10. Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic

Nesredin Mahmud, Cristina Seceleanu and Oscar Ljungkrantz. *In the International Conference on Software Engineering and Formal Methods (SEFM)(pp. 332-348). Springer, Cham, 2017.*

Abstract: Due to the increasing complexity of embedded systems, early detection of software/hardware errors has become desirable. In this context, effective yet flexible specification methods that support rigorous analysis of embedded system requirements are needed. Current specification methods such as pattern-based, boilerplates normally lack meta-models for extensibility and flexibility. In contrast, formal specification languages, e.g., temporal logic, Z language, etc., enable rigorous analysis; however, they usually are too mathematical and difficult to be understood by the average software engineers. In this paper, we propose a specification representation that considers thematic roles and domain knowledge, which enable a deep semantic analysis of requirements. The specification is complemented by our constrained natural language specification framework, ReSA, which acts as interface to the representation. The representation that we propose is encoded in description logic, which is decidable and computationally tractable ontology language. With support from the ontology reasoner, Hermit, we check for consistency and completeness of requirements. Moreover, we show transformation of the ontology-based specifications into Timed Computation Tree Logic formulas, to be used further in model checking embedded systems.

10.1 Introduction

The difficulty of specifying embedded systems requirements has increased due to the high degree of systems complexities, and the dependability expected from the solutions, especially of safety-critical systems [23]. The failures to properly operate safety-critical systems could be a catastrophe, causing property damages and casualties. Consequently, the development of such embedded systems demand stricter approaches of requirements specification and their analysis. However, effective specification methods and tools that meet industrial needs, e.g., engineer friendliness, flexibility, fine-grained analysis etc., are lacking. If we consider the automotive industry as the domain of focus, current experience tells us that: i) first, most requirements specification documents are expressed in natural language. As a result, it is not uncommon to frequently find inconsistent, ambiguous, vague and imprecise specifications; ii) second, even if engineers use semi-formal specification methods such as requirements boilerplates [15] and property specification patters [10][19] as a solution to the aforementioned problems, these methods have fixed set of templates that frequently limit specification expressiveness, and rigid syntactic structures that disregard the use of different but equivalent sentences alterations to improve flexibility of the language; iii) last but not least, the requirements specification methods, mentioned earlier, does not support computational semantic analysis [16] as used in linguistics, hence making the requirements analysis shallow.

Current solutions to the requirements specification and analysis problem involve one or more of the following techniques: domain knowledge, semantics analysis, natural language processing (NLP), logic-based reasoning, conceptual modeling in object-oriented development, property specification patterns, etc. NLP techniques and domain ontologies are used to ease manual selection of boilerplates [11][3], which is error prone and demanding. However, the approach suffers from inaccurate translations and a lack of robust meta-model that supports scalability and prevents from inconsistent definition of boilerplates. The scalability issue is also shared in property specification patterns [10][19], which define limited sets of patterns that allow specification of the functional and timing requirements of embedded systems. A linguistic approach that uses domain ontology is also proposed [17], to specify and analyze requirements. However, the proposed semantic relations and the thesaurus are very limited, consequently shallow semantic representations with little application on real-world problems.

To address the above problems, in this paper, we propose a flexible yet rigorous specification of embedded systems requirements. In our previous proposal [21][22], we introduced a requirements specification language *ReSA*, which is a constrained natural language tailored to specify embedded systems requirements. Here, we combine *semantic analysis* and *domain ontology* for semantic representation of ReSA requirements specifications

(ReSA specifications). The semantic analysis enables the consideration of lexical semantics and semantic relations into the analysis of requirements, hence fine-grained. The semantic representation employs an *event-based* approach [9] in conjunction with *thematic roles* [28]. The ontology is encoded in the *description logic* (DL) [1], which is the decidable and computationally tractable part of firs-order logic (FOL). Consequently, description logic is a viable solution for the core reasoning of requirements specifications, such as consistency, completeness and entailment. The ontology is also used to automatically generate optimal temporal computation tree logic (TCTL) properties from ReSA specifications. The ontology is developed using OWL in the Protégé tool. Our approach is demonstrated on some requirements from the *Adjustable Speed Limiter* (ASL) automotive use case, which is an operational and safety critical automotive system in Volvo tracks. ASL limits vehicle speed to not exceed a predefined speed, set by the driver.

The rest of the paper is organized as follows. In Section 10.2, we give a brief overview of ReSA, DL, and TCTL. In Section 10.3, we show the semantic representation of ReSA specifications using the event-based approach. We define the equivalence axioms that increase the flexibility of the ReSA specification in Section 10.4, followed by analysis of ReSA specifications in Section 10.5. In Section 10.7, we discuss the related work. Finally, we conclude the paper and outline possible future work in Section 10.8.

10.2 Preliminaries

In this section, we briefly overview our requirements specification language, ReSA and its underlying semantic encoding language, DL. Furthermore, we briefly explain Temporal Computation Tree Logic (TCTL).

10.2.1 ReSA

In our previous work [21], we proposed the requirements specification language, ReSA, which closely renders the syntax and semantics of natural language, and also allows engineers to express requirements in a constrained manner. The language targets the specification of requirements in embedded systems, hence uses concepts such as *System*, *Para{meter}*, *Device*, *Mode*, *State*, *Event*, etc., defined in [21]. Further, the language has axioms that guide the specification process, improve the readability, and reduce ambiguity. The concepts and the specification axioms are maintained in a system-level ontology. The ontology is employed for type checking during the specification process in the ReSA Editor [22]. Examples of requirements expressed in ReSA are shown in structured (1-a), tagged (1-b), and plain (1-c) format.

