Advanced Engineering Informatics 30 (2016) 77–94

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

An ontological framework for knowledge modeling and decision support in cyber-physical systems

Leonard Petnga [a,*], Mark Austin [b]

[a] Department of Civil and Environmental Engineering, University of Maryland, College Park, MD 20742, USA
[b] Department of Civil and Environmental Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA

ARTICLE INFO

Article history:
Received 18 August 2014
Received in revised form 31 December 2014
Accepted 30 December 2015

Keywords:
Ontologies
Cyber-physical systems
Reasoning
Decision making
Artificial intelligence
Semantic Web

ABSTRACT

Our work is concerned with the development of knowledge structures to support correct-by-design cyber-physical systems (CPS). This class of systems is defined by a tight integration of software and physical processes, the need to satisfy stringent constraints on performance and safety, and a reliance on automation for the management of system functionality and decision making. To assure correctness of functionality with respect to requirements, there is a strong need for system models to account for semantics of the domains involved. This paper introduces a new ontological-based knowledge and reasoning framework for decision support for CPS. It enables the development of determinate, provable and executable CPS models supported by sound semantics strengthening the model-driven approach to CPS design. An investigation into the structure of basic description logics (DL) has identified the needed semantic extensions to enable the web ontology language (OWL) as the ontological language for our framework. The $\mathcal{SROIQ}$ DL has been found to be the most appropriate logic-based knowledge formalism as it maps to OWL 2 and ensures its decidability. Thus, correct, stable, complete and terminating reasoning algorithms are guaranteed with this $\mathcal{SROIQ}$-backed language. The framework takes advantage of the commonality of data and information processing in the different domains involved to overcome the barrier of heterogeneity of domains and physics in CPS. Rules-based reasoning processes are employed. The framework provides interfaces for semantic extensions and computational support, including the ability to handle quantities for which dimensions and units are semantic parameters in the physical world. Together, these capabilities enable the conversion of data to knowledge and their effective use for efficient decision making and the study of system-level properties, especially safety. We exercise these concepts in a traffic light time-based reasoning system.

1. Introduction

Cyber-physical systems (CPS) are defined by the integration of physical systems with sophisticated (highly automated, autonomous, multi-agent) computation and networking. Embedded computers and networks are tasked with monitoring and controlling the physical processes, usually with feedback loops where computation affects physical processes, and vice versa [1,2]. Early applications of CPS can now be found in a variety of industries, including energy-efficient buildings, air and ground transportation, healthcare and manufacturing. The disruptive and transformative potential of these applications have led governmental entities and researchers to position CPS as the next technological revolution that will equal and possibly surpass the Internet [3,4].

The long-term expectation of CPS design is that engineers will be provided with methods and mechanisms to create systems that will always work correctly, and will operate with superior levels of performance, reliability and safety. Perhaps CPS will achieve these purposes through the use of new architectural models that redefine form and function? At this time, however, the full potential of this opportunity is hindered by the lack of a mature science to support systems engineering (i.e., definition, composition, integration) of high-confidence CPS. Capturing and analyzing CPS behaviors in formal models, even minimal ones, is cumbersome. Present-day procedures for the engineering of CPS's are weakened by the presence of non-determinate models, weak temporal semantics, coupled with the high sensitivity of CPS to timing [5]. For CPS applications that are safety critical, this is a problem because notions of design correctness will correspond to the satisfaction of physical world constraints and, in turn, their dependency on formal models of time and space.

* Corresponding author.
E-mail addresses: lpetnga@umd.edu (L. Petnga), austin@isr.umd.edu (M. Austin).

http://dx.doi.org/10.1016/j.aei.2015.12.003
1474-0346/

This paper takes a first step toward mitigating these deficiencies. We lay down the foundational building blocks to support the development of determinate CPS models, with strong temporal and domain-specific semantics strengthening model-driven approaches to CPS design. Our focus will be on the data and information processing layer of CPS modeling, with a particular attention to procedures and mechanisms for producing determinate, provable and executable CPS models. We introduce and describe an innovative ontological framework, and illustrate the structure and phases of construction for a knowledge modeling and decision support framework for CPS (CPS-KMoDS). The framework offers some flexibility in its implementation, for example, for the selection of tools and type of tasks targeted by the model. System dependability characteristics, especially safety, are viewed as multi-domain models that drive the evaluation of decision tasks and, as such, development of the ontological framework.

