TEPE: A SysML Language for Time-Constrained Property Modeling and Formal Verification

Daniel Knorreck, Ludovic Apvrille
Institut Telecom / Telecom ParisTech, LTCI CNRS,
2229 Route des Crêtes, B.P. 193, 06904 Sophia-Antipolis Cedex, France
{daniel.knorreck, ludovic.apvrille}@telecom-paristech.fr

Pierre de Saqui-Sannes
Université de Toulouse, LAAS-CNRS, ISAE,
10 av. Edouard Belin, B.P. 54032, 31055 TOULOUSE Cedex 4, France
pdss@isae.fr

Abstract

Using UML or SysML models in a verification-centric method requires a property expression language, a formal semantics, and a tool. The paper introduces TEPE, a graphical TEmporal Property Expression language based on SysML parametric diagrams. TEPE enriches the expressiveness of other common property languages in particular with the notion of physical time and unordered signal reception. TEPE is further instantiated in the AVATAR real-time UML profile. TTool, an open-source toolkit, implements a press-button approach for the formal verification of AVATAR-TEPE properties with UPPAAL. An elevator system serves as example.

1 Introduction

The increasing importance of real-time systems in life-critical applications has stimulated research work on modeling techniques that combine the friendliness of UML / SysML with the formality of verification tools such as UPPAAL. So far, the use of SysML in verification centric methods has been hampered by the poor formality of Requirement Diagrams and the lack of powerful property expression language. Thus, UML / SysML profiles commonly require the use of temporal logics (e.g., CTL) or the use of languages based on traces (e.g., the VSL language of MARTE [OMG08]) which are not always adequate to specify complex sets of sequential and parallel behaviors.

The paper extends SysML Parametric Diagrams to introduce TEPE[1], a graphic but formal language for describing logical and temporal properties. In TEPE, various design elements, such as blocks, attributes, and signals, can be combined together with logical (e.g., sequence of signals) and temporal operators (e.g., a time interval for receiving a signal) to build up complex but graphical properties. Moreover, TEPE may be introduced into the OMG-based SysML and a broad variety of SysML profiles. As a demonstration of this, we include TEPE in the real-time SysML profile - named AVATAR[2] - which is supported by TTool, an open-source toolkit interfaced with UPPAAL. The strength of the AVATAR-TEPE combination is that requirement capture, analysis, design, property description and verification tasks can seamlessly be accomplished in the same language, namely UML, and in the same environment [ASS09]. The designer is merely required to have minor UML skills and does not need to familiarize with formal languages like CTL or UPPAAL.

The paper is organized as follows. Section 2 surveys papers on property expression languages and explains why we do not reuse other UML / SysML diagrams such as state machines. Section 3 introduces the TEPE language. Section 4 presents the integration of the TEPE language into the AVATAR real-time profile in terms of methodology and language. Section 5 addresses the toolkit issue. Section 6 discusses an example: an elevator system. At last, Section 7 concludes the paper and outlines future work.

2 Related work

2.1 Property specification

There are several widely accepted and standardized verification oriented languages which bear some resemblance with our approach as far as support for sequential behavior is concerned. These languages mostly target the verification of HDL designs. System Verilog [AOI] provides concurrent assertions for describing behavior that spans over time. The underlying event model is based on clock ticks. However AVATAR temporal operators, either for system design or property verification, are tied to physical time, and so to state machine temporal operators.
The e-language [VDI02] somewhat extends the System Verilog event model by introducing user defined events derived from behavior or other events. However, temporal expressions require a trigger events to be selected for condition evaluation. Our approach offers more flexibility for operators may specify several sampling events or signals respectively. Furthermore, in AVATAR, the set of sampling events may evolve over time.
PSL [AOI04] can be considered as an extension of LTL and

[1] TEmporal Property Expression Language
[2] Automated Verification of reAl Time softwARe

CTL temporal logics and the expressiveness of its temporal layer resembles the System Verilog specification language. PSL is also tightly coupled to clock based events. So called "properties" are used to describe behavior over time and they are made up of a Boolean expression and a clock expression amongst others. However, the aforementioned languages fail to model physical time independently of clock cycles.
The SystemC Verification Standard [otSVWG03] addresses the creation of test benches and allows both for random stimulus generation and recording of resulting transactions. To our knowledge, it does not comprise a syntax for expressing temporal properties, nor automated ways to verify them.
[Smi01] advocates a nice graphical notation which aims to simplify the formalization of requirements for model checking. System executions are expressed in the form of timeline diagrams discriminating optional, mandatory, fail events and related constraints. As for other trace based approaches, conditional or varying system behavior cannot easily be expressed. Moreover, the approach does not address real-time or performance requirements.

