Assisting Temporal Requirement Specification

Ahmed Mekki1,2, Mohamed Ghazel1,2 and Armand Toguyéni1,3
1. Univ Lille Nord de France, Lille F-59000, France
2. IFSTTAR, ESTAS, Villeneuve d’Ascq F-59650, France
3. EC LILLE, LAGIS, Villeneuve d’Ascq F-59650, France

Received: August 17, 2011 / Accepted: September 29, 2011 / Published: January 25, 2012.

Abstract: The aim of the present work is to introduce a pattern-based method for assisting the user during the temporal Requirement Specification (RS) phase. Indeed, since the user usually has to handle abstract notation as well as mathematical-based languages within this phase, RS is becoming more and more tedious and error-prone especially when dealing with complex systems. The authors’ method begins by defining a new typology while taking into account all the common temporal requirements one may meet when specifying a system. Then, a literal word-based formal grammar able to express all the types of the identified requirements has been developed. Actually, the goal is to assist the user during the requirement identification with some means that are simple, intuitive, albeit rigorous. Finally, a generic set of observation patterns relative to the new time constraint taxonomy is defined. In practice, to check a given temporal constraint, its relative observation pattern is instantiated to obtain an observer that will stand for a watchdog for the associated requirement on the system.

Key words: Temporal requirements, system specification, verification & validation, observation patterns.

1. Introduction

Since it constitutes a baseline for validation and verification, Requirement Specification (RS) is one of the most capital research topics in critical systems engineering. Consequently, its implication on the safety and the correctness of critical systems (e.g., nuclear plants, medical devices, transportation systems) is generally undeniable, since such systems must achieve a high level of robustness and reliability.

On the other hand, critical systems usually involve time-dependent functionalities, and therefore, tools and techniques for behavior design and verification (especially temporal requirements) are increasingly important and strongly recommended. The notations most used are Temporal Logic and Timed Automata (TA) [1]. The former is generally used for expressing requirements, while the later is used for specifying timed systems. Both of them, temporal logic and TA, are well-suited for expressing timed behavior and for modeling real-time components. This is proven by the number of developed automatic verification tools, like Uppaal [2] and Kronos [3] model-checkers). Moreover, these tools have proven to be efficient in both academic and industrial applications. Nevertheless, specifying time-constraint properties is becoming a more and more difficult task, due to the widespread applications and increasing complexity of checked systems. Furthermore, the specification process is generally tedious and error-prone, since it requires manipulating abstract notations, and thus, demands sophisticated knowledge in temporal logic, as well as mathematical skills.

The aim of the present work is to guide the user during the temporal requirement specification phase. Indeed, the cost of errors due to requirement specification identified during the system verification and validation is very important. For instance, a study presented by many different companies and
organizations over several years shows the relative cost of requirement error correction over the software life-cycle (cf. Fig. 1). Indeed, the cost of fixing errors increases significantly as one proceeds in the system/software life-cycle. For instance, in the case where it is not identified until the operations stage, an error cost from $40 to $1,000 to fix it. Errors are generally due to ambiguity, inaccuracy or inconsistency during the specification. To deal with this, Gerrit Muller proposed in Ref. [5] the criteria of a right RS:

- It should reflect the real needs of all stakeholders in a simple, implicit and latent way;
- It should describe a feasible product;
- It must answer most critical design questions;
- It should be useful for human product creators.

Consequently, assisting the user with some guiding means while specifying the temporal requirements, will considerably reduce the number of errors, and therefore, help the user to improve the system’s quality and reduce the system’s cost.

As mentioned previously in this paper, we focus on the specification stage of temporal requirements. In fact, the idea is to propose assisting means that are easy to manipulate, and at the same time, accurate and formal. Nevertheless, the more accurate, rigorous and generic a notation/language is, the more abstract and difficult to handle and understand it becomes. Therefore, one of the challenges that we have faced while dealing with this work was to look at both intuition/simplicity and rigor/accuracy. To deal with this, first, we propose a taxonomy that identifies all the common temporal constraints that one may meet within time-constrained systems. Then, based on this taxonomy, we have developed a Structured English Grammar (denoted SEG hereafter) able to cover all temporal property types. Indeed, SEG is a set of literal compoundable elements with a precise and rigorous interpretation. In this way, formal aspects—met when dealing with requirement specification—are hidden to the user, since he will only handle literal terms to make assertions expressing the requirements. Once introduced, the temporal requirements classification as well as the SEG will be assigned to an observation pattern repository. Indeed, we will assign to each requirement type an observation pattern (observer) in order to check the validation/violation of the requirement. In other words, the observer will act as watchdog to the associated requirement. Therefore, each generated sentence from SEG identifies a unique requirement type and, consequently, identifies a unique observer and vice-versa.

