Formalizing Incremental Design in Real-time Area: SCTL/MUS-T *

Ana Fernández Vilas, José J. Pazos Arias, Rebeca P. Díaz Redondo and A. Belén Barragáns Martínez
Departamento de Ingeniería Telemática. Universidad de Vigo. 36200, Vigo, Spain.
{avilas,jose,rebeca,belen}@det.uvigo.es

Abstract

Achievement of quality in software design, while never easy, is made more difficult by the inherent complexity of hard real-time (HRT) design. Furthermore, timing requirements in HRT are by nature functional requirements, since system correctness depends on their fulfillment. Whereas the correctness dependence of the time imposes considering timing requirements from the early stages of the production process, complexity enforces a lifecycle model which fits in with requirements change and splits complexity by means of an incremental and iterative structure. Taking these aims as a starting point, this paper introduces SCTL/MUS-T methodology as supporting HRT design in a formalized and incremental way.

1. Introduction

Hard Real Time (HRT) design are, per se, intrinsically complex. In spite of such great complexity, there can not be any doubt that real-time formal methods have been delayed with respect to untimed ones, going back few years. This delay is largely due to integrating time as a functional requirement. If results are not produced in time, they are not just late, but wrong. Moreover design often depends on implementation details that are essential issues when deciding if it is feasible to fulfill timing requirements.

Nowadays, there is a broad consensus that dense-time formal methods are more expressive and suitable for composition and refinement since a quantum of time is not needed to fix a priori. Dense formal approaches to HRT have been carried out extending a large amount of formalisms studied at length. In the context of timed formal methods in general, and specifically of specification of real time systems in terms of timed process algebras and real-time logics, timed automaton provides a simple, and yet general, way to model the behavior of real-time systems. While timed automaton can be considered the standard timed state-machine, in the property approach a variety of logics have been applied to requirements specification. As far as analysis is concerned, formal techniques assuming dense time have their early origin in [1]. In this work, Alur et al introduced the first model checking algorithm for timed automaton with real-valued clocks.

Once maturity in timed formal methods has been reached, technology-transfer problems, which had been discovered in conventional untimed systems, appear once again. In order to get closer to the professional practice, formal methods should be incorporated into the software process. With regard to the structure of the software process, requirements, in large and complex systems, are often difficult to elicit at the beginning, since these requirements continuously evolve throughout the lifecycle. What is more, usually, in early phases, the designer has no a deep knowledge about the system and, in any case, the complexity could be excessive for a one-step design. In this respect, traditional cascade lifecycles are unlikely suitable. With the aim of improving software quality, SCTL/MUS methodology [6] advocates merging two different nature solutions: the formalization of the software process—gaining the advantages of formal methods—; and a process approach which is both iterative and incremental—fitting in with requirements change throughout the lifecycle—. In brief, SCTL/MUS methodology proposes the use of the many-valued temporal logic SCTL, and the model of unspecified states MUS, as a combination of formal methods to articulate a totally formal software process which is iterative, incremental and prototyping-based.

Due to complexity and criticalness, in HRT design it is specially advisable to inspect design alternatives, receive immediate feedback, provide prototypes from the early phases and automate as much the process as possible, allowing designers to focus on high-value activities. So, an incremental and iterative lifecycle, as the SCTL/MUS one, meets the production of this kind of systems. Apart from the lifecycle structure, timing requirements have to be taken into account from the early stages of the production process. Considering timing requirements late in the process often result in ad hoc changes to the system. On the basis of this

---

*This work was partially supported by the Xunta de Galicia Basic Research Project PGIDT01PX132203PR.
statement and in order to make SCTL/MUS methodology suitable for HRT area, it is totally needed extending both the property-oriented language (SCTL) and the model-oriented one (MUS) on which initial SCTL/MUS phases are based. Despite SCTL/MUS-T methodology (Timed SCTL/MUS) includes different kind of analysis and synthesis techniques, this paper focuses on the main ideas behind incremental design by means of verification-results evaluation and automated incremental synthesis.