The paper is organized into six sections. Section 2 briefly presents CPS and semantics challenges in modeling such systems as well as key requirements for the CPS-KMoDS. Section 3, along with Appendices A–C, provides a summary background on the mathematical foundations supporting the framework with an emphasis on description logics (DL) and their central role in supporting reasoning tasks. In Section 4 the proposed framework is introduced and its construction process described. A Jena-based implementation of the framework is presented in Section 5. We exercise the framework through the development of a time-based reasoning system to support decision making for cars passing through a traffic intersection controlled by traffic lights. The paper concludes with a summary and suggestions for future work.

2. CPS knowledge modeling and ontologies

2.1. CPS: overview and key characteristics

An examination of CPS application domains (e.g., aerospace, healthcare, transportation, energy or automotive) reveals components that span multiple physics and engineering domains, operate across multiple time scales, and have dynamics that are sometimes affected by human-in-the-loop interactions. Thus, we can categorize CPS components as follows [6]:

(a) **Cyber components:** These are computation, control and communication platforms, each implementing some specific system function. Given their software (or cyber) nature, these components need a physical (or hardware) platform to run the corresponding program, to support communication among cyber components and with the surrounding environment.
(b) **Physical components:** They act as facilitators for physical interactions as well as implementation of functional specifications for the system. Generally speaking, physical component complexity increases when components cover multiple engineering domains, and when components embed computational capability. Examples of the latter include onboard computers in automobiles, unmanned aerial vehicles (UAV), smart sensors in bridges, and smart medical implants.

Fig. 1 shows the network structure and components in a prototypical CPS. The system is made of four integrated and networked platforms with a physical plant. A network (wireless in this case) allows the various platform to communicate with each others. This network could be as small as a Local Area Network (LAN) or as big as the Internet. Some of the links between the platforms are direct and would not go through the wireless network. One of the platforms (#4) is embedded in the physical plant which interacts with the cyber world through physical interfaces. Each platform is made of all or some of the following components:

1. **Computation module:** Computation modules process plant data collected by sensors and/or output from other platforms. System architectures may impose dependency relationships among computation modules, independently on their location. For our illustrative example (see Fig. 1), this capability allows physical processes occurring in the plant to affect or modify computations in platform #2 using both the embedded platform (#4) and the wireless network to communicate with platform #2.
2. **Sensors:** Sensors collect plant data (physical measurements) and pass them to the computation module for further processing. For example, sensors are illustrated on platforms #1 and #4. They usually operate as a node in a sensor network architecture.
3. **Actuators:** They intervene in the feedback control loop of the plant to control mechanisms or processes according to the system specifications. Platform #3 illustrates one of them.
4. **Interfaces:** Network interfaces allow for the flow of data between platforms directly or through a network. Physical interfaces allow for plant and platform connectivity. In Fig. 1, all platforms are equipped with both types of interfaces except for platform #2, which has only network interfaces.

2.2. Semantic challenges in CPS modeling and analysis

The design and realization of a CPS satisfying even a small subset of the architecture shown in Fig. 1 is challenging. Difficulties in development stem from a variety of sources including the need to deal with a multiplicity of physics and engineering disciplines, each requiring expertise. Lee [8] illustrates this complexity using a subset of an aircraft electrical power system (EPS). Depending on the domain-specific viewpoint, the perception of the system can range from a software to an electrical system passing by a mechanical, control or communication network. This leads to multiple domain-specific models of the CPS, with none of them covering the CPS entirely. In a slightly different take on strategies to address challenges for CPS development, Sztipanovits [6] explains this complexity through the observation that, often, the behavior of physical components in CPS is defined by interactions among multiple physics that are difficult to capture in a single model. Thus, the CPS designer will face the challenge of composition of multi-models for heterogeneous physical systems.

To complicate matters, modeling challenges seem even harder when the subject of investigation covers CPS model semantics. Doyle [9] observes that theories backing the various disciplines involved in CPS are "deep but fragmented, incoherent and incomplete." The landscape of theories span from Turing and Von Neumann for computation to Einstein, Carnot or Newton for system physics through Nash and Bode in control or Shannon in communication domain. Fig. 2 illustrates this complexity and a view of some of the key challenges in the context of CPS modeling. Various domains involved in the modeling and design effort are orthogonally mapped to the main models abstraction layers.