The paper is organized as follows: In section 2, an overview of the context and related works is given; in section 3, we discuss the classification of temporal requirements we have established; the developed formal grammar for specifying temporal requirements is introduced in section 4; section 5 presents an overview of a validation method that we are developing, before concluding and suggesting some future works in section 6.

2. Context and Related Work

Several studies dealing with the analysis and classification of temporal requirements have been undertaken. The most referenced classifications are the followings:

- The Allen classification [6] is one of the most referenced classifications in the artificial intelligence field. Allen proposes a process-based taxonomy where he defines 14 possible relations between processes,
such as meets, overlaps, starts, finishes, etc. Nevertheless, event-based requirements, as well as quantitative properties, could not be expressed using this classification;

- The Dwyer classification [7] is a taxonomy of time-dependent constraints. Dwyer formally expresses and defines all the identified constraint using Computation Tree Logic (CTL), Linear Temporal Logic (LTL), Quantified Regular Expressions (QRE) and Graphical Interval Logic (GIL) temporal logic formulas. Furthermore, he introduces the “scope” notion in the requirement expression. The scope is used to express the applicability context of the constraints. Dwyer introduced five scopes: Globally, Before an event/state occurs, After an event/state occurs, Between two events/states, and finally, Until an event/state. Nevertheless, despite the fact that this classification deals with states and events, it only deals with qualitative temporal requirements. In fact, one cannot express either quantitative constraints or punctuality properties using this classification;

- The Konrad classification [8] is a real-time extension to the Dwyer classification. This classification uses Metric Temporal Logic (MTL), Timed Computational Tree Logic (TCTL), and Real-Time Graphical Interval Logic (RTGIL) to formally express quantitative requirements. Like Dwyer classification, the Konrad classification uses the five scopes mentioned above. Nevertheless, his classification still suffers from the absence of the punctuality constraint, as well as the absence of interval requirements;

- The Sadani classification [9] is an interval-based taxonomy. This classification is very suitable for expressing requirements on process. Nevertheless, despite the fact that the punctuality requirement is considered, requirements on events cannot be specified using this classification, since it is interval-logic-based.

3. Temporal Requirement Taxonomy

Here, we establish a new classification with several differences in comparison with the classifications mentioned in section 2. The first difference is in the way requirements are organized and the second is the numerous types of requirements which we can express and which the other classifications did not integrate. In fact, requirements are split into response, precedence and absolute presence subsets. Moreover, our classification refers only to events (factual or fictive), which improve the coherence and the clarity of requirements. In practice, for requirements dealing with states/processes, instead of directly considering these states/processes, we express the requirements using two events: The first event represents the activation of the state/process and the second the deactivation. In this way, we obtain a repository of event-based requirements. The definition of each requirement type is given in Table 1.

As shown in the Dwyer classification, requirements could be monitored permanently or restricted to a special context. The scope of a given requirement defines the applicability interval in which it has to be applied. Here we consider 4 scopes, namely:

- Globally: The requirement must be satisfied all the time and this is the scope by default;
- After a Reference: The requirement must be satisfied all the time after a given reference;
- Before a Reference: The requirement must be satisfied all the time before a given reference;
- Between two References: The requirement must be satisfied all the time after a given begin-reference and before an end-reference.

4. Structured English Grammar

4.1 Idea

Nowadays, critical systems are usually widespread and put together components that come from different manufacturers. Consequently, this rapidly increases the complexity of specification, as well as, the complexity of verification/validation of such systems. In other words, RS of critical systems is becoming challenging for system designers. Furthermore, despite the higher
Table 1  The defined temporal requirement taxonomy.