2. Formalized Lifecycle

SCTL/MUS-T lifecycle defines a software process which is both iterative and incremental; and moreover, it relies on formal basis. Regarding formal specification, SCTL/MUS-T methodology takes a dual approach to design: model and property-oriented. Firstly, as model-oriented, we propose a state-transition formalism for the incomplete design model of the system, MUS-T (Timed Model of Unspecified States, section 3). Secondly, as property-oriented, we propose the real-time temporal logic SCTL-T (Timed Simple Causal Temporal Logic, section 4). Defining SCTL-T as many-valued allows us to support the explicit modeling of uncertainty and disagreement inherent to an iterative and incremental software process.

![Figure 1. Formalized lifecycle model](image)

The formalized lifecycle model consists, at a very abstract level, of three macro-phases. At the initial macro-phase (figure 1) the system is iteratively designed; a model-oriented specification MUS-T is obtained by incrementally incorporating timing scenarios identified on the target solution. These timing scenarios and general requirements are specified by means of the property-oriented formalism, SCTL-T. Outcome of the initial macro-phase is a set of MUS-T components which are transferred to the refinement and maintenance macro-phases.

On the basis of physical and logical architecture, the designer specifies the components of the system. For every component, tasks within a cycle in the initial phase are:

1. **Requirements and Scenarios Capture**: The designer identifies (by using SCTL-T) not only requirements, but typical scenarios of behavior of the components or the whole system that is being formally specified.
2. **Synthesis**: New SCTL-T scenarios, which lead to a growth in the system functionality, are incorporated (if feasible) into the system or into a single component that are modeled as MUS-T graphs. This task is automated by means of an incremental synthesis algorithm. Otherwise, inconsistency failure, counterexamples are computed.
3. **Validation**: The designer validates the model-oriented specification, MUS-T, by animation.
4. **Model Analysis (static)**: SCTL-T requirements are model checked over incomplete MUS-T components, that ones in the current iteration, or the whole specification in order to decide: if the model already satisfies the requirements; if it is not able to provide, in a future iteration, these requirements from the current design (inconsistency); or, if the model does not satisfy the requirements, but it is able to do it (incompleteness).
5. **Results Evaluation**: In case of inconsistency failure, counterexamples are symbolically simulated in order to decide which of the conflicting scenarios (or maybe the requirement) are error-prone. In case of incompleteness failure, the current model with supplied suggestions can be symbolically simulated in order to explore what alternative conforms to wishes.

3. MUS-T: Model for incomplete HRT systems

For modeling real-time systems we define MUS-T graphs (Timed Model of Unspecified States), timed extension of MUS models proposed in [6]. This extension defines a dense-time model similar to the timed automaton in [1], but event-driven. However, whereas atomic propositions on a location of a timed automata are assertions being true or false, in order to support incompleteness and consistency-checking in an incremental process, timed events in MUS-T can be characterized as possible (true), non-possible (false) or non-specified (unspecified).

3.1 Syntax

Before defining MUS-T graphs, some definitions concerning clocks and valuations are in order. A clock is a real-valued variable drawn from the finite set \( C \). A clock constraint \( \psi \in \Psi(C) \) is a boolean combination of atomic formulas of the from \( x < c \) or \( x \neq c \), where \( x \) and \( c \) are clocks, \( c \) is integer constant, and \( < \) is taken from \{\( \leq, \geq, <, >, = \} \). A clock valuation \( \gamma \) is a point in \( \Re^C \). If \( \gamma \) is a clock valuation, \( \gamma + \tau \) for some \( \tau \in \Re^+ \) stands for the clock valuation obtained by adding \( \tau \) to the value of every
clock. Given a clock constraint \( \psi \), \( \psi(\gamma) = 1 \) iff \( \gamma \) satisfies the clock constraint \( \psi \), otherwise \( \psi(\gamma) = 0 \). Finally, let \( \lambda \subseteq \mathcal{C} \) be a set of clocks, then \( \gamma_\lambda \) is the clock valuation obtained from \( \gamma \) by setting every clock in \( \lambda \) to 0.