2.2 Property specification in UML

The MARTE profile embraces VSL [OMG08] which aims at specifying the values of constraints, properties and stereotype attributes particularly related to non-functional aspects. Even when used in combination with sequence diagrams, VSL makes it cumbersome if not impossible to specify complex sets of sequential behaviors.
The Rhapsody tool used by [dSV09] similarly enables formal verification of SysML diagrams using UPPAAL. Unlike TTool, Rhapsody does not distinguish between requirements and properties. Nor it supports a property expression language - such as TEPE - and computation operators in state machines. In terms of user-friendliness, TTool allows one to right-click on an action symbol and automatically verify the reachability of that action. In the same situation, the user of Rhapsody is obliged to enter a logic formula, which assumes some knowledge in logic.The OMEGA2 environment [OD10] has also strong connections with Rhapsody for it implements the same semantics. OMEGA2 supports requirement diagrams as defined in SysML. Conversely ARTISAN [HH10] extends SysML to cope with continuous flows. ARTISAN models may contain probabilities and interruptible regions, two concepts not yet supported by AVATAR. The open-source environment Topcased also enables requirement modeling in a SysML fashion [AM10].
Electronic System Level (ESL), which is an emerging electronic design methodology, has stimulated research work on joint use of SysML and formal languages supported by simulation tools. Several papers discuss solutions where a model is designed in SysML and translated into VHDL-AMS [SCo] or Simulink [VD06]. Mechanical engineering is another area where SysML is combined with already existing domain specific languages, such as Modelica or bond graphs.

2.3 Property specification with TEPE

Finally, TEPE matches a high abstraction level in contrast to languages closely tied to static sampling events, especially clock cycles [AOI][VDI02][AOI04]. The language supports reasoning in terms of high level signals, timing and the value of system variables (equations). As the objective is to verify sequential behaviors - and their timing -, the property descriptions could surely rely on state machines. However, overusing UML Statecharts both for modeling and property purposes is probably not a good idea. Indeed, if property description does not rely on a different formalism, it runs the risk of being hampered by the same errors in reasoning as the model. Moreover, (1) Statecharts are not adequate to model situations where events may be received in any order, which are commonly encountered in properties, and (2), statecharts do not put an emphasis on property relations, like TEPE. Apart from Statecharts, formally defined descriptions for sequential behavior fall short in UML. For example, scenario-based models like Sequence Diagrams fail to describe relations between attributes of various instances (e.g., attribute x of instance *I0* is equal to attribute y of instance *I1*), and they might be inadequate for describing complex situations, in particular to reference past events. Even though Live Sequence Charts [DH01] provide more semantics to scenarios, modeling several acceptable traces is still cumbersome. Additionally, the integration of equations that have to be fulfilled as a function of the system behavior is not straightforward in UML and requires the usage of OCL, thereby circumventing the graphical notation.

3 TEPE: TEmporal Property Expression language

SysML Requirement Diagrams (RDs) structure *requirements* and define *testcases*. Basically, requirements may be linked together using $<<$ *derive* $>>$ and composition relations. Requirements may also be copied from other views ($<<$ *copy* $>>$). SysML RDs also support the definition of testcases (that we rename "*properties*") that may be linked to requirements using the $<<$ *verify* $>>$ relation. Unfortunately, properties are only defined in an informal way with an identifier and a text. To address that limitation, this chapter introduces TEPE.

3.1 TEPE and Parametric Diagrams

A specification in TEPE represents functional and non-functional properties in a formal way, using Parametric Diagrams. As opposed to informal SysML PDs, TEPE PDs are amenable to automated verification. A small set of operators can be leveraged to make up complex properties.
In TEPE, each property is expressed as a graph of *Signals*, *Attributes*, *Constraints* (*Equations*, *Logical Constraints*, *Temporal Constraints*) and *Properties*. An excerpt from the meta model of TEPE PDs is depicted in Figure 1. All

stereotypes of PDs are derived from their respective SysML counterpart: *Blocks*, *Operators*, and *Links* interconnecting *Operators*. A block defines all *Attributes* and *Signals* which are referred to by *Operators*. Operators are assembled by means of *Links* which are attached to the *Operator's* ports. *Links* are characterized by the respective type of the data they convey: Attribute, Signal and Property. *Ports* must obviously have the same data type as the connected *Link*, and two connected ports must have an opposite type (input, output).