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response presence between</td>
<td>R ensures that an event (E_{mon}) must occur within a temporal interval [t_{begin}; t_{end}] from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Response min delay</td>
<td>R ensures that an event (E_{mon}) must occur after a minimum delay T_{min} time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Response max delay</td>
<td>R ensures that an event (E_{mon}) must occur before a maximum delay T_{max} time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Response punctuality delay</td>
<td>R ensures that an event (E_{mon}) must occur exactly at a delay T time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Reaction</td>
<td>R ensures that an event (E_{mon}) must occur after “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Response interdiction between</td>
<td>R ensures that an event (E_{mon}) cannot occur within a temporal interval [t_{begin}; t_{end}] from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden response from a delay</td>
<td>R ensures that an event (E_{mon}) cannot occur after a minimum delay T_{min} time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden response before a delay</td>
<td>R ensures that an event (E_{mon}) cannot occur before a maximum delay T_{max} time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden response at date</td>
<td>R ensures that an event (E_{mon}) cannot occur exactly at a delay T time unit from “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden response</td>
<td>R ensures that an event (E_{mon}) cannot occur after “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence presence between</td>
<td>R ensures that an event (E_{mon}) must occur within a temporal interval [t_{begin}; t_{end}] before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence min delay</td>
<td>R ensures that an event (E_{mon}) must occur after a minimum delay T_{min} time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence max delay</td>
<td>R ensures that an event (E_{mon}) must occur before a maximum delay T_{max} time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence punctuality delay</td>
<td>R ensures that an event (E_{mon}) must occur exactly at a delay T time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence</td>
<td>R ensures that an event (E_{mon}) must occur before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Precedence interdiction Between</td>
<td>R ensures that an event (E_{mon}) cannot occur within a temporal interval [t_{begin}; t_{end}] before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden precedence from a delay</td>
<td>R ensures that an event (E_{mon}) cannot occur after a minimum delay T_{min} time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden precedence before a delay</td>
<td>R ensures that an event (E_{mon}) cannot occur before a maximum delay T_{max} time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Forbidden precedence at date</td>
<td>R ensures that an event (E_{mon}) cannot occur exactly at a delay T time unit before “i” occurrence of event E_{Ref}</td>
</tr>
<tr>
<td>Absence</td>
<td>R ensures that an event (E_{mon}) cannot occur</td>
</tr>
<tr>
<td>Presence</td>
<td>R ensures that an event (E_{mon}) must occur</td>
</tr>
</tbody>
</table>

level of automation of verification tools (Model-checking [10]: Uppaal, Kronos, Spin…; theorem-proven: HOL, Isabelle, LP, PVS…), users often have to write logic formula to express system properties to be checked. Several Temporal Logics (LTL [11], MTL [12], CTL [13] and TCTL [14]) have been suggested in order to specify timing constraints. Although using a formal notation allows a rigorous specification that is well supported by tools, this remains tedious and error-prone, since formal notations require sophisticated logical and/or mathematical skills. Thus, our aim is to assist the user with some guiding means which are intuitive and simple, and at the same time, precise, rigorous, and expressive.

The idea is to propose a supporting approach that hides the formal foundation from the user. In fact, we elaborate a formal specification grammar that covers all the common requirements one may meet within critical systems. Then, based on this formal grammar, we will derive some compoundable literal-based elements which will be handled by the user. In addition, we will embed the grammar generation rules within the composition procedure. Consequently, the user will be guided in such a way as to obtain precise assertions.
while handling simple literal-based constructs.

4.2 BNF Notation

4.2.1 Definition

A Formal Grammar is a formalism allowing the definition of a precise syntax. In fact, a formal grammar determines the set, finite or infinite, of admissible strings on a given alphabet, i.e., the strings are combinations of alphabet elements which are acceptable in a precise domain. The notation most used for defining formal grammar is Backus-Naur Form (BNF) [15]. BNF is a formal notation that is usually used to describe formal languages. It provides a concise and accurate method for describing possible modes of combination of constituents. In other words, it shows exactly how to build the recursive data structures for syntax trees during the language implementation process.

4.2.2 BNF Foundation

Developed by John Backus and Peter Naur, BNF is a meta-syntax widely used to express context-free grammars. In fact, it is used to define the syntax of a programming language by using a set of rules, sometimes called productions. Each rule describes one possible way of constructing a constituent belonging to a syntactic category (non-terminal elements). In practice, we write the names of syntactic categories at the beginning of the rule within angle brackets (e.g., `<expression>`). On the other hand, the right side of the rule lists the components of the constituents. These components could be syntactic elements (terminal elements), other syntactic categories (non-terminal elements), or both. Moreover, the right side components are listed in order and without limit on the number of components. At the beginning of the rule, the non-terminal elements are separated from the right-hand side by the symbol ::=. Several rules could be combined into one by concatenating their right-hand sides if and only if they have the same non-terminal name. The combination is made using vertical bars to separate alternative constructions (e.g., `<R>` ::= `<A>` and `<R>` ::= `<B>  
⇒ `<R>` ::= `<A>` | `<B>` )