A MUS-T graph \( \mathcal{M} \) is a 6-tuple \( \langle s_0, S, T, I, A, \mathcal{C} \rangle \) over \( \mathcal{L}_3 \), where:

- \( \mathcal{L}_3 = \{0, \frac{1}{2}, 1\} \) is the truth set which establishes the specification condition \(-\) (possible \(1 \)), non-possible \(0 \) or unspecified \(\frac{1}{2}\) of the transitions in the graph.
- \( A \) is a finite set of events.
- \( \mathcal{C} \) is a finite set of real-valued clocks.
- \( S \) is a finite set of locations, including two fictitious locations referred to as unspecified \( s_u \) and non-possible drain \( s_{np} \).
- \( s_0 \) is the initial location.
- \( T \subseteq S \times A \times \Psi(\mathcal{C}) \times 2^C \times S \) is the transition relation; and \( CS : T \rightarrow \mathcal{L}_3 \) is a total function which assigns a truth value (specification condition) to each transition.
- \( I_p \) and \( I_{np} : S \rightarrow \Psi(\mathcal{C}) \) are functions assigning to every location \( s \in S \) a progress invariance \( I_p(s) \), establishing progress of the time in the location; and a stop (non-progress) invariance \( I_{np}(s) \), establishing temporal contexts in which time cannot progress.

Every transition \( t \in T \) with \( CS(t) = 1 \) defines a possible transition \( (s, < a, g_p(s, a), \lambda >, s') \) with source location \( s \) and target location \( s' \). The timed event \( < a, g_p(s, a), \lambda > \) specifies an event \( a \in A \). A possible guard \( g_p(s, a) \in \Psi(\mathcal{C}) \) and a subset \( \lambda \subseteq \mathcal{C} \) of clocks to reset in the transition. In the same way, every \( t \in T \) with \( CS(t) = \frac{1}{2} \) defines a partially-unspecified transition \( (s, < a, g_{np}(s, a), \lambda >, s') \) with source location \( s \), target location \( s' \) and an unspecified guarded event \( < a, g_{np}(s, a), \lambda > \). Finally, transitions \( t \in T \) with \( CS(t) = 0 \) define non-possible transitions \( (s, < a, g_{np}(s, a), \{\} >, s_{np}) \), with the non-possible drain \( s_{np} \) as fictitious target location.

Specification conditions in MUS-T partition the valuation space \( \mathbb{R}^C \) in three subsets (possible, non-possible and unspecified). That partition is complete and disjoint. For completeness, the totally-unspecified guard \( g_{np}^T \) is defined:

\[
g_{np}^T(s, a) = -(\bigvee_{f} g_{np}(s, a)) \land -(\bigvee_{f} g_p(s, a)) \land -(g_{np}(s, a))
\]

In this way, valuation subsets defined by \( g_p, g_{np}, g_p^T \), and \( g_{np}^T \) cover \( \mathbb{R}^C \). Given a location \( s \) and an event \( a \), \( g_{np}^T(s, a) \) implicitly defines a totally-unspecified transition \( (s, < a, g_{np}^T(s, a), \{\} >, s_u) \). Also, the unspecified invariance, \( I_{np} \), is defined as \( I_{np}(s) = -I_p(s) \land -I_{np}(s) \); the subsets defined by \( I_p, I_{np} \) and \( I_u \) cover \( \mathbb{R}^C \).

The drain-locations \( (s_u, s_{np}) \) are the fictitious target locations of non-transitions in the MUS-T graph, i.e., non-possible transitions and totally-unspecified transitions. However, their nature is definitively different, the drain-location \( s_{np} \) is zero-evolution, and the drain-location \( s_u \) is maximal-evolution. That is, \( \forall a \in A, g_p^T(s_u, a) = true \) whereas \( g_{np}(s_{np}, a) = true \). Similarly, \( I_u(s_u) = true \) and \( I_{np}(s_{np}) = true \).

### 3.2. Semantics

Intuitively, a MUS-T graph operates by taking multi-valued transitions from location to location. Executing transitions takes no time. If no transitions are taken, time progresses by incrementing every clock by an arbitrary real number. The intent of clock constraints on locations is the following: the model can progress in a temporal way in a location \( s \), with specification condition \emph{true}, as long as \( I_p(s) \) is satisfied; with specification condition \emph{false}, as long as \( I_{np}(s) \) is satisfied in some point; otherwise with specification condition \emph{unspecified}. Clock constraints labeling transitions are enabling conditions, i.e., a transition \( t \in T \) can only be taken, with specification condition \( CS(t) \), if the constraints defining its enabling condition \( \langle g_p, g_{np}, g_p^T, g_{np}^T \rangle \) are satisfied. Also, when a transition in \( T \) occurs, all clocks in the set of clocks labeling it are reset to 0. In this way, a MUS-T graph is a multi-valued timed automaton.

