMeter}\},\\
& \{\texttt{NewFrequencies, UpdateMeter, DisplayMeter}\},\\
& \{\texttt{Wait, Update, Display}\},\\
& \{\texttt{Acknowledgment, Start, Acknowledgment}\},\\
& \{\texttt{BFControl, Update,Display}\}, \ldots\}.
\end{aligned}
$$

Please note that Criterion 7 is not verified by HLC {Acknowledgment, Start, Acknowledgment}. Such HLC expresses the phone capability for establishing the communication and monitoring the quality with the base station. Its consistency can be satisfied if the last Acknowledgment clause is removed from the HLC.

According to Criterion 8, no HLCs couple which are simultaneously activated must produce inconsistent effects, by considering all sub-effects which can be produced by each clause of the HLC. In particular, an HLC formed by {Handover, UpdateMeter, DisplayMeter} must be verified with respect to the ordered sequence {NewFrequencies, UpdateMeter, DisplayMeter} by considering the corresponding reduced-form clause of these HLCs. According to Criterion 8:

$$
clauses\_decoupled(\Lambda(\{\texttt{Handover, UpdateMeter, DisplayMeter}\}), \Lambda(\{\texttt{NewFrequencies, UpdateMeter, DisplayMeter}\})).
$$

For the definition of the above function, since $\neg compatible(\texttt{Rin=NEWFREQ}, \; \texttt{Meter<LOWER\_QUALITY})$ is true, the HLCs are not in conflict. The same verification is performed for each HLCs couple in $\mathcal{HLC}_{\texttt{CellularPhone}}$.

In Fig.7, the propagation of temporal constraints of the reduced-form clauses Handover, UpdateMeter, DisplayMeter is depicted. In this example, the propagation is due to the presence of temporal constraints on inputs Rin and Meter, outputs Frequencies and ToLcd, and the related temporal constraints of clauses Handover, UpdateMeter, and DisplayMeter. The grey areas below the diagonal line are delimited by both

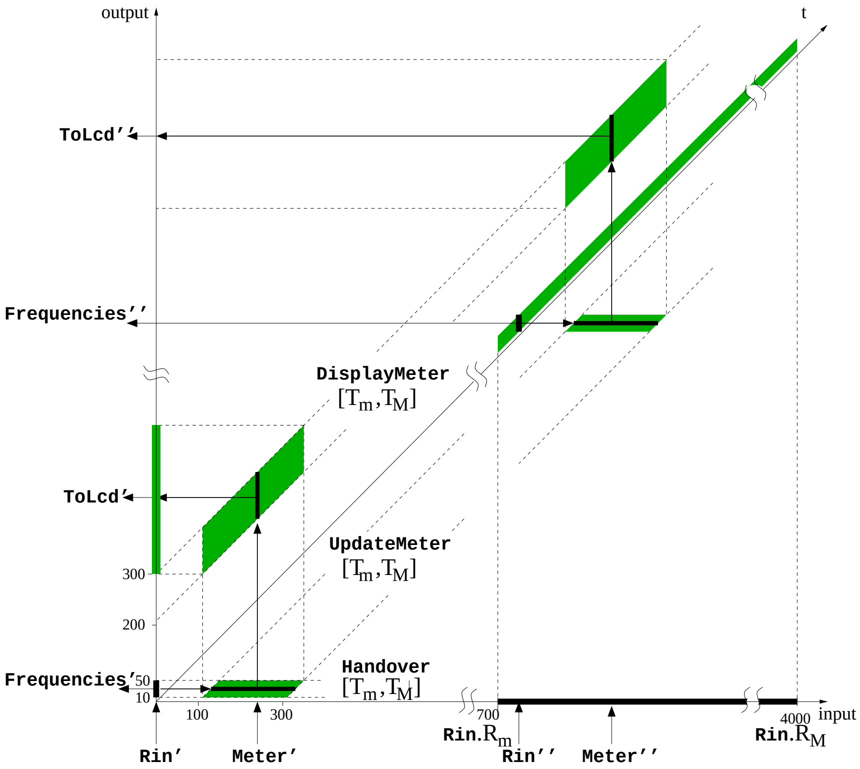


Figure 7. Temporal constraints of high-level clause $HLC_{\texttt{Handover\_DisplayMeter}}$ of class `CellularPhone`.

reaction time and output time bounds and correspond to all possible occurrences of events. This figure shows that the arrival of message `Rin` produces bounds on the production of `Frequencies` which in turn produces bounds on the arrival of `Meter`, and this further produces bounds on the production of `ToLcd`. The grey areas above the diagonal line are the bounds, while heavier lines in the grey areas are just an example given on the basis of a specific instant in which output `Frequencies` is produced.