(1) (a) If “activation button”:inDevice is pressed
then
  ASL:system shall limit “vehicle speed”:parameter within 500ms
endif

(b) If “activation button”:inDevice is pressed, ASL:system shall limit “vehicle speed”:parameter within 500ms.

(c) If activation button is pressed, ASL shall limit vehicle speed within 500ms.

Valid strings of the language include requirements boilerplates, which are sentential-forms with dynamic (variable) syntactic elements. For the example (1), the boilerplates that are instantiated to construct the requirement are show in Table 10.1 (the variable elements are enclosed within angle brackets).

Table 10.1: *Boilerplates Applied in Example (1)*

Boilerplate	**Type**
⟨InDevice⟩ is ⟨Action⟩	Proposition
⟨System⟩ shall ⟨Action⟩ ⟨Parameter⟩	Simple
... within ⟨Time⟩	Prepositional Phrase
if ⟨Antecedent⟩ then ⟨Consequent⟩ endif	Conditional

At this stage, the language lacks a semantic representation that defines the lexical semantics (e.g., semantic roles, quantification, etc.) and semantic relations (e.g., opposite, synonyms, antonyms, hyponyms and hypernyms, etc.), which are crucial not just to detect non-trivial specification errors, such as inconsistency and incompleteness, but also to discover implicit knowledge of the specifications from explicitly stated requirements, hence providing better insight into the specifications. In this paper, we give semantics in DL that enables such functionality.

10.2.2 Description Logic

Description logic (DL) [1] is a language for knowledge representation and is mainly used in artificial intelligence [5], semantic web [4] and biomedical informatics [30]. It is designed to contain the decidable fragments of first-order logic, for which efficient reasoners exist, e.g., FACT^{++}, HermiT [34].

A DL knowledge base $K = (T,A)$ contains terminological assertions (a.k.a. TBox, T) and instances (or facts) assertions (a.k.a. ABox, A), and it is built recursively from concepts (unary predicates), roles (binary predicates) and instances (constants) via *Constructors*. DL is inspired by set theory; consequently, the interpretation I of the knowledge-base K is a tuple (Δ^I, F^I), where: Δ^I is the domain, F^I is an interpretation function over the domain, which relates a concept A to a set $A^I \subseteq \Delta^I$, and a role R to a binary set $R^I \subseteq \Delta^I \times \Delta^I$. The knowledge base is consistent if $I \vDash K(or \vDash T \wedge \vDash A)$ [1]. Table 10.2 shows the constructors used in this paper, where: (C, D, $\top$ (everything), $\bot$ (nothing) are concepts, and R, S are roles).

Table 10.2: *DL constructors*

Constructor	Usage	Semantics
$\sqcup$ (Union)	$C \sqcup D$	$C^I \cup D^I$
$\sqcap$ (Intersection)	$C \sqcap D$	$C^I \cap D^I$
$\exists$ (Existential Qunat.)	$\exists R.C$	$\{x \mid \exists y(x,y) \in R^I \rightarrow y \in C^I\}$
$\forall$ (Value Restriction)	$\forall R.C$	$\{x \mid \forall y(x,y) \in R^I \rightarrow y \in C^I\}$
o (Role Composition)	$R \text{ o } S$	$\{(x,z) \in \Delta^I \times \Delta^I \mid \exists(x,y) \in R^I \wedge \exists(y,z) \in S^I\}$
$\sqsubseteq$ (Concept Inclusion)	$C \sqsubseteq D$	$C^I \subseteq D^I$
$\sqsubseteq$ (Role Inclusion)	$R \sqsubseteq S$	$S^I \subseteq R^I$
$\equiv$ (Equivalence)	$C \equiv D$	$C^I = D^I$
: (Assertion)	x:C	$a^I \in C^I$
	(x,y):R	$(x^I, y^I) \in R^I$
$\doteq$ (Definition)	C$\doteq$D	$C^I = D^I$

Besides storing the semantically consistent specifications, the ontology can be used as a knowledge base that other systems can use. In this paper, we show how we generate time-bounded response formulas, in TCTL, from the ontology.

10.2.3 Timed Computational Tree Logic

TCTL [2], the timed extension of CTL, is used in this paper as the notation that encodes formalized properties, after the transformation from their ontology-based representation. A well-formed TCTL property is inductively generated based on the production rule (10.1) using the minimal set of operators. $\{true, \neg, \vee, AU, EU\}$.

$$\phi := true \mid p \mid \neg p \mid \phi \mid \phi \vee \phi \mid \mathsf{E}[\phi \mathsf{U}_{\bowtie t} \phi] \mid \mathsf{A}[\phi \mathsf{U}_{\backsim t} \phi] \qquad (10.1)$$

where: p ranges over a set of atomic formulas, A and E are the universal and existential path quantifiers, respectively, U (Until) is a temporal operator and, $\bowtie$ represents one of the relational operators: $\{=, <, \leq\}$. Equivalently, the operators $\Box$ and $\Diamond$ are defined as $\neg E[trueU \neg \phi]$ and $A[trueU \phi]$, respectively.

We use the following (T)CTL property types (p, q are state properties):

- Invariance: $A \Box p$ - The specification evaluates to true if (and only if) every reachable state satisfies p. In this property, A is the universal path quantifier, whereas $\Box$ is the path-specific temporal operator.

- Time-bounded Response: $A \Box (p \Rightarrow A \Diamond_{\leq t} q)$. This property asserts that, for all paths, it is always the case that once p holds, q eventually holds within t time units ($\Diamond_{\leq t}$ is the "eventually within t" operator).

10.3 Defining ReSA Semantics in DL

In this section, we describe a framework for semantic analysis, the semantic representation approach and the domain ontology that captures the representations as a knowledge base system.