The semantics for a MUS-T graph is given in terms of a multi-valued labeled transition system, which are often called dense due to the time domain. Formally, every MUS-T graph induces a dense multi-valued transition system \( \mathcal{S}_M = \langle (s_0, \gamma_0), ST = \{S \times \mathbb{R}^C \}, T, A, \mathcal{C} \rangle \), over the truth set \( \mathcal{L}_3 \), where \( ST \) is a set of timed states, i.e., pairs of the form \( (s, \gamma) \), \( s \in S, \gamma \in \mathbb{R}^C \); \( (s_0, \gamma_0) \) is the timed initial state with \( \gamma_0 \) assigning 0 to every clock; and \( T \subseteq ST \times (A \cup \mathcal{L}_3) \times \mathcal{L}_3 \times ST \) is the transition relation. The transition relation \( T \) identifies a source and a target timed state in \( ST \); a label \( a \in A \) (discrete transition) or \( \tau \in \mathbb{R}^+ \) (temporal); and a specification condition in \( \mathcal{L}_3 \).

### 4. SCTL-T: Requirements elicitation and incremental design

We will specify requirements and scenarios of incomplete HRT models using the logic SCTL-T (Timed Simple Causal Temporal Logic), a real-time extension of SCTL [6]. SCTL-T is a branching and dense real-time logic that fits within the explicit clock real-time logics. SCTL-T formulas match the causal pattern **Premise \( \mathcal{A} \otimes \text{Consequence} \)**. This generic causal formula establishes a causing condition (Premise); a temporal operator which determines the applicability of the cause (\( \mathcal{A} \)); a condition which is the effect (Consequence); and a quanti-
SCTL-T semantics is defined over the truth set $\mathcal{L}_0 = \{0, \frac{1}{2}, 1\}$; truth values in $\mathcal{L}_0$ have their origins in the MUS-T specification condition and the causal operation ($\rightarrow$) defined over $\mathcal{L}_0$ (figure 2), where:

$$
\rightarrow (a, b) = (a \lor \frac{1}{2}) \land \left( \neg a \lor \frac{1}{2} \right) \lor b
$$

![Figure 2. SCTL-T truth values](image)

$\mathcal{L}_0$ is a quasi-boolean lattice [3] (Hasse diagram of figure 2), with least upper bound $\lor$; greatest lower bound $\land$; and the unary operation $\neg$ defined by horizontal symmetry. The 4-tuple $(\mathcal{L}_0, \land, \lor, \neg)$ has the structure of the De Morgan algebra called the algebra of Middle Point Uncertainty (see [6] for details). The partial order relation defined in MPU is an order relation related to the degree of satisfaction. So 0 is the smallest truth degree, whereas 1 is the greatest. The values $\frac{1}{2}$ and $\frac{3}{4}$ are middle points in this order. $\frac{3}{4}$ is far from $\frac{1}{2}$, which is near them, and it can get to either. This is the reason why it is called algebra of Middle Point Uncertainty.

### 4.1. Syntax

A SCTL-T requirement $I_R := \phi$ establishes an identifier $I_R \in I_R$ and a SCTL-T formula $\phi \in \Phi_{SCTL-T}$. $\Phi_{SCTL-T}$ is given by the following grammar:

$$
\Phi_{SCTL-T} ::= \phi \lor \phi \mid \phi \land \phi \mid \neg \phi \\
\phi ::= \psi \mid \theta \mid \theta \lor \theta \mid \theta \land \theta \mid \theta \lor \theta \\
\psi ::= \psi \mid \psi \land \psi \mid \neg \psi \\
\theta ::= \theta \mid \theta \lor \theta \mid \theta \land \theta \\
\phi ::= \psi \land \phi \land \phi \mid \neg \phi \land \phi \land \phi
$$

where $\lor, \land, \neg$ are the usual path quantifiers adapted to multi-valued reasoning; $\lor$, the set of temporal operators; $\theta \in \Theta := \{true, false\}$, a propositional constant of state; $a \in A$, an event in the alphabet of a MUS-T graph; $\mathcal{C}$, the set of specification clocks; and $y$, the reset quantifier which binds and resets the specification clock $y$. Finally, $\psi \in \Psi$ is the set of clock constraints over $\mathcal{C} \cup \mathcal{E}$.