In this case, it has also been shown that the Criteria and the graphical notation presented are very useful for detecting the problems and describing the relationships among features of the External Specification.

5 Conclusions

In this paper, some general criteria to verify the completeness and consistency of the External Specification of reactive systems have been described. This problem is frequently neglected since the External Specification is assumed to be correct during the validation process – e.g., model checking. The criteria defined can be easily applied to other approaches such as OSDL, ObjecTime, ObjectChart. In some of these models the addition of temporal logics or other formal tools for modeling time constraints is quite diffuse. In this paper, most of the criteria proposed allow to verify the consistency and the completeness of the behavioral and temporal aspects of the external specification.

The diffusion of approaches integrating denotational and operational aspects is becoming one of the most relevant improvement of recent specification models. These are typically called dual approaches – [1]. To fully exploit their potentialities they have to be supported by a suitable specification framework. TOOMS/TROL has been one of the first to move in this direction, presently several other dual methods are coming.

Companies that have used our approach have verified its innovative capabilities and power, observing that the effort spent in verification and validation along the development life-cycle is recovered in the phase of final validation in which typically less problems must be solved.

It has been shown that verifying the External Specification can be very useful for detecting problems existing in specifications during the early phases of the development life-cycle. When inconsistencies are detected in the External Specification, the specification analysis has to be focused on preventing anomalies with respect to both the outside environment and the definition of general system requirements. The analysis of External Specification is not only constrained to operate in the early phases, but whenever a subsystem is identified. In this context, the verification of the External Specification can be very useful during the system composition/decomposition process for verifying component completeness and consistency, at least at a high level, and for validating the internal implementation.

For these reasons, incompleteness and inconsistency detection in the External Specification allows the revision of the specification during the whole system development and can be considered as a valid tool for specification evolution management. In our experiences we have observed that this technique increases the general quality of specification and allows preventing problems of validation detecting inconsistency and incompleteness since the early phases of system specification. The approach can be used at each level of the system specification hierarchy. The approach has been used to assist analysts/developers in using a real middle-out approach (bottom-up and top-down) by integrating the concepts of rapid prototyping, reuse, continuous improvement and continuous verification and validation at each level of abstraction and specification detail.

6 Acknowledgements

The authors would like to thank Prof. G. Bucci for his encouragement and suggestions, and the reviewers for their contribution in making the paper stronger and clearer. A special thanks to the many people who

have built, tested, and used the CASE tool TOOMS.

References

[1] G. Bucci, M. Campanai, and P. Nesi, "Tools for specifying real-time systems", *Journal of Real-Time Systems*, vol. 8, pp. 117–172, March 1995.

[2] J. A. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems", *IEEE Computer*, pp. 10–19, Oct. 1988.

[3] S.-T. Levi and A. K. Agrawala, *Real-Time System Design*, McGraw-Hill Publishing Company, New York, USA, 1990.

[4] J. A. Stankovic and K. Ramamritham, *Advances in Real-Time Systems*, IEEE Computer Society Press, Washington, 1992.

[5] M. A. Bruno and P. Nesi, "Life-cycle of a object-oriented specification model for real-time systems", *Information and Software Technology*, vol. 41, no. 1, pp. 35–52, January. 1999.

[6] H. Thayer and M. Dorfman, *System and Software Requirements Engineering*, IEEE Compute Society Press, Los Alamitos CA, 1990.

[7] M. Alford, "SREM at the age of eight; the distributed computing design system", *Computer*, April 1985.

[8] A. Borgida, S. Greenspan, and J. Mylopoulos, "Knowledge representation as the basis for requirements specifications", *Computer*, pp. 82–91, April 1985.

[9] R. Braek and O. Haugen, *Engineering Real Time Systems: An object-oriented methodology using SDL*, Prentice Hall, New York, London, 1993.