A TEPE PD is supposed to be constructed in the following way:

1. First, *Blocks* are represented with their particular *Attributes* and *Signals* subject to verification. These entities have been identified during the design phase.

2. Values derived from original attributes and signals are introduced (cf. *Equation* and *Alias* operators).

3. The reasoning about the sequential and temporal behavior of the system is expressed in terms of logical and temporal operators connected to *Signals* and *Properties*. These logical and temporal operators can be cascaded.

4. Several *Properties* may be merged using logical property operators (*Conjunction*, *Disjunction*, *Property* Definition Operators).

5. Finally the formal property is labeled to link it to an informal SysML RD and to determine whether (non-) liveness or (non-) reachability should be verified on that property.

6. To avoid overloaded diagrams, constituting properties of a requirement can be spread over several diagrams.

The purpose of the following example (see Figure 2) is to informally present operators of PD. The PD defines two Blocks. *BlockA* has two attributes x and y as well as two signals *s1* and *s2*. *BlockB* declares one signal called *s3*. A *Setting* operator declares a temporary variable which serves as a shorthand to simplify expressions. An equation imposing a constraint on the variable z is introduced as well. An *Alias* operator denotes the logical disjunction of signals, the resulting signal is thus raised upon occurrence of one of the two entry signals. Two properties are logically combined using an *AND* operator. The first one states that upon reception of an *s2* signal, the compound signal resulting from the *Alias* operator must be observed as well, i.e. *s2* or *s3*. Furthermore, if the *s1* signal is received or the equation evaluates to false between the occurrence of *s2* and the compound signal, the LS operator evaluates to false. The second property requires the signal *s2* to be sent *less than 10 time units after* signal *s1*. The overall property is checked for liveness, which is made explicit by a *Property Definition* operator.

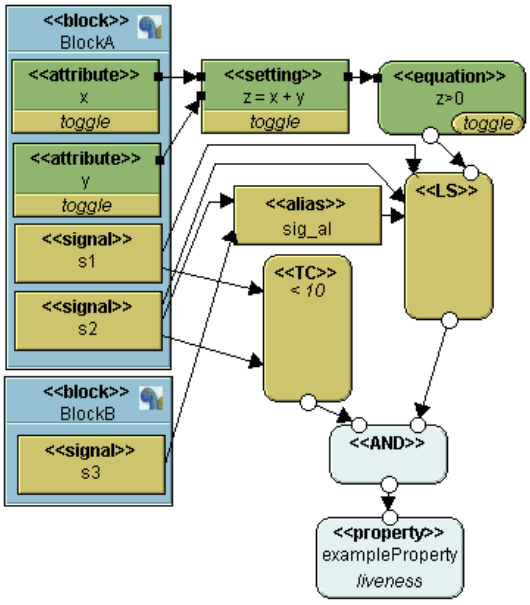


Figure 2: Example of an TEPE Parametric Diagram

3.2 TEPE: operators

TEPE operators manipulate three kinds of data: *attributes*, *signals* and *properties*.

- **Attributes**: defined in blocks at system design level, or as new attributes from existing ones (*Setting*).
- **Signals**: directly defined in blocks. Two additional signals are also considered: $entry(state)$ and $exit(state)$.
- **Properties** are Boolean values resulting from SysML constraints: either *Equations*, or temporal / logical constraint operators.

3.2.1 Attribute-based operators

Two operators define attributes: *attribute declaration* and *attribute setting*. The *Equation* operator takes attributes as input, and outputs a property. Moreover, *attribute operators* output a signal indicating a value change (*toggle*).

3.2.2 Signal-based operators

Alias operators merge several distinct *Signals* to one. The resulting *Signal* is notified upon notification of one of the constituting *Signals*.
SigToPropOperators introduce a partial order of transitions and a notion of time and can thus be used to limit the temporal scope of *Properties*. *SigToPropOperators* thus translate temporal behavior of *Signals* into a *Property* which can be further evaluated. Three *SigToPropOperators* are defined: the temporal constraint, the partial order, and the logical sequence.