The vertical bar (|), angle brackets (<,>) and the ::= symbol are part of the meta-notation and are not part of the programming language that is being described. Hereafter, the BNF notation is expressed using BNF itself:

```
<syntax> ::= {<rule>};
<rule> ::= <identifier> "::=" <expression>;
<Expression> ::= <term> {"|" <term>};
<Term> ::= <factor> {<factor>};
<Factor> ::= <identifier> | <quotedSymbol> | "("<expression>")" | "["<expression>]"]" | 
"{"<expression>"}";
<Identifier> ::= letter {letter j digit};
<quotedSymbol> ::= "" anyCharacter "" | 
 anyCharacter "";
```

4.3 The Developed SEG

To facilitate the expression and the formalization of temporal properties, we propose a structured English grammar (SEG). This grammar supports both qualitative and quantitative properties. The aim of developing such a grammar is to provide the user with a simple means for expressing temporal requirements. The SEG is a guiding framework which offers a means of expression to the user. It also constrains him to make assertions according to a predefined syntax, hence making it possible to automatically recognize the expressed requirements. Concretely, each sentence generated by our grammar describes one temporal property and serves as a handler that aids expressing and understanding the requirement. SEG is expressed below using BNF (Backus-Naur Form) notation:

```
<Property> ::= <scope> "." <Specification>;
<Scope> ::= "global" | "before" <Entity> | "after" <Entity> | "between" <Entity> > "and" <Entity>;
<Specification> ::= <Entity> <Obligation> <OccurrenceType>;
<Entity> ::= <Event> | activate(<state>) | ...
deactivate(<state>);
<obligation> ::= “must” | “cannot”;
<occurrenceType> ::= (“occur”
[<referenceOccur>] | (“precede”
<referencePrecede>);
<referenceOccur> ::= ([<punctuality> | <fromAdelay> | <BeforeAdelay> | <Between>] “after”)
(<Repetition> | <LastOcc>);
<referencePrecede> ::= ([<punctuality> | <fromAdelay> | <BeforeAdelay> | <Between>]
“before”) <OccurrenceOfRef>;
<punctuality> ::= “exactly” <delay>;
<fromAdelay> ::= “after a minimum delay of”
<delay>;
<BeforeAdelay> ::= “before a maximum delay of”
<delay>;
<Between> ::= “between a minimum delay of”
<delay> “and a maximum delay of” <delay>;
<Repetition> ::= <OccurrenceOfRef> and
(“Regardless following occurrences”) | (“before the
next occurrence”);
<OccurrenceOfRef> ::= “i th occurrence of”
<entity>;
<LastOcc> ::= “the last occurrence of” <entity>;
<delay> ::= <integer> “time unit”;
<integer> ::= <digit>+;
<digit> ::= {0|1|2|3|4|5|6|7|8|9};
<i> ::= <number> | <digit> <i> | <i> <digit>;
<number> ::= {1|2|3|4|5|6|7|8|9};

An assertion obtained using the specification above is a composition of

- A scope: indicates the applicability context, or range of the requirement. In practice, it consists in an interval on which the requirement must be satisfied;
- An entity: represents the monitored entity, which could be an event, a state/process activation or a state/process deactivation;
- An obligation: expresses the obligation (must occur) or the prohibition (cannot occur). The analysis of the various case studies shows that all the temporal requirements can be expressed while combining must/cannot elementary assertions;
- An occurrence type: indicates the type of occurrence. Three types are used in our case; Response, Precedence and Absolute Occurrence.

Comparing our work to the works previously mentioned, we note
- As mentioned above, here we present a classification that deals only with events. In fact, unlike Dwyer and Konrad, and in order to ensure homogeneity between requirement declarations, the requirements dealing with states/processes are brought back to deal with events while considering 2 events: the activation and deactivation of the involved states/processes;
- Repetitive events are not considered in previous works. Here, we offer the possibility to express requirements that refer not only to an event, but also to the occurrence number “i” of this reference event;
- More requirement types can be expressed using our work. For instance, one can express interval constraints using new elementary requirements that we have defined and which cannot be expressed using the previously mentioned classifications.

5. The Verification Process
Both, temporal requirement classification and SEG, are used as a start step for a new verification method of temporal requirements. In this section, first, we introduce the next step of this new verification method which is the observation pattern repository. Then, we briefly outline an overview of the global method that we are developing.

5.1 Observation Pattern Basis
A pattern is a commonly reusable model in software systems that guarantee a set of characteristics and functionalities. The identification of a pattern is based on the context in which it will be used. The goal behind developing patterns is to offer a support for system design and development [16]. Moreover, using patterns helps keep the design standardized and useful, and
minimizes the reinventing in the design process, since they facilitate reusability and knowledge capitalization [17].