Addressing semantic challenges: Some researchers have investigated ways to address these challenges with mixed success. In Derler [5], a landscape of technologies ranging from hybrid systems modeling and simulation to concurrent and heterogeneous models of computation (MoC) is presented. The use of MoC in Ptolemy II [10] is possible thanks to well-defined semantics for concurrency and communication between actor-oriented component models. However, despite its many computational advantages, the use of superdense time models [11,12] for timing is not

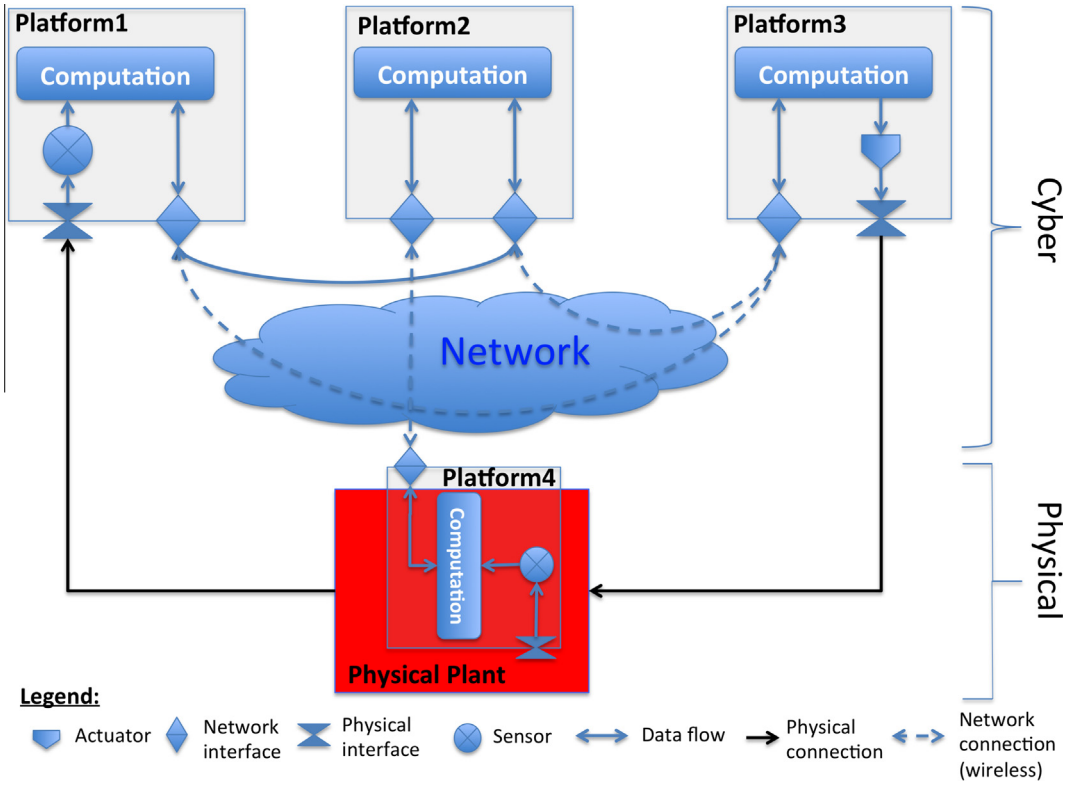

Fig. 1. Example schematic of a CPS (Adapted from [7]).

intuitive for system modeling. Jensen [13] builds on these foundations to propose a step-by-step methodology for model-based design of CPS. This contribution addresses challenges in development of the aforementioned abstraction layers, but there is no explicit mention on how to handle non-functional requirements in the broken chain of models produced by the design process. Bhave [14] proposes an architectural-based approach centered on an "architectural view" that encapsulates structural and semantic correspondences between the model elements and system entities represented at multiple layers of abstraction (physical, control, software, hardware). While the mapping between various views enhances reliance of the run-time base architecture, the underlying process remains manual and error prone, especially as the size and complexity of a system grows.

Critical role of safety in CPS development: It is evident from Fig. 2 that system safety properties are critical at all abstraction layers, which makes it a permanent concern for any CPS designer. Due to the presence of physical-related elements and concepts in the physical and platform abstraction layers, both are obvious subjects of safety concerns. One way that safety concerns can become an issue in the software abstraction layer is through deadlocks, which in turn can lead to unsafe system configurations. The rationale here is that timing (from the physical world) in models at this abstraction layer is not a simple performance or quality factor for the software but a design correctness criterion. As a result, the determination as to whether or not the system operates safely at any point in time requires consideration of all of the relevant aspects across the participating domains, physics, and abstraction layers. Fortunately, all CPS share some commonality in the ways they process information [1]. We illustrate this in Fig. 2 as the so-called "commonality of information" that crosses all domains and abstraction layers of the system design. The basic idea is to design a data structure that encapsulates the relevant knowledge of the CPS of interest while providing the foundation for meaningful construction of models. We would like to provide a mean to structure, organize and formalize that knowledge, and address the challenge of modeling aspects of the system response related to the evaluation of non-functional and safety requirements. The premise here is that these safety properties and non-functional requirements can be formulated as decision problems with true/-false or yes/no solutions.