[10] K. Lano and H. Haughton, *Object-Oriented Specification Case Studies*, Prentice Hall, New York, London, 1994.

[11] D. Carrington, D. Duke, R. Duke, P. King, G. Rose, and G. Smith, "Object-Z: An object-oriented extension to Z", in *Formal Description Techniques*, S. T. Voung, Ed. Elsevier Science, 1990.

[12] E. H. H. Dürr and J. vanKatwijk, "VDM++: A formal specification language for object-oriented designs", in *Proc. of the International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 7*, G. Heeg, B. Mugnusson, and B. Meyer, Eds. 1992, pp. 63–78, Prentice-Hall.

[13] G. Bucci, M. Campanai, P. Nesi, and M. Traversi, "An object-oriented dual language for specifying reactive systems", in *Proc. of IEEE International Conference on Requirements Engineering, ICRE'94*, Colorado Spring, Colorado, USA, 18-22 April 1994.

[14] A. Morzenti and P. SanPietro, "Object-oriented logical specification of time-critical systems", *ACM Transactions on Software Engineering and Methodology*, vol. 3, no. 1, pp. 56–98, Jan. 1994.

[15] D. Coleman, F. Hayes, and S. Bear, "Introducing ObjectCharts or how to use Statecharts in object-oriented design", *IEEE Transactions on Software Engineering*, vol. 18, no. 1, pp. 9–18, Jan. 1992.

[16] B. Selic, G. Gullekson, and P. T. Ward, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, Inc, New York, USA, 1994.

[17] D. C. Luckham and J. Vera, "An event-based architecture definition language", *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 717–734, Sept. 1995.

[18] A. M. K. Cheng, J. C. Brown, A. K. Mok, and R.-H. Wang, "Analysis of real-time rule-based systems with behaviornal constraint assertions specified in Estella", *IEEE Transactions on Software Engineering,*, vol. 19,, no. 9,, pp. 863–885,, Sept., 1993.

[19] D. Rosenblum, "A practical approach to programming with assertions", *IEEE Transactions on Software Engineering*, vol. 21, no. 1, pp. 19–31, Jan. 1995.

[20] R. Koymans, "Specifying real-time properties with metric temporal logic", *Real-Time Systems Journal*, vol. 2, pp. 255–299, 1990.

[21] R. Mattolini and P. Nesi, "Using TILCO for specifying real-time systems", in *Proc. of the 2nd IEEE International Conference on Engineering of Complex Computer Systems ICECCS'96*, Montreal, Canada, Oct. 1996, IEEE Press.

[22] R. Mattolini and P. Nesi, "An interval logic for real-time system specification", *IEEE Transactions on Software Engineering, in press*, 2000.

[23] L. Lamport, "A simple approach to specifying concurrent systems", *Communications of the ACM*, vol. 32, no. 1, pp. 32–45, Jan. 1989.

[24] L. Lamport, "TLA in pictures", *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 768–775, Sept. 1995.

[25] J. S. Ostroff and W. Wonham, "Modeling and verifying real-time embedded computer systems", in *Proc. of the 8th IEEE Real-Time Systems Symposium*, pp. 124–132. IEEE Computer Society Press, Dec. 1987.

[26] J. S. Ostroff, *Temporal Logic for Real-Time Systems*, Research Studies Press LTD., Advanced Software Development Series, 1, Taunton, Somerset, England, 1989.

[27] J. Armstrong and L. Barroca, "Specification and verification of reactive system behavior: The railroad crossing example", *Journal of Real-Time Systems*, vol. 10, pp. 143–178, 1996.

[28] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification", *in press, ACM Computing Surveys*, December 1999.

[29] B. W. Boehm, "Verifying and validating software requirements and design specifications", *IEEE Software*, vol. 1, no. 1, pp. 75–88, Jan. 1984.

[30] P. Bellini, M.A. Bruno, and P. Nesi, "Verification criteria for a compositional model for reactive systems", *Proc. of the IEEE International Conference on Complex Computer Systems*, Sept. 11-15 2000.

[31] P. Zave and M. Jackson, "Conjunction as composition", *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 4, pp. 379–411, Oct. 1993.