10.3.1 Semantic Analysis Framework

In order to speed up time to market, reduce cost of software development and improve productivity, an integrated tool support is crucial. In the context of requirements specification, industrial tools need to support detection and correction of specifications errors at the syntactic and semantic levels. Figure 10.1 illustrates our proposed framework for specifying and analyzing ReSA specifications using semantic analysis that takes advantage of open source lexical resources and ontology [1].

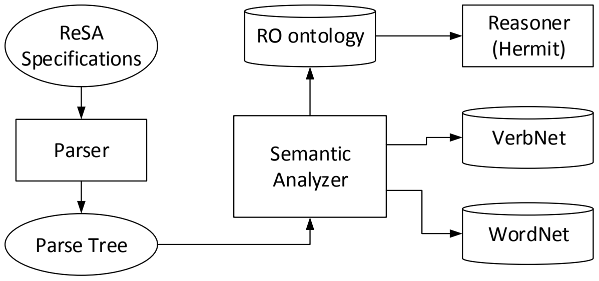

Figure 10.1: Semantic Analysis Framework

The semantic analysis process is described briefly as follows: i) first, requirements are specified in ReSA, which pass through syntactic and spelling validations; ii) during the parsing of the specifications, the semantic analyzer determines the semantics of each specification, by consulting the *VerbNet* [33][26] and the *WordNet* [24] lexical resources through Application Program Interfaces (API), as explained in Section 10.3. VerbNet is a popular open source lexical resource of verbs, which provides verbs semantics, whereas WordNet provides conceptual semantics and lexical relations of words, e.g., antonyms, synonyms, etc.; ii) third, the *Semantic Analyzer* asserts the domain types and semantic roles of the lexical elements, and the logical forms of the specifications, into the ontology, also known as *ReSA Ontology* (RO). Each logical form captures the semantic structure of a sentence as opposed to the surface (syntax) of a sentence; iii) finally, RO is checked for consistency, completeness and entailment relations via an ontology reasoner (or reasoner engine) such as Hermit [34].

[1]The *Oval*, *Rectangular*, and *Cylindrical* shapes represent artifacts, computing functions, knowledge base, respectively.

Table 10.3: *The Definition of Thematic Roles. For a Comprehensive Definition, Refer to the VerbNet Lexical Resource [33].*

Thematic Role	**Description**
Agent	initiates and carries out an action, and exists independently of the action.
Attribute (Attr)	is the property of an entity
Patient	undergoes state changes and exists independently of the action
Theme	similar to Patient, but doesn't undergo state changes
Recipient	destination of an action
Instrument	used by an agent to cause an action, and exists independently of the action

10.3.2 Semantic Representation via the Event-based Approach

The semantics of each specification is computed compositionally from the predicates (i.e., the main verbs) and operators (i.e., the prepositions and conjunctives) via an *Event-based semantics* approach [27][7]. In this approach, an event E denotes an occurrence, state or condition. It is expressed using a clause, an adjunct or a statement. The arguments of a predicate or an operator are mapped to an existentially quantified event through *thematic roles* [28], which define the sematic roles of arguments with respect to the predicate. In Table 10.3, we briefly define the commonly used thematic roles, of which some are used in this paper.

To illustrate the event-based approach and thematic roles, let us consider a simple ReSA specification (2-a). The predicate is Limit(args.) and its arguments are ASL and vehicle speed. The argument ASL has an agent role in the predicate which enables the vehicle speed argument to undergo change of state. Hence, the latter argument has a patient(theme) role. The thematic roles are applied in the event-based semantic representation. Considering the same example, the theme of the specification is about `Limiting`, hence the event Limiting(e) (or, E(e) & Action(e, limit)). The the arguments ASL and vehicle speed are related to the event through the Agent and Patient thematic roles, respectively, as shown in (2-b). Further, the thematic roles are restricted to the domain concepts in order to effectively represent the semantics of the specification using *semantic selection* [25], e.g., Agent(e, ASL) & System(ASL).

(2) (a) ASL:system shall limit “vehicle speed”:para.

(b) ∃e.[Limiting(e) & Agent(e, ASL) & Patient(e, “vehicle speed”) & System(ASL) & Para(“vehicle speed”)]

The benefits of our semantic representation are threefold: i) first, event-based representations are suitable to automate while delivering deep and complex semantic representations through nesting; ii) second, voice alternations, using active/passive forms, can be represented with almost similar semantic representations, allowing flexibility; and iii) third, our domain concepts are well-defined in advance, hence the concepts are applied on the thematic roles effectively to restrict the selection of arguments, thereby discarding semantically not sound statements, such as the driver is activated.

The main challenge of applying thematic roles is the lack of a standard definition. Some thematic roles have inconsistent definitions across the various literatures. In order to mitigate this problem, we opt to apply thematic roles from the VerbNet, which defines around 23 thematic roles and over 5200 verb classes [33].

10.3.3 The ReSA Ontology

The ReSA Ontology (RO) is an upgrade to our previously defined system-level ontology [21] with semantic representations, based on the event-based and thematic roles. Further, RO introduces complex concepts such as *Entity*, *Attribute*, *User*, which categorize the embedded systems concepts based on semantic similarities. In this ontology, we also define equivalence and definition axioms that realize the flexibility of ReSA, to provide engineers with the choice of equivalent words and phrases, equivalent operators such as prepositions and conjunctives, and sentence alterations (passive and active statements). The ontology can be downloaded from Bitbucket[2].

The RO ontology consists of the TBox and the ABox DL parts. The TBox contains assertions of logical forms, thematic restrictions, classifications of domain concept and statements. The TBox classification assertions are created once and maintained by ontology experts with domain knowledge. The TBox is also populated with logical forms that conform to various statement types during parsing of the requirements specifications. The logical form can be captured through simple and complex event structures as shown in the next section. The TBox assertions are the meta-language of the ABox that users eventually assert to it during specification. In contrast, the ABox contains assertions of concrete instantiations of concepts and roles.