### 4.2. Applicability set and Quantified Causality

Temporal operators are used to reason about successors and predecessors of a given timed state. A temporal operator $\in \mathcal{C}$ fixes the order pattern between the timed state in which premise is formulated, and timed states in the scope of the consequence (applicability set). For every temporal operator, an applicability set, $\perp$, is defined in table 1. Thus, $\Rightarrow \perp, \Rightarrow \perp$ are discrete temporal operators (successor and predecessor) and $\Rightarrow \Rightarrow \Rightarrow \Rightarrow \perp$ are dense temporal operators, present, future and past respectively.

![Table 1. Applicability set $\perp$](image)

Table 1. Applicability set $\perp$

Whereas temporal operators determine the order pattern between the premise state and the consequence states, quantification determines the degree of satisfaction required in the consequence states. Thus, $\{\lor, \land\}$ provide existential and universal quantification over the applicability set. For quantified causality ($\mathcal{E} \otimes$), the truth degree of the quantified consequence depends on the specification condition of the transition that makes a timed state accessible in the causal formula, referred to as accessibility condition ($c_s$, table 1).

### 4.3. Satisfaction Relation

We start defining SCTL-T by giving the semantics of propositional constants. If $(s, \gamma, c_s)$ is a timed state, the satisfaction relation, $\vdash$, is as follows:

i) $\psi \in \Psi, \vdash (\psi, (s, \gamma, c_s))$ is the boolean satisfaction of the clock constraint $\psi$ in the valuation $\gamma, \psi(\gamma)$.

ii) $a \in A, \vdash (a, (s, \gamma, c_s)) = c_s \Rightarrow (s, \gamma, c^s_s) \Rightarrow (s', \gamma')$.

iii) $\theta \in \Theta, \vdash (true, (s, \gamma, c_s)) = c_s \Rightarrow (s', \gamma') \Rightarrow (s, \gamma, c_s)$ is satisfied in a state $(s, \gamma, c_s)$ iff this state is reached by a possible (discrete or temporal) transition ($c_s = 1$). On the contrary, $\vdash (false, (s, \gamma, c_s)) = false \Rightarrow (s, \gamma, c_s) = 0$. i.e., false is satisfied iff the state is reached by a non-possible transition ($c_s = 0$). Finally, $\emptyset$ is always satisfied ($\vdash (\emptyset, (s, \gamma, c_s)) = 1$).

We now proceed by defining the semantics of generic formulas. Given a MUS-T graph $\mathcal{M}$, SCTL-T formulas
are interpreted with respect to the multi-valued dense graph $S_{M'}$ induced by $M'$, which is $M$ with $C$ extended by all clocks $E$ mentioned in the SCTL-T formula. Given a SCTL-T formula, the satisfaction relation $\models$ is defined as:

$$\models (\Phi_{SCTL-T} \times (S \times 2^{\mathcal{C} \cup E} \times L_3)) \rightarrow L_0$$

where $\models (\Phi, (s, \gamma, c_a))$ assigns a truth value $\in L_0$ to the formula $\Phi$ evaluated in a state $(s, \gamma)$ with accessibility condition $c_a$. The complete satisfaction relation is as follows:

iv) $\models (y, \phi, (s, \gamma, c_a)) \equiv \models (\phi, (s, \gamma, c_a))$

v) $\models (\neg \phi, (s, \gamma, c_a)) \equiv \neg \models (\phi, (s, \gamma, c_a))$

vi) $\models (\phi \lor \phi', (s, \gamma, c_a)) \equiv \models (\phi, (s, \gamma, c_a)) \lor \models (\phi', (s, \gamma, c_a))$

vii) $\models (\phi \land \phi', (s, \gamma, c_a)) \equiv \models (\phi, (s, \gamma, c_a)) \land \models (\phi', (s, \gamma, c_a))$