2.3. Requirements on CPS models for decision making

By definition a model is a simplified abstraction of another entity [1,15]. From a MBSE perspective, models are designed to allow for the quantitative evaluation of questions that can be traced back to designer concerns (e.g., correctness of system functionality; adequacy of performance; assurance of safety). A well-designed model contains just enough detail to answer the relevant questions and nothing more. For the purposes of this research, the main task at hand is support for decision making, which, in turn, drives the need for the development of models that are determinate, provable and executable. The details are as follows.

1. **Determinate:** A model is determinate if it provides answers to questions that are certain and conclusive. For the design of CPS, it is well known that physical processes are not determinate. Similarly, on the cyber side of development, the use of threads as a dominant sequential model of computation to concurrency results in models that are non-determinate [16]. The long-term challenge is to counter these realities by "dynamically changing" programming models so that their correct execution always produces acceptable behaviors at subsystem I/O [8]. This capability will ease the modeling, simulation and verification of non-functional requirements and dependability properties with

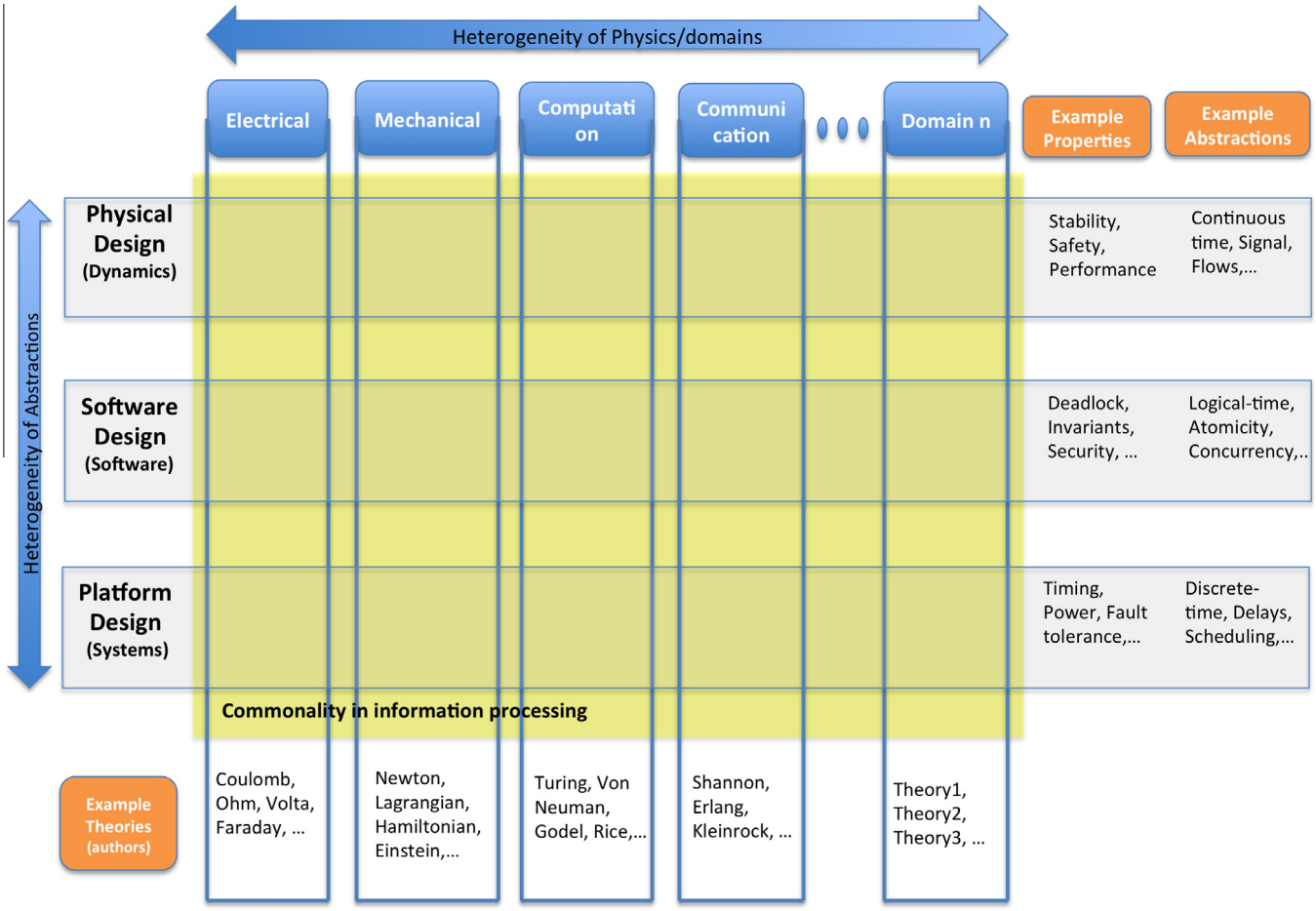

Fig. 2. Complexity and challenges in CPS Modeling (Adapted from [6]).

safety as one of the most important. Given the restrictions and intrinsic weaknesses of computer systems, this is not an easy task [17]. However, we ought to be able to start by producing well-defined, determinate models and progressively move toward stochastic ones along with ways to deal with uncertainties.

2. **Provable:** A model is said to be provable if it has the capability to establish the validity (or truth) of assumptions. For the design of CPS, the development of provable models is complicated by the heterogeneity of physics, domains, and abstractions emanating from different types of models. Still, with safety at the heart of system characteristics, the precise meaning of models is required; thus, the need for formal semantics and formal descriptions of models that keep unambiguity away.
3. **Executable:** For our purposes, a model is said to be executable if it is formal enough to be processed by a machine. Complicating factors include data and information emanating from multiple distinctive sources, and the need for evaluation of system behaviors that are dependent on multiple physics and multiple abstractions. See Fig. 2. From this perspective, the CPS model will be similar to a computer program that provides a precise and concise description on how data can be cast into a representation to support decision making. For this process to work well, the underlying modeling language should be decidable in the sense that the designer should be able to automatically determine model correctness and the point of program termination. Unfortunately, standard languages such as Fortran, C, C++ or Java are not decidable, as demonstrated by the unsolvability of the Halting problem [18,19]. Therefore, to move forward, some restrictions are needed to achieve decidability of a problem formulation casts in one of these languages.