Thematic Role Restrictions: The thematic roles restrictions in the TBox detect implausible requirements specifications, such as "a user is activated" ($\exists$e.[Patient(e, x) & User(x) &...]). The given example implies that the user has a *Patient* role, which is considered an illegal assertion since a user cannot change the state upon receiving an action according to our interpretation. In order to apply the restrictions, first, we generalize the domain concepts into complex concepts (*Entity*, *Attribute* and *User*) based on semantic similarity (3). Next, through expert knowledge and analysis of corpus data, we apply the restrictions following the classification, as shown in (4). The restrictions are defined on the range of roles, so that the roles map to the appropriate concepts, for instance the range of the *Patient* role is restricted to *Entity* and *Attribute* concepts only.

[2]https://bitbucket.org/nasmdh/ro

(3) $\text{System} \sqsubseteq \text{Entity}, \text{Device} \sqsubseteq \text{Entity}$
$\text{Para} \sqsubseteq \text{Attribute}, \text{State} \sqsubseteq \text{Attribute}, \text{Mode} \sqsubseteq \text{Attribute}$

(4) $\top \sqsubseteq \forall\ \text{Patient.(Entity} \sqcup \text{Attribute)}$
$\top \sqsubseteq \forall\ \text{Agent.(User} \sqcup \text{Entity)}$
$\top \sqsubseteq \forall\ \text{Instrument.(InDevice)}$

10.3.4 Semantics of Clauses and Statements in ReSA

The restricted thematic roles are binary predicates that relate verb and operator arguments to their corresponding events, E. Events can be expressed using a clause ($\mathsf{E}_{\text{clause}}$), time phrase ($\mathsf{E}_{\text{time}}$), simple ($\mathsf{E}_{\text{simple}}$) and complex statements. Further, a simple statement can be timed ($\mathsf{E}_{\text{tsimple}}$), and a complex statement can be compound complex ($\mathsf{E}_{\text{ccomplex}}$) or nested complex ($\mathsf{E}_{\text{ncomplex}}$), as shown in (5).

(5) $$\left[\ \mathsf{E}_{\text{clause}}, \mathsf{E}_{\text{time}}, \mathsf{E}_{\text{simple}}, \mathsf{E}_{\text{tsimple}}, \mathsf{E}_{\text{complex}}, \mathsf{E}_{\text{ccomplex}}, \mathsf{E}_{\text{ncomplex}}\ \right] \sqsubseteq \mathsf{E}$$

Clause event E_{clause}**:** n ReSA, a clause is an atomic proposition (or a simple statement without an adjunct). It expresses an *action* (action clause), or describes a *state* or *condition* (descriptive clause). The predicates of the action clauses are transitive E_{tra} (e.g., activate [object]), intransitive E_{int} (e.g., reboot), or ditransitive E_{dit} (e.g., send[object] to [indirect object]), whereas the descriptive clauses usually assume copula verbs E_{cop} (e.g., is). In the following example, we show how the semantics of a clause, specified in ReSA, is represented via the clause event.

(6) ASL:system shall reboot /* valid ReSA clause */

(7) Event-based: $\exists$e.$\big[$ E_{int}(e) & Theme(e, ASL) & Action(e, reboot) & System(ASL) & IntV(reboot)$\big]$

(8) (a) TBox: $\mathsf{E} \sqcap \exists\text{Agent.System} \sqcap \exists\text{Action.IntV} \sqsubseteq \mathsf{E}_{\text{int}}$
(b) ABox: e:E_{int}, (e, ASL):Agent, (e, reboot):Action, ASL:System, reboot:IntVerb

The thematic roles of the "reboot" argument, in clause (6), are fetched from the VerbNet semantic resource, which are used in the semantic representation (7). In TBox, we assert the logical form of the specification as shown in (8-a), which states, the event is an element of top event E which is related to instances of System and Intransitive Verb (IntVerb) through the Agent and Action thematic roles, respectively, and it belongs to the clause event. In the ABox, we assert the facts (8-b), which are the instances of the clause event, the predicate and its arguments as stated in the specification.

Simple event: This event type is expressed via a simple statement that makes use of a single clause and an optional adjunct. A *Timed-Simple* event is a simple event extended with an adjunct that states timing information (or uses a prepositional phrase of time). Its event structure is constructed from two events as shown in Figure 10.2(a). The timed-simple event e1 has outer scope over the adjunct event e2, which indicates the extension of e1 with e2 and the method we employ to resolve scope ambiguity for quantified statements [20]. The event structure is constructed during parsing according to the input string matches. An example of a timed-simple requirement, which corresponds to Figure 10.2(a), is shown in (9) and its representation using event-based and DL approach.

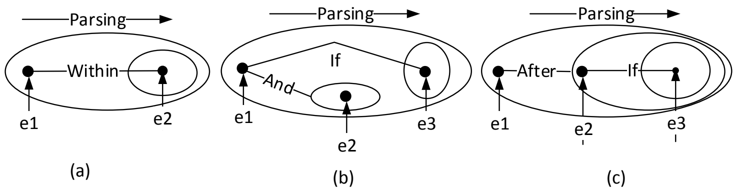

Figure 10.2: Event Structures

(9) Timed-Simple: $\{$ASL:system shall limit "vehicle speed":para $\{$within 500ms.$\}_{e2}\}_{e1}$
Event-Based:
$\exists e1.[E_{tsimple}(e1)\&\exists e2.[E_{time}(e2)\&Value(e2,500ms)\&Within(e1,e2)]]$
TBox: $E \sqcap \exists Value.Time \sqcap \exists Within.E_{time} \sqsubseteq E_{tsimple}$