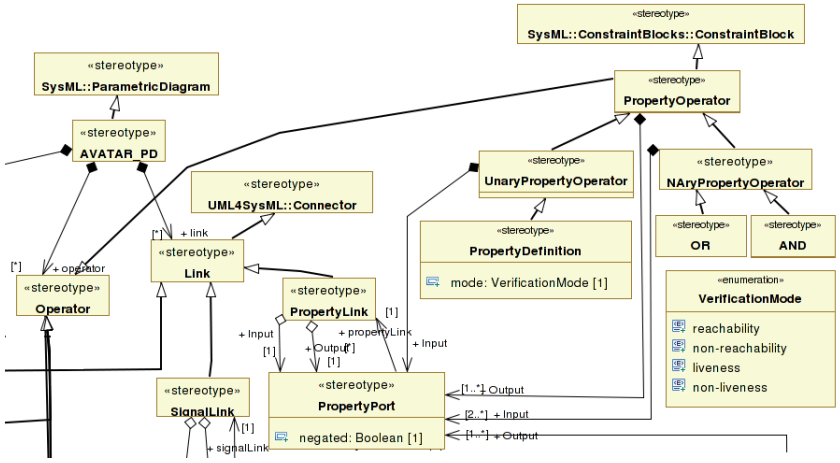


Figure 1: Excerpt from the TEPE PD Meta Model

3.2.3 Property-based operators

Property Operators comprise conjunction and disjunction functions for *Properties*.
Property Definition Operators assign a name to a property, and specify its verification kind: (non-) reachability or (non-) liveness. This verification kind is similar to CTL quantifiers.

3.3 TEPE: signal-based operators

Two operators are of outstanding importance in TEPE: *Logical Constraints* and *Temporal Constraints*. They both observe signals and properties after a given signal condition is met, and output another property based on that observation.

3.3.1 Logical Constraint

Inputs: set S of n signals $s_1 \dots s_n$, set S_f of m signals $s_{f1} \dots s_{fm}$, where $S_f \cap S = \emptyset$ and a property P_i (optional), *Output*: P_o
The operator defines a set of transitions which may be reached irrespective of their order. Once any signal s_{first} in S is encountered, the operator requires all signals $S \backslash \{s_{first}\}$ to be observed for P_o to be true. If none of the signals S is ever received, P_o is defined to be true. Furthermore, the operator handles failure signals $s_{f1} \dots s_{fm}$ forcing P_o to be false in case they are notified between the first received signal of S and the last one. In addition to that, P_i is required be true during all that period, otherwise P_o is set to false. A more formal description of the operator applied to two signals s_1, s_2 and failure signal s_f is given in Figure 3, where T stands for TRUE, F for FALSE and $CF(P_i)$ denotes the change of property P_i from true to false. Sending and reception of a message are symbolized by exclamation and question marks respectively. The value of P_o is indicated on the state symbol, in the Moore machine style.

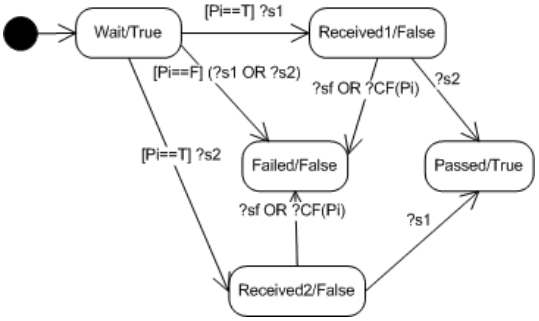


Figure 3: Semantics of Logical Constraints

3.3.2 Logical Sequence

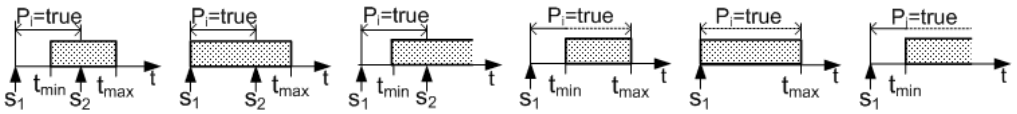
Inputs: set S of n signals $s_1 \dots s_n$, the set S_f of m signals $s_{f1} \dots s_{fm}$, where $S_f \cap S = \emptyset$ and a property P_i (optional), *Output*: P_o
This operator represents a property of a system defined in terms of a logical sequence of state transitions. It establishes an order among a given set of signals $s_1 \dots s_n$, that is, it works similarly to the Logical Constraint, apart from the fact that the order in which input signals are received is imposed.