viii) $\models (\phi \land \phi', (s, \gamma, c_a)) \equiv \models (\phi \land \phi', (s, \gamma, c_a)) \land \models (\phi, (s, \gamma, c_a))$

$$\models (\phi, (s, \gamma, c_a), \bigwedge_{i}^{n} c_{a_i} \lor \models (\phi', (s_i, \gamma_i, c_{si}'))$$

where $(s_i, \gamma_i, c_{si}') \in \bigoplus ((\otimes, \phi), (s, \gamma))$

ix) $\models (\phi \lor \phi', (s, \gamma, c_a)) \equiv \models (\phi \lor \phi', (s, \gamma, c_a)) \lor \models (\phi, (s, \gamma, c_a))$

$$\models (\phi, (s, \gamma, c_a), \bigvee_{i}^{m} c_{a_i} \lor \models (\phi', (s_i, \gamma_i, c_{si}'))$$

where $(s_i, \gamma_i, c_{si}') \in \bigoplus ((\otimes, \phi), (s, \gamma))$

Additionally the recursion operators $\ll I_R \gg$ and $\ll I_R \gg$ are understood respectively as the least and greatest fixed points. We use a fixed point character similar to the one used in [7]. In this way, usual operators in branching logics (for instance $\mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{M}, \mathcal{U}$) can be expressed in SCTL-T but with a multi-valued nature.

5. Examples

In this section we extract some examples of requirements specification, verification and incremental synthesis in SCTL/MUS-T methodology. Since the state space is infinite and dense, both verification and synthesis are computed over an exact abstraction obtained by minimization.

5.1. Outlining degrees of satisfaction

Degrees of satisfaction in SCTL-T express capability of a MUS-T graph to satisfy a requirement now or in future evolutions. The intuitive meaning of degrees of satisfaction $\in L_0$ of a SCTL-T formula is as follows:

- 0: It is falsified, $false$.
- 1: It cannot become $true$.
- 2: It cannot become $true$ nor $false$ (non-applicable).
- 3: It can become $true$ and $false$.
- 4: It cannot become $false$.
- 5: It is satisfied, $true$.

Figure 3. MUS-T graph for the examples

Figure 3 shows a MUS-T graph (unspecified transitions are drawn as dashed lines). In the following we illustrate several examples of requirements evaluated in every instance of the initial location $S_0$:

- $y.b \forall \Rightarrow (0 \Rightarrow a \land y < 3)$. Results are in the set $\{0, 1, 3\}$, i.e., the requirement is not falsified and, in instances in which it is applicable, the degree of satisfaction is $\frac{3}{4}$ (incompleteness failure), that is, every feasible evolution of the model does not falsify it.
- $y.b \forall \Rightarrow (0 \Rightarrow a \land y = 5)$. Results are in the set $\{0, 1\}$. In instances in which it is applicable, the degree of satisfaction is $\frac{3}{4}$ (inconsistency failure), since a state delayed $5$ time units from the $b$-transition is forbidden by the stop invariance. Every feasible evolution of the model cannot satisfy the requirement, and falsifying it is potentially possible by means of turning $b$ into possible at the location $S_0$.
- $y.a \forall \Rightarrow (0 \Rightarrow b \land y < 3)$. Results are in the set $\{\frac{1}{2}, \frac{1}{2}\}$, the requirement (when applicable) can be satisfied and falsified in the future since $b$ event is unspecified in every instance of the location $S_1$.

- Finally the requirement $a \forall \Rightarrow (0 \Rightarrow a) \land y = 2$ is falsified $(0)$ in instances of the location $S_0$, and $y.a \exists \Rightarrow (a \land y = 2)$ is satisfied $(1)$ in every instance in which it is applicable. These results will be the same in future evolutions of the model.