Complex event: In contrast to a simple event, a complex event is expressed via a complex statement that uses subordinate conjunctives, e.g., **if**, **when**, etc. A typical application of this event is in denoting conditionals. Its event structure is similar to the timed-simple event, except in this case we use subordinate conjunctives (10). A compound-complex contains statements that are connected using the *and/or* conjunctives. Figure 10.2(a) and 10.2(c) show the event structure for compound-complex and nested-complex events, respectively, and examples are shown for each class of complex statements that capture their respective event structure in (11)(12). ***Note:*** *Curly braces are used in order to simplify the reading.*

(10)	Complex:	{ if “activation button”:inDevice is pressed, {“ASL”:system shall be activated}$_{.e2}$}$_{e1}$
	Event-Based:	$\exists e1.\left[E_{complex}(e1)\&...\exists e2.\left[E_{simple}(e2)\&...\&If(e1,e2)\right]\right]$
	TBox:	$E \sqcap \exists If.E_{simple} \sqsubseteq E_{complex}$
(11)		
	Compound-Complex:	{ if vehicle:entity is in running:mode and {“enabling button”: inDevice is pressed}, {ASL:system shall be enabled.}$_{e3}$}$_{e2}$}$_{e1}$
	Event-Based:	$\exists e1.\left[E_{ccomplex}(e1)\ \&...\exists e2.\left[E_{simple}(e2)\ \&...\ And(e1,e2)\right]\ \&\ \exists e3.\left[E_{simple}(e3)\ \&...If(e1,e3)\right]\right]$
	TBox:	$E \sqcap \exists And.E_{simple} \sqcap \exists If.E_{simple} \sqsubseteq E_{ccompound}$
(12)		
	Nested-complex:	{ After “ASL”:system is enabled; {if “activation button”:inDevice is pressed, {“ASL”:system shall be activated.}$_{e3}$}$_{e2}$}$_{e1}$
	Event-Based:	$\exists e1.\left[E_{complex}(e1)\ \&...\exists e2.\left[E_{simple}(e2)\ \&...\ After(e1,e2)\ \&\ \exists e3.\left[E_{simple}(e3)\ \&...If(e2,e3)\right]\right]\right]$
	TBox:	$E \sqcap \exists After.E_{complex} \circ If.E_{simple} \sqsubseteq E_{ncomplex}$

10.4 Semantic Equivalence in ReSA

ReSA allows specification of semantically equivalent statements via different syntactic constructions, e.g., by changing the voice of a statement (or diathesis). The advantage of having alternative ways of specifying requirements is that engineers get the choice to select convenient syntactic constructs when expressing requirements, hence increasing the language's usability in practice. However, the task of identifying semantically equivalent constructs is not trivial, as equivalences exist not just at the statements level but also at syntactic categories level. Besides, a lot of flexibility could contribute to ambiguity and impreciseness of specifications [13], therefore, we consider the trade offs carefully in the defining the equivalence axioms. In our context, we define semantic equivalence as:

Definition 1 (Semantic equivalence) *The constituents `C1` and `C2` of ReSA specifications are semantically equivalent if and only if `C1` and `C2` belong to the same syntactic categories, and C1 is substitutable for C2, and vice versa, in any instances of their usage in requirements specification.*

The notion of semantic equivalence exists at the concept, role and instance levels of the ontology. Equivalent concepts have the same number of instances in the domain, e.g., $\mathsf{Parameter} \equiv \mathsf{InParameter} \sqcup \mathsf{OutParameter}$. Equivalent roles relate the same domain-range concepts and denote the same properties, e.g., $\mathsf{Greaterthan} \equiv \mathsf{LessThanorEqTo}^{-}$. Equivalent instances denote the same value or entity. Since model elements are normally distinct, we consider the N/NPs instances distinct (or not equivalent) in the ontology; however, there exist equivalent instances of verbs and description words. For the latter case, the WordNet lexicon resource provides semantic and syntactic relations between lexicons, e.g., synonyms, antonyms, etc. We use the WordNet lexical database [24] and the JWNL library [14] to automate the identification of synonyms and antonyms for the verb and descriptive words of the ReSA syntactic categories.

Tables 10.4 illustrates the various equivalences axioms that exist in RO. The prepositions *within* and *between* are defined in terms of the After_p and Before_p roles, similarly the conjunctive *between*, in terms of the Before_{sc} role. The comparison operators are defined in terms of the Lessthan and Equalto roles. Antonyms and synonyms are defined in terms of the Ant and Syn roles, respectively.

Table 10.4: *Equivalence Axioms in ReSA Ontology*
sc-subordinating conjunctive, p-preposition

Subordinate Conj. Equiv. $\mathsf{Before}_{sc} \equiv \mathsf{After}^{-}_{cc}$ $\mathsf{Between}_{sc} \doteq \mathsf{Before}_{sc} \circ \mathsf{Before}_{sc}$	**Synonymous** (limit, restrict):Syn (display, expose):Syn
Prepositions of Time Equiv. $\mathsf{Between}_{p} \doteq \exists \mathsf{After}_{p}.\mathsf{Time} \sqcap \exists \mathsf{Before}_{p}.\mathsf{Time}$ $\forall \mathsf{Within.Time} \equiv \exists\ \mathsf{After}_{p}.\{\mathsf{Now}\} \sqcap \mathsf{Before}_{p}.\mathsf{Time}$	**Antonyms** (enable, disable):Ant (active, inactive):Ant
Comparison Equivalence $\mathsf{Greaterthan} \equiv \mathsf{LessThanorEqTo}^{-}$ $\forall \mathsf{LessThanorEqTo} \equiv \exists \mathsf{Lessthan}^{-} \sqcup \exists\ \mathsf{Equalto}\ \mathsf{GrThanorEqto} \equiv \mathsf{Lessthan}^{-}$	

10.5 Automated Analysis and Transformation to TCTL Properties

So far, we have presented the semantic representation of ReSA and ReSA specifications in event-based semantics, encoded in DL. We have also discussed the contributions of the thematic roles and the equivalence axioms for enabling flexible specifications. In this section, we present the various reasoning (or analysis of requirements specifications) that we can carry out on the RO ontology. Moreover, we show the transformation of the ReSA specifications to TCTL properties, exemplified on the time-bounded properties.