3.3.3 Temporal constraint

Inputs: two signals s_1, s_2 (the latter is optional), two time values t_{min}, t_{max} (either of the two is optional) and a property P_i (optional, considered to be true by default), *Output*: P_o
Depending on the provided arguments, P_o is defined to be true under the following conditions:

1. $s_1, s_2, t_{min}, t_{max}$: s_2 has to occur at least t_{min}, at most t_{max} after s_1 and P_i must be true from the reception of s_1 to the reception of s_2 (Figure 4a)

2. s_1, s_2, t_{max}: s_2 has to occur at most t_{max} after s_1 and P_i must be true from the reception of s_1 to the reception of s_2 (Figure 4b)



(a) Semantics 1 (b) Semantics 2 (c) Semantics 3 (d) Semantics 4 (e) Semantics 5 (f) Semantics 6

Figure 4: Temporal Constraint Operator Semantics

3. s_1, s_2, t_{min}: s_2 has to be notified at least t_{min} after s_1 and P_i must be true from the reception of s_1 to the reception of s_2 (Figure 4c)

4. s_1, t_{min}, t_{max}: after reception of s_1, P_i must be true for at least t_{min} and at most t_{max} (Figure 4d)

5. s_1, t_{max}: after reception of s_1, P_i must be true for at most t_{max} (Figure 4e)

6. s_1, t_{min}: after reception of s_1, P_i must be true for at least t_{min} (Figure 4f)

4 Integrating TEPE into a UML profile for real-time systems

4.1 The basics of AVATAR

The AVATAR profile reuses eight of the SysML diagrams (Package diagrams are not supported). It further structures Sequence Diagrams using an Interaction Overview Diagram (a diagram supported by UML2, not by SysML). The AVATAR profile is syntactically and semantically defined by a meta-model. Besides a syntax, a semantics and a tool support, a profile is also characterized by a methodology.

4.2 Methodology

The AVATAR methodology comprises the following stages:

1. **Requirement capture**. Requirements and properties are structured using AVATAR Requirement Diagrams. At this step, properties are just defined with a specific label.

2. **System analysis**. A system may be analyzed using usual UML diagrams, such as Use Case Diagrams, Interaction Overview Diagrams and Sequence Diagrams. This stage is not covered in this paper.

3. **System design**. The system is designed in terms of communicating SysML blocks described in an AVATAR Block Diagram, and in terms of behaviors described with AVATAR State Machines.

4. **Property modeling**. The formal semantics of properties is defined within TEPE Parametric Diagrams (PDs). Since TEPE PDs involve elements defined in system design (e.g, a given integer attribute of a block), TEPE PDs may be defined only after a first system design has been performed.

5. **Formal verification** can finally be conducted over the system design, and for each testcase.

Once all properties are proved to hold, requirements, system analysis and design, as well as properties may be further refined. Thereafter, and similarly to most UML profiles for embedded systems, the AVATAR methodological stages are reiterated. Having reached a certain level of detail, refined models may not be amenable to formal verification any more. Therefore the generation of prototyping code may become the only realistic option.

4.3 AVATAR: Block and State Machine Diagrams

Apart from their formal semantics, AVATAR Block and State Machine Diagrams only have a few characteristics which differ from the SysML ones.
An AVATAR block defines a list of attributes, methods and signals. Signals can be sent over synchronous or asynchronous channels. Channels are defined using connectors between ports. Those connectors contain a list of signal associations.
A block defining a data structure merely contains attributes. On the contrary, a block defined to model a sub-behavior of the system must define an AVATAR State Machine.
AVATAR State Machine Diagrams are built upon SysML State Machines, including hierarchical states. AVATAR State Machines further enhance the SysML ones with temporal operators:

- **Delay:** $after(t_{min}, t_{max})$. It models a variable delay during which the activity of the block is suspended, waiting for a delay between t_{min} and t_{max} to expire.

- **Complexity:** $computeFor(t_{min}, t_{max})$. It models a time during which the activity of the block actively executes instructions, before transiting to the next state: that computation may last from t_{min} to t_{max} units of time.

The combination of complexity operators ($computeFor()$), delay operators, as well as the support of hierarchical states - and the possibility to suspend an ongoing activity of a substate - endows AVATAR with main features for supporting real-time system schedulability analysis.