[32] S. C. Cheung and J. Kramer, "Context constraints for compositional reachability analysis", *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 4, pp. 334–377, Oct. 1996.

[33] J. J. Hooman and W. P. deRoever, "Design and verification in real-time distributed computing: an introduction to compositional methods", *Protocol Specification, Testing, and Verification Elsevier Science*, pp. 37–56, 1990.

[34] G. Bucci, M. Campanai, P. Nesi, and M. Traversi, "An object-oriented case tool for reactive system specification", in *Proc. of 6th International Conference on Software Engineering and Its Applications (sponsored by: EC2, CXP, CIGREF, and SEE)*, Le CNIT, Paris la Defense, France, 15-19 Nov. 1993.

[35] G. Bucci and P. Nesi, "Using TOOMS/TROL for specifying a cellular phone", in *Proc. of the 7th Euromicro Workshop on Real-Time Systems, EWRTS'95*, Odense, Denmark, June 1995, pp. 49–56, IEEE Press.

[36] P. Nesi and M. Campanai, "Metric framework for object-oriented real-time systems specification languages", *The Journal of Systems and Software*, vol. 34, pp. 43–65, 1996.

[37] A. C. Shaw, "Communicating real-time state machines", *IEEE Transactions on Software Engineering*, vol. 18, no. 9, pp. 805–816, Sept. 1992.

[38] M. Ben-Ari, *Mathematical Logic for Computer Science*, Prentice Hall, New York, 1993.

[39] M. Felder and A. Morzenti, "Validating real-time systems by history-checking TRIO specifications", in *Proc. of 14th International Conference on Software Engineering*, Melbourne, Australia, 11-15 May 1992, pp. 199–211, IEEE press, ACM.

[40] G. Conedera, L. Sentimenti, and P. Nesi, "Relazione finale: Analisi di un sistema cellulare OTE", Tech. Rep., Universita' di Firenze, OTE Srl, Florence, Italy, Gennaio 1994.

Authors' Biographies

Pierfrancesco Bellini was born in Florence, Italy, in 1969 He received his DrEng degree in Computer Science from the University of Florence, Italy in the field of formal methods. He is a Ph.D. candidate in software engineering and telecommunication at the University of Florence. His research interests include software engineering, formal methods, computer music and object oriented technologies.

Mario Adres Bruno Mario A. Bruno received a Bachelor Degree in Informatics Engineering from Universidad Tecnica Federico Santa Maria of Valparaiso, Chile, in 1990, and a Ph.D. degree in Computer and Telecommunications Engineering from University of Florence,Italy, in 1998, supported by a grant of the Italian Foreign Ministry. Dr. Bruno is currently a consultant on Object-Oriented Technology for industry. His research interests are in Object-Oriented software development and methodologies, Software Engineering, Formal Verification Techniques and Real-Time Systems.

Paolo Nesi was born in Florence, Italy, in 1959. He received his full degree in electronic engineering from the University of Florence, Italy, and the Ph.D. degree from the University of Padoa, Italy. In 1991, he was a visitor at the IBM Almaden Research Center, CA, USA. Since 1992, he is with the Dipartimento di Sistemi e Informatica, where he is Associate Professor for the University of Florence, Italy. He is active on several research topics: formal methods, object-oriented technology, real-time systems, system assessment, physical models, parallel architectures. He has been program chair or co-chair of several international conferences. He is the general Chair of IEEE International Conference on Software Maintenance, Florence, Italy, 2001. He is a member of the program committees of several international Conferences. He has been Guest Editor of special issues of international journals and is an editorial board member of journal and book series. He is the author of more than 120 technical papers. Nesi has been the project responsible and coordinator of several national and international research projects. He is a member of IEEE, ACM, IAPR, AIIA, and TABOO.

Contents

List of Figures

List of Tables

List of Footnotes

Manuscript received ...

P. Bellini, M. A. Bruno, P. Nesi — Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze This work was partially supported by the Italian Research Council, CNR (Consiglio Nazionale delle Ricerche).

1. ETACS (Extended Total Access Communication System, 900 MHz): adopted in Italy and derived from the American standard AMPS (Advanced Mobile Phone Service). This system has been specified in the context of a joint collaboration between the University of Florence and OTE S.p.A.