10.5.1 Consistency checking

The ontology *RO* is consistent if there exists an interpretation (or a model) that satisfies the TBox (T) and the ABox(A) assertions, i.e., $M \models ax$, where: $ax \in T \cup A$. Inconsistency in the ontology can occur due to inconsistencies at different levels of syntactic categories and statements. For instance at the lexical syntactic level, the quantifiers of model elements are unique, hence D$\not\models$ (ASL:System & ASL:Parameter), i.e., the stated facts are inconsistent.

At the clause level, if we consider Opposite(active,inactive), the fact that clauses e1 and e2 (13) are contradictory is detected using the assertion in (14), stating that a contradictory clause related through a reflexive chains of roles.

(13) e1="ASL is active" and e2="ASL is inactive"

(14) $\top \sqsubseteq \exists$ (Theme$^{-}\circ$ Att$\circ$ Opposite $\circ$ Att$^{-}\circ$ Theme)

10.5.2 Completeness

The completeness of the requirements specifications is checked indirectly from the completeness of the ontology. The ontology is complete if every instance of the syntactic categories has thematic roles (15), and every clause belongs to some statement (16). In the completeness axioms, we assume that the ontology is not empty, since an empty ontology would satisfy universally quantified axioms.

(15) NP $\sqcup$ VP $\sqcup$ ADJ $\sqsubseteq \exists$ThematicRole.E

(16) Clause $\sqsubseteq \exists$ThematicRole.E

10.5.3 Transformation to TCTL

In the ABox, RO maintains concrete representations of requirements specifications, and such representations can be transformed into different formal logics for further analysis. In this paper, we show the transformation of specifications in the ontology to their counterpart in TCTL, using Specification Pattern System (SPS) [19][29], which is a collection of recurring patterns of functional and timing requirements. The SPS patterns have formal semantics in various temporal logics, including TCTL. The benefits of automatically generating optimal TCTL properties are: i) first, properties are not generated for inconsistent specifications; and ii) second, properties are not generated for entailed specifications. Consequently, the effort of subsequent model checking is reduced by optimizing the number of properties to be checked.

Algorithm 1 shows the generation of a time-bounded response formula in TCTL. The input to the algorithm consists of sets of simple and complex events, paired sets of *If* and *Within* roles, and paired sets of thematic roles.

For event *e* in the Complex set, we check if it is related with the *If* role. If it is, we extract its thematic arguments and verbalize the arguments (construct equivalent ReSA syntax), and store them into *p*. Besides, we get the successor of *e*, which is represented by $e1$. By using $e1$, we get its arguments and verbalized to *q*. Finally, we get the successor of $e1$, which is represented by $e2$, and from the last event, we get the time argument, t. The output is a time-bounded TCTL formula $A\Box(p \Rightarrow A\Diamond_{\leq t}q)$, where *p*, *q* are clauses that are expressed in ReSA, and *t* is a time bound. Other properties can be extracted similarly.

```
input : A ReSA
        specification⟨$E_{tsimple}$⟩
output: Time-bounded Response
        Formula
while e in Complex do
    arg ⟵ getArguments(e);
    p ⟵ verbalize(arg);
    e1 = getEvent(e,If);
    if e1 is found then
        arg ⟵
         getArguments(e1);
        q ⟵ verbalize(arg);
        e2 ⟵
         getEvent(e1,Within);
        if e2 is found then
            arg ⟵
             getArguments(e2);
            t ⟵ verbalize(arg);
        end
    end
    tctl_boundedResponse[0,1,..i]
     ⟵ $A\Box(p \Rightarrow A\Diamond_{\leq t}q)$.
end
```

```
Complex={e1,
e2,...en};
TimedSimple={em,
em+1,...
If={<e1,em>,..};
Within={<em,t1>,...};
Agent={<e1,
ASL>,...};
Patient={<e2,vehicle
speed>}; ⋮
```

Algorithm 1: Extracts Time-bounded Response from the ReSA Ontology, RO

10.6 Use Case: Adjustable Speed Limiter (ASL)

We show the application of our approach in selected requirements from the Adjustable Speed Limiter (ASL) system. The system is operational and safety critical which is found in Volvo trucks. It limits vehicle speed from exceeding a predefined speed set by the driver. Table 10.5 shows four requirements of type functional, constraint, timing and safety that are realized by the system. The requirements are expressed in the ReSA language.