4.4 Translation to UPPAAL: the basics

The translation of a an AVATAR-TEPE specification to UPPAAL is defined as the following tr function:

$tr : BD \times SMDs \times PDS \mapsto UPPAALSpec$
More precisely, *tr* takes as input one AVATAR Block Diagram, a set of State Machine Diagrams, and a set of Parametric Diagrams. *tr* returns a UPPAAL specification. A UPPAAL specification is made upon a set of timed automata communicating using synchronized channels.
Basically, one block and its state machine are transformed into one automata. Each time two blocks can communicate, a channel is created between the two corresponding automata. AVATAR State Machine operators are transformed into a set of transitions between automata states. In particular, the use of AVATAR delay and complexity operators can be translated using UPPAAL clock initializations, state invariants, and guards on clocks.
For each property defined in a Parametric Diagram, a corresponding observer automata [FSsA08] is derived. The latter makes states and transitions related to verification explicit in the UPPAAL model. In so doing, proving the satisfiability of a given TEPE property is reduced to searching for the accessibility or liveness of a given observer state, using the UPPAAL verifier.

5 Toolkit

5.1 TTool

The open-source toolkit *TTool* [ASS09] supports several UML / SysML profiles, in particular TURTLE [ACLdSS04] and DIPLODOCUS [Apv08]. TTool offers UML modeling edition facilities, and well as press-button approaches for formal verification and simulation. TTool and its profiles are supported by several academic and industrial partnerships.
TTool is interfaced to verification tools that implement reachability analysis and model-checking. For example, to decide whether some UML action is reachable or not, it suffices to right click on the corresponding action's symbol: The UPPAAL verifier is invoked with corresponding CTL formula, and the result is displayed on UML diagrams.
TTool encourages the user to use viewpoints simply by selecting the blocks to be considered for model transformation. If a property refers to excluded entities, it is simply ignored during verification as its evaluation is impossible. Alternatively, the property could be considered to hold or to be violated by default.
Moreover, a very fast simulation engine has been developed for DIPLODOCUS [KAP09], and integrated into TTool. It features the animation and interactive simulation of UML diagrams [KAP10].

5.2 Extending TTool for TEPE and AVATAR

TTool can now edit TEPE diagrams. TTool also partially supports the *AVATAR-TEPE to UPPAAL* translation. Currently, AVATAR Block and State Machine diagrams can always be translated to UPPAAL (no limitation), and AVATAR properties expressed in AVATAR Parametric Diagrams can automatically be formally checked out only when they target the reachability or liveness of one specific state of an AVATAR State Machine. Otherwise, they must be translated by hand. Their full translation is under development.

6 Case study

6.1 Requirements

As a case study, we consider an elevator system. Four functional safety-related requirements have been identified and modeled in a Requirement Diagram:

- **Req1**: The door does not open when the elevator is moving.
- **Req2**: The elevator does not depart with an open door.
- **Req3**: The operational profile requires the elevator to accelerate after being set in motion and to decelerate before stopping.
- **Req4**: Deceleration must be accomplished between 1 and 5 seconds before the selected floor is reached.

6.2 System design

The block diagram (See Figure 5) comprises three main elements: The *ElevatorControl* block, which is charge of controlling the cabin of the elevator, the elevator door and the shaft doors. Three actuators are also represented as blocks: *ElevatorCabin*, *ElevatorDoor* and *ShaftDoors*. Furthermore, another block stands for actions taken by the user of the system. As previously stated, blocks are interconnected with signals. For instance, the *ElevatorControl* unit may send a signal to its environment. By explicitly connecting it to a corresponding signal defined within *ElevatorCabin*, the two finite state machine are able to synchronize.

6.3 Property modeling in TEPE

After having structured the system in terms of blocks, attributes and signals, the developer may proceed with the formal model of the properties to be verified (see Figure 6), corresponding to requirements (*Req1* to *Req4*). More precisely, one property corresponds to the four requirements. In Figure 6, *moveElevator* is declared as composite signal of *ascendCabin* and *descendCabin*; *moveElevator* is thus raised upon notification of one of the constituting signals. *Req1* is captured by a Logical Sequence operator receiving as input the sequence of the composite signal and the *stopCabin* signal. During that sequence, the reception of an *elevatorDoorOpen* signal is considered as an incorrect system behavior. A temporal operator is dedicated to *Req2*: At the instant when the cabin is set in motion (notifed by the composite signal), the system variable *doorOpen* must evaluate to false. To satisfy the operational profile requirements *Req3*