5.2. Supporting incremental design

Incremental synthesis is based on the following principle: changes in the model refer to specifying transitions which are unspecified in the semantic graph (from $\frac{1}{2}$ up to $1$, or from $\frac{1}{2}$ down to $0$). Any other change in the specification condition reflects an inconsistency failure. We use a imperative form of model checking which forces a truth degree $1$ for the new scenario to be synthesized. With the simple “declarative past causes imperative future” idea in [2], semantics is overloaded for synthesis, that is, the synthesis algorithm proceeds selecting synthesis contexts (states
luctance about formal methods, future directions in order to
in the model satisfying the declarative part) and, in these
cornerstone of this paper, adding the new future behavior specified by the
informal part. This is the intuition in the execution of the
system is preserved. Also, SCTL-T syntax is extended in order to allow the explicit management of model clocks
from the logic. That is, we introduce a imperative form of
bind \((a[x])\), which adds a clock \(x\) to \(C\) (if it does not exists)
and reset the clock in a discrete transition labeled by \(a\).

In figure 4 we show different examples of SCTL/MUS-T
incremental synthesis (scenarios in table 2) constructing a
MUS-T prototype from scratch. In the scenarios in table
2, \(ini\) is a SCTL-T constant referring to the initial state in the
semantic graph. Since synthesis rules (scenarios) are
global-scoped (invariances), it is necessary to reuse locations
in order to avoid MUS-T graphs indefinitely growing. In
brief, reuse criteria are defined as preserving simulations
and bisimulations, at different demanding levels, between
the non-reusing model and the reusing one.

In figure 4 we show different examples of SCTL/MUS-T
incremental synthesis (scenarios in table 2) constructing a
MUS-T prototype from scratch. In the scenarios in table
2, \(ini\) is a SCTL-T constant referring to the initial state in the
semantic graph. Since synthesis rules (scenarios) are
global-scoped (invariances), it is necessary to reuse locations
in order to avoid MUS-T graphs indefinitely growing. In
brief, reuse criteria are defined as preserving simulations
and bisimulations, at different demanding levels, between
the non-reusing model and the reusing one.

\[
\begin{align*}
R_1 := & x \quad \text{bind} \quad (x < 3 \Rightarrow \text{true}) \land (x > 3 \Rightarrow \false) \\
R_2 := & x \quad \text{bind} \quad (x \leq 4 \Rightarrow \true) \land (x > 4 \Rightarrow \false) \\
R_3 := & \text{not-applicable-rule} \\
R_4 := & \text{conflicting rule with } R_2 \text{ (footnote \textsuperscript{5})} \\
R_5 := & \text{by reusing location } S_0 \text{ with automatic clock reset} \\
\end{align*}
\]

\[\text{FIGURE 4(a)}\]

Table 2. Examples of synthesis rules

6. Conclusions and future work

With respect to other formal approaches proposed in the
literature, our lifecycle model is based on an iterative
and incremental structure in which real-time characteristics are considered from the beginning. One advantage
of this approach is early detection of timing failures.

\[\text{FIGURE 4(b)}\]

Also, as the system gradually takes form as more is learned
about the problem, alternative solutions can be explored.
In [4] it is showed a complete example which applies the
SCTL/MUS-T methodology to the steam-boiler case study.

To conclude, we believe that higher formalization of software engineering in practice should involve approaching
formal methods to current trends in software industry.
At this point, we believe incremental approach is a
major milestone. However, as well as formalizing an
iterative and incremental process intends to avoid industry reluctance about formal methods, future directions in order to

\[\text{FIGURE 4(c)}\]

achieve this goal should invest more effort in OO paradigm.

In this field we merely cite the works that include OO in
real-time logic specifications carried out in TRIO+ [5].

\[\text{FIGURE 4(d)}\]

\[\text{References}\]

[1] R. Alur, C. Courcoubetis, and D. Dill. Model Checking in

\[\text{FIGURE 4(a)}\]


\[\text{FIGURE 4(b)}\]

Methodology for Software Development of Distributed Sys-

\[\text{FIGURE 4(c)}\]

tems. A Case Study. Formal Aspects of Computing, 13:50–91,

\[\text{FIGURE 4(d)}\]


[7] O. V. Sokolsky and S. A. Smolka. Local Model Checking for

\[\text{FIGURE 4(a)}\]

Real-Time Systems. In Computer Aided Verification, 7th

\[\text{FIGURE 4(b)}\]

International Conference (CAV’95), volume 939 of LNCS,

\[\text{FIGURE 4(c)}\]


\[\text{FIGURE 4(d)}\]