The requirements are inserted into the RO ontology via the Protégé user interface. Table 10.6 displays the event types and the number of events used for representing the semantics of each requirements. For instance, **Req1** is

ReqId	Requirement Specification in ReSA	Type
Req1	**Enabling ASL:** when "the driver":user selects "ASL speed control":mode AND "the vehicle":entity is in "prerunning mode", "the ASL":system shall be enabled AND "the ASL enable status":status shall be presented to "the driver".	Functional
Req2	**ASL activation precondition:** ASL:system is activated only if ASL:system is enabled AND vehicle is in "running mode" AND "Engine speed control":component is not active.	Constraint
Req3	**ASL deactivation time:** When/if the driver presses DACT:inDevice, "the ASL":system shall deactivate within 160ms.	Timing
Req4	**ASL internal fault:** if "fault affecting":event occurs on ASL:system while "the ASL":system is active, ASL:system shall be deactivated.	Safety

Table 10.5: ASL requirements samples expressed in ReSA

expressed using four events: a compound complex event (Selecting(e1)), two simple events (InPrerunning(e2), Presenting(e3)) and a compound simple event (Enabling(e)). The type and number of thematic roles employed in relating the events to their corresponding arguments are shown in Table 10.7. In summary Table 10.8, the RO ontology consists of 168 logical axiom assertions that cover concepts and roles assertions, concept and role individuals assertions, complex concept assertions, equivalence assertions, disjointness assertions, individuals distinct assertions but exclude annotations assertions (with annotations assertions, it amounts to 300 axioms).

Even Type	Req1	Req2	Req3	Req4
Simple	2	1		1
Time			1	
TimedSimple			1	
Complex		1	1	1
CompoundSimple	1	1		
CompoundComplex	1			
NestedComplex			1	1
Total	**4**	**3**	**4**	**3**

Table 10.6: *Number of Events*

Thematic Role	#Roles
Agent	2
Patient	1
Theme	5
Instrument	1
Att(ribute)	4
Total	**13**

Table 10.7: *Thematic Roles*

Metric	**#Assertions**
Axiom	300
Logical Axiom Count	166
Class Count	49
Object Property Count	7
Individual Count	45
DL Expressivity	ALCROI(D)

Table 10.8: *RO ontology Metric*

In the current version of the RO ontology, the DL variant has expressivity of *ALCROI(D)* [3]. It is decidable and more expressible than its base description language *ALC*, however, the ALCROI(D) has ExpTime-hard time complexity for ABOX consistency and Concept satisfiability problems [6]. Existing and popular reasoners, such as Hermit, use optimization strategies to make core reasoning problems tractable (e.g., solvable in polynomial time) so that applications of expressible DL language have relevance for practical problems. Under the current settings of the ontology, the reasoner, Hermit (default config.), took around 300ms for the classifications of concepts. In the future, we plan to conduct further complexity analysis for core reasoning problems in the context of requirements specification analysis.

10.7 Related Work

The related work discusses the automated support of requirements specification and analysis that primary uses natural language as initial input language. In the literature, various specification and analysis methods are proposed, and the methods usually employ techniques based on logic, semantic analysis, specification patterns, Natural Language Processing (NLP), etc.

Axel et al. [8] propose requirements specification through a goal-oriented approach, where they use a requirement model (a.k.a. KAOS meta-model) to capture the possible specification scenarios and translate the specifications into temporal logic for reasoning. Similarly, Zawgi et al. [35] apply the *Default logic* to the underlying representation of constrained natural language. However, the two approaches face scalability issues that require major changes on the meta-modeling level, especially when new specification patterns are discovered. In a similar logic-based approach, the Requirement Apprentice (RA) [31] has a user-friendly requirements specification interface that makes use of *clitché library* that documents the analysts' experience, but also a general-purpose deduction-based reasoning system (Cake). The

[3]AL (Attribute Language), C (Complex concept negation), R (Limited Role inclusion assertion), O (Nominals), I (Inverse property), (D) (Data type property)

reasoning service provides detection of contradiction and completeness in the specification. However, the approach does not handle analysis that requires resolution of complex deduction problems. The approach is domain agnostic and too constraining to be applied in industry.

Haruhiko et al. [17][18] proposes a lightweight requirements analysis method, based on ontology, where the ontology is a knowledge base for treasures and inference rules. The authors also define metrics for measuring the quality of the requirements specification, including inconsistency, completeness, correctness and ambiguity. The approach lacks a well-founded formalism of the ontology language. In contrast, we use Description logic (DL), which is decidable and supported by the popular Semantic Web language, OWL. In another study by Stephen et al. [12][11], a boilerplate approach and domain-based ontology are applied in requirements specification of embedded systems. The tool prototype, DODT, supports matching of requirements to existing boilerplates through the NLP technique, and requires manual work for those requirements that do not match automatically. The boilerplates use concepts with no deep semantic relations, which could potentially lead to implausible boilerplates. In a similar approach, Michael et al. [32] uses ontology for semantic analysis of requirements that are first preprocessed through semantic labeling techniques. In the labeling, lexical words and phrases are assigned semantic roles such as *Actor* and *Object*, used for semantic analysis purposes. However, the number of semantic relations are very limited. In contrast, we use the VerbNet and WordNet lexical resources that contain substantial lexical resources.

Template-based specification methods lack flexibility due to the limited set of templates [10][19][11][21][3]. In our approach, we provide relatively more flexible requirements specification language and an ontology that captures the semantics of the specifications with reasoning support for consistency and completeness of requirements specifications.

10.8 Conclusion

Natural language is intuitive for engineers to use for requirements specification of embedded systems. However, it is frequently inconsistent, vague, ambiguous and imprecise. In order to take advantage of the appeal of natural language but also reduce the aforementioned problems, template-based methods such as pattern-based systems and boilerplates are often employed in industry, especially for safety-critical applications. However, these methods have limited sets of templates that restrict the engineers' expressiveness. The pattern-based templates abstract substantial content into propositional variables, and the boilerplates lack formal specifications and support for rigorous analysis. In this paper, we apply linguistic techniques to improve specifications and analysis. We define a semantic representation to the requirements specification language ReSA -

a constrained natural language that is close to the syntax and semantics of English.