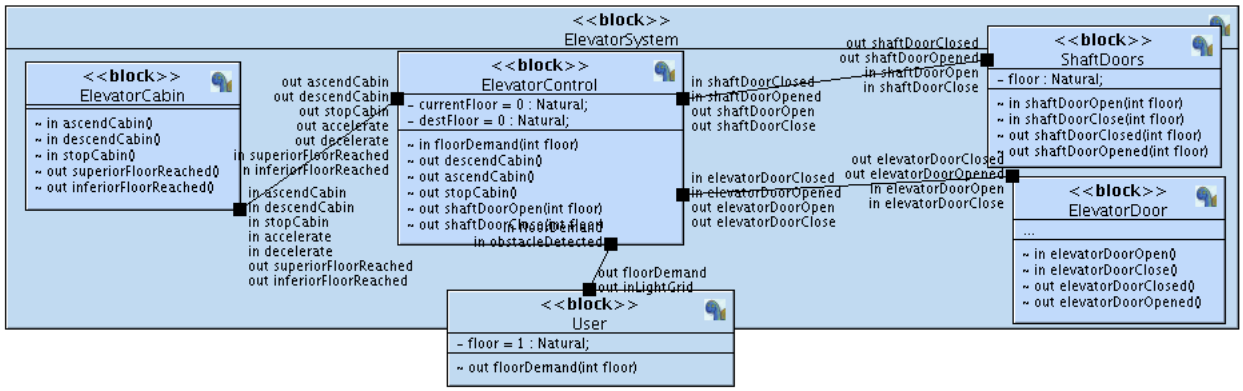


Figure 5: Elevator Block Diagram

and *Req4*, a sequence operator monitors the *accelerateCabin* and the *decelerateCabin* signals. The cascading of the latter sequence operator with the one dedicated to cabin motion suggests that the sequence of *accelerateCabin* and *decelerateCabin* must occur when the cabin is in motion (*Req3*). A second temporal Operator accounts for *Req4*: at least 1, at most 5 time units have to elapse between notification of the signals *decelerateCabin* and *stopCabin*. The results of all requirements are finally combined using an AND operator.

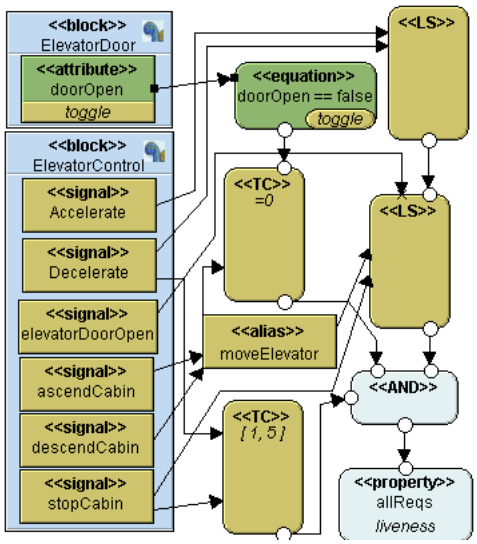


Figure 6: TEPE Model of the elevator's requirements

6.4 Discussion

The case study demonstrates the applicability of TEPE language for the verification of system properties. While sharing the most important semantics with other temporal logics like CTL, TEPE enriches their expressiveness with the notion of physical time and an operator matching a set of unordered signals. The granularity and the abstraction level of diagrams is in line with the system model; system transitions are referred to using signals and state variable modifications. These elements are familiar to the designer as he/she introduced them during the design phase. By combining static equations and sequential operators, a temporal scope is attached to the former. In our example, initial SysML informal requirements are easily translated into TEPE. A formal definition however opens the door for an automatic verification on the fly during simulation or by transformation into an UPPAAL model enhanced with observers. Although nothing prevents from using the textual from of TEPE, the graphical representation based on Parametric Diagrams far outreaches the latter in terms of readability. Moreover, an adequate coloring of operators facilitates the clear distinction between timed (signals) and untimed parts (properties) of the diagram.