In this work, we have set up a repository comprising generic timed automata observation patterns to watch all the common temporal requirements that we defined within our classification (cf. Table 1). An observer [18] is a TA model with, inter alia, a specific error-state “KO” reached as soon as the associate requirement is violated. The basis of patterns is introduced regardless of the system specification. Each pattern is assigned to a different requirement category.

5.2 Verification Process

In practice (Fig. 2), the temporal requirements extracted (using our SEG) from the specification of the system to be checked are classified based on their types. For each extracted requirement from the investigated system, the suitable observation patterns (See section 5.1) are picked up and instantiated [19]. This instantiation step generates a set of TA Observers for the monitored requirements. On the other hand, the model to be checked, which is depicted in the shape of Unified Modeling Language (UML) State Machines (SM) [20] with temporal annotations, is automatically translated into more formal notation, the TA, which provides support for analytic verification. The translation is done according to a transformation algorithm that we have developed [21]. The generated model is synchronized with the instantiated TA
observers to obtain a global model, as shown in Fig. 2. Hence, the verification task is performed on the basis of this obtained model, with a reachability analysis while checking whether the observers KO-states—corresponding to requirements violation—are reachable. Uppaal [2] is used as a back-end tool to carry out this investigation.

6. Conclusions and Perspectives

In this paper, we have suggested some means to assist and guide the user while specifying temporal requirements. First, we have established a new classification of all the common temporal requirements that one may meet when dealing with specification of time-constrained systems. The advantages of our classification are: (1) It is exclusively event-based, which ensures a certain homogeneity between the requirement assertions; (2) It offers the ability to express more temporal requirements types; and finally (3) Using this classification, it becomes feasible to deal with repetitive events. Second, we have introduced a formal structured English grammar (SEG) for describing all the temporal requirements types that we have identified within the new classification. This SEG consists of a set of natural-language/literal assertions with rigorous semantics. The aim of defining the SEG consists in hiding abstract formal aspects from the user, while specifying temporal requirements. Hence, SEG syntax is intuitive, albeit being at the same time, rigorous and based on formal foundations. Third, we have shown how this new classification is used to define an observation pattern repository.

To summarize, the specification/verification chain is made up of four processes: (1) First, the temporal requirements are expressed on the basis of our SEG. Once expressed, the type of each requirement is determined, since, for each requirement, we have associated a generated sentence and vice-versa; (2) Second, for each requirement the appropriate TA observation pattern is picked up from the generic repository, then instantiated to produce an associate observer. This process results in some TA observers. Each TA observer watches an elementary temporal requirement; (3) On the other hand, the specification of the system to be checked is abstracted and translated into TA model; (4) Finally, the instantiated TA observers are synchronized with the obtained (via model transformation) TA models to generate a global model holding both the specification of the system to be checked and the requirement monitoring. Consequently, the verification task is reduced to an error-state (“KO” node) reachability analysis on the global model obtained.

We aim at extending our work by allowing the definition of more complex temporal requirements. In fact, experience has shown us that, sometimes, one may need to define requirements that are a composition of several requirements. Thereby, we will define patterns to handle various relations linking elementary requirements or even linking complex requirements. Consequently, relations over (elementary or complex) requirements such as “AND”, “OR” and “XOR” can be defined using our patterns basis.

Moreover, we wish to extend the current SEG to deal with requirements that refer to other requirements. Indeed, the idea is to allow the user to express requirements where the reference events as well as the monitored events refer to the validation or the violation of other requirements, and not only to events that come from the checked model. Besides, this step seems easily implementable, since we presented an event-based classification as well as event-based pattern repository. However, a close view is still needed.

Acknowledgment

This research has been partially supported by Région Nord Pas de Calais and European fund Feder under the FUI National project FerroCOTS, labelled by i-Trans.

References