We propose an event-based semantic representation approach that uses thematic roles and domain concepts from embedded systems, in order to effectively represent the semantics of syntactic elements such that the representations are ready for deep analysis, i.e., analysis at various levels of syntactic categories, clauses and statements. The semantics are encoded in a Description-logic-based ontology. We show how to perform consistency checking and completeness on the ontology. Finally, we demonstrate the benefit of the ontology by automatically generating optimal property specifications in Timed Computation Tree Logic (TCTL). In the future, we plan to include temporal reasoning and implementation of the ontology in the ReSA editor and validate the approach and its toolchain on industrial systems.

10.9 References

[1] *The Description Logic Handbook: Theory, Implementation and Applications.*

[2] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-Checking in Dense Real-Time. *Information and computation*, 104(1):2–34, 1993.

[3] C. Arora, M. Sabetzadeh, L. C. Briand, and F. Zimmer. Requirement Boilerplates: Transition from Manually-enforced to Automatically-verifiable Natural Language Patterns. In *2014 IEEE 4th International Workshop on Requirements Patterns (RePa)*, pages 1–8, Aug 2014.

[4] Sean Bechhofer. OWL: Web Ontology Language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[5] Mehul Bhatt and Christian Freksa. Spatial Computing for Design—an Artificial Intelligence Perspective. In *Studying Visual and Spatial Reasoning for Design Creativity*, pages 109–127. Springer, 2015.

[6] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in Expressive Description Logics. *Handbook of automated reasoning*, 2:1581–1634, 2001.

[7] Lucas Champollion. The Interaction of Compositional Semantics and Event Semantics. *Linguistics and Philosophy*, 38(1):31, 2015.

[8] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal-directed Requirements Acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.

[9] Donald Davidson. *Essays on Actions and Events: Philosophical Essays Volume 1*. Clarendon Press, 2001.

[10] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-state Verification. In *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, pages 411–420, May 1999.

[11] S. Farfeleder, T. Moser, A. Krall, T. Stålhane, H. Zojer, and C. Panis. DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded Systems Development. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 271–274, April 2011.

[12] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Inah Omoronyia, and Herbert Zojer. Ontology-driven Guidance for Requirements Elicitation. *The semanic web: research and applications*, pages 212–226, 2011.

[13] Victor S. Ferreira and Gary S. Dell. Effect of Ambiguity and Lexical Availability on Syntactic and Lexical Production. *Cognitive Psychology*, 40(4):296 – 340, 2000.

[14] Mark Alan Finlayson. Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation. In *Proceedings of the 7th Global Wordnet Conference, Tartu, Estonia*, 2014.

[15] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements engineering*. Springer Science & Business Media, 2010.

[16] Pauline Jacobson. *Compositional Semantics: An Introduction to the Syntax/Semantics Interface*. Oxford University Press, 2014.

[17] H. Kaiya and M. Saeki. Ontology based Requirements Analysis: Lightweight Semantic Processing Approach. In *Fifth International Conference on Quality Software (QSIC'05)*, pages 223–230, Sept 2005.

[18] H. Kaiya and M. Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 189–198, Sept 2006.

[19] S. Konrad and B. H. C. Cheng. Real-time Specification Patterns. In *27th International Conference on Software Engineering (ICSE)*, pages 372–381, May 2005.

[20] Howard S Kurtzman and Maryellen C MacDonald. Resolution of Quantifier scope Ambiguities. *Cognition*, 48(3):243–279, 1993.

[21] N. Mahmud, C. Seceleanu, and O. Ljungkrantz. ReSA: An ontology-based Requirement Specification Language Tailored to Automotive Systems. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2015.

[22] N. Mahmud, C. Seceleanu, and O. Ljungkrantz. Resa tool: Structured requirements specification and sat-based consistency-checking. In *Proceedings of 2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2016.

[23] Luiz Eduardo G. Martins and Tony Gorschek. Requirements Engineering for Safety-critical Systems: A Systematic Literature Review. *Information and Software Technology*, 75:71 – 89, 2016.

[24] George A Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[25] Jamal Ouhalla. *Functional Categories and Parametric Variation*. Routledge, 2003.

[26] Martha Palmer. Semlink: Linking Propbank, Verbnet and Framenet. In *Proceedings of the Generative Lexicon Conference*, pages 9–15. Italy, 2009.

[27] Terence Parsons. *Events in the Semantics of English*, volume 5. Cambridge, Ma: MIT Press, 1990.

[28] Terence Parsons. Thematic Relations and Arguments. *Linguistic Inquiry*, pages 635–662, 1995.

[29] Amalinda Post, Igor Menzel, Jochen Hoenicke, and Andreas Podelski. Automotive Behavioral Requirements Expressed in a Specification Pattern System: a Case Study at BOSCH. *Requirements Engineering*, 17(1):19–33, 2012.

[30] Alan Rector and Jeremy Rogers. Ontological and Practical Issues in using a Description Logic to Represent Medical Concept Systems: Experience from GALEN. In *Reasoning Web*, pages 197–231. Springer, 2006.

[31] H. B. Reubenstein and R. C. Waters. The Requirements Apprentice: Automated Assistance for Requirements Acquisition. *IEEE Transactions on Software Engineering*, 17(3):226–240, Mar 1991.

[32] Michael Roth and Ewan Klein. Parsing Software Requirements with an Ontology-based Semantic Role Labeler. *Language and Ontologies*, 15, 2015.

[33] Karin Kipper Schuler. VerbNet: A Broad-coverage, Comprehensive Verb Lexicon. *Dissertations available from ProQuest. AAI3179808*, 2005.

[34] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *OWL: Experiences and Directions*, volume 432, page 91, 2008.

[35] D. Zowghi, V. Gervasi, and A. McRae. Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements. In *Proceedings Eighth Asia-Pacific Software Engineering Conference*, pages 133–140, Dec 2001.