7 Conclusion

The TEmporal Property Expression language, or TEPE for short, customizes SysML parametric diagrams. Properties are built up upon logical and temporal relations between block attributes and signals.
As an OMG-SysML compliant language, TEPE may be integrated to a broad variety of SysML real-time profiles, such as AVATAR. AVATAR is a verification-centric profile that improves SysML's capability to express and verify properties of time-constrained systems. Unlike other real-time profiles, AVATAR-TEPE puts the emphasis on requirement and property modeling. AVATAR further reuses SysML block diagrams and state machines so as to distinguish between waiting time and computation time. AVATAR state machines also support nested states, as well as suspension. A complete suspend/resume mechanism is currently under investigation, both in terms of language and formal verification.
Moreover, AVATAassociated with a verification centric method supported by the open-source toolkit TTool. The toolkit includes a diagram editor, a UPPAAL code generator and a press-button interface to formal verification. TTool thus enables formal verification of SysML design diagrams against temporal properties expressed in TEPE. In particular, the SysML model of the elevator system discussed in the

paper has been developed using TTool.
Short term extension shall include automatic generation of observers from TEPE properties. We also plan to introduce a methodological assistant to guide newcomers to AVATAR and to make TTool as friendly as possible for education activities.

References

[ACLdSS04] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE: A real-time UML profile supported by a formal validation toolkit. In *IEEE transactions on Software Engineering*, volume 30, pages 473–487, Jul 2004.

[AM10] M. Audrain and B. Marconato. Topcased 3.4 tutorial - requirement management. In *http://www.topcased.org/index.php? documentsSynthesis=y&Itemid=59*, 2010.

[AOI] Accellera Organization Inc. SystemVerilog 3.1a Language Reference Manual, www.systemverilog.org.

[AOI04] Accellera Organization Inc. Property specification language, reference manual, version 1.1. 2004.

[Apv08] L. Apvrille. TTool for DIPLODOCUS: An Environment for Design Space Exploration. In *Proceedings of the 8th Annual International Conference on New Technologies of Distributed Systems (NOTERE'2008)*, Lyon, France, June 2008.

[ASS09] Ludovic Apvrille and Pierre De Saqui-Sannes. Making formal verification amenable to real-time UML practitioners. In *Proceedings of the 12th European Workshop on Dependable Computing*, Toulouse, France, May 2009.

[DH01] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[dSV09] E. C. da Silva and E. Villani. Integrando SysML e model checking para v&v de software crítico espacial. In *Brasilian Symposium on Aeropspace Engineering and Applications, São José dos Campos, SP, Brasil*, September 2009.

[FSsA08] B. Fontan, P. De Saqui-sannes, and L. Apvrille. Timing requirement description diagrams for real-time system verification. In *ERTSS - Embedded Real Time Software and Systems*, Jan 2008.

[HH10] M. Hause and J. Holt. Testing solutions with UML/SysML. In *http://www.artist-embedded.org/docs/Events/2010/UML_AADL /slides/Session1_Matthew_Hause.pdf*, 2010.

[KAP09] Daniel Knorreck, Ludovic Apvrille, and Renaud Pacalet. Fast simulation techniques for design space exploration. In *Objects, Components, Models and Patterns*, volume 33 of *Lecture Notes in Business Information Processing*, pages 308–327. Springer Berlin Heidelberg, 2009.

[KAP10] Daniel Knorreck, Ludovic Apvrille, and Renaud Pacalet. An interactive system level simulation environment for Systems on Chip. In *ERTSS - Embedded Real Time Software and Systems*, May 2010.

[OD10] I. Ober and I. Dragomir. Omega2: Profiles and tools for system modeling and verification with UML2.x and SysML, UML & AADL. In *http://www.artist-embedded.org/docs/Events/2010/UML_AADL /slides/Session4_Iulian_Ober.pdf*, 2010.

[OMG08] OMG. A UML profile for MARTE, beta 2, www.omg.org. 2008.

[otSVWG03] Members of the SystemC Verification Working Group. SystemC Verification Standard Specification Version 1.0e, www.systemc.org. 2003.

[SCo] SysML companion. In *http://www.realtimeatwork.com/?page_id=683*.

[Smi01] Margaret H. Smith. Events and constraints: a graphical editor for capturing logic properties of programs. In *In Proceedings of the 5th International Symposium on Requirements Engineering*, pages 14–22, 2001.

[VD06] Yves Vanderperren and Wim Dehaene. From UML/SysML to matlab/simulink: current state and future perspectives. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 93–93, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[VDI02] Verisity Design Inc. e Language Reference Manual, www.ieee1647.org/downloads/prelim_e_lrm.pdf. 2002.