2.4. Ontologies for Model Based System Engineering (MBSE) of CPS

Now that the engineering community is recognizing the benefits of model-centric approaches to system development, increasingly computer-based models are becoming the authoritative source of information to drive design processes forward. In order to support decision making in team-based development, the associated frameworks need to be represented with modeling languages that have formal semantics. Thus, as a pattern for what constitutes a system (i.e., parts, connections, identity, dependence, etc.), system ontologies provide a strong foundation for modeling and analysis of the participating domains that includes meaning of terms and validation of model correctness [20].

Given the breadth and complexity of CPS applications, in our opinion, the search for a "unique foundational ontology for CPS" is not likely to produce fruitful results. Some researchers have oriented their work toward supporting the development of CAD and CAE based tools and their semantic interoperability [21,22]. Others adopt a manufacturing and product-based focus to their ontologies, framework or language development and use [23,24]. A variety of studies have highlighted the key role of ontologies and Semantic Web technologies in heterogenous engineering knowledge capture and formalization within the context of knowledge based engineering efforts [25], and systems integration in the wide field of construction [26].

Despite the numerous benefits of these framework, the models that they produce lack several of the properties and characteristics needed to satisfy the requirements identified in Section 2.3. Weaknesses include the lack of support for the physical aspects of CPS and ontological modeling of time, which is one of its critical meta-domains. The CPS-KMoDS framework that we introduce in this paper aims to mitigate these gaps with the use of models and tools based upon ontological and logic-based mathematical foundations.

3. Description logic semantics and support for reasoning

This section introduces formalisms for the capture and representation of knowledge, suitable for decision making support in CPS development. Traditional approaches to knowledge representation and reasoning stem from artificial intelligence (AI) and classical logics, which may or may not be decidable. Our focus is on methods that are computationally decidable. As such, we make extensive use of description logics (DL) and its various extensions.

3.1. Knowledge representation formalisms

In their effort to formally capture and represent domains knowledge, researchers have developed several knowledge representation formalisms such as Semantic Networks [27–29], Frame Systems [30–32], Description Graphs [33–35] and Logic-based formalisms [36]. The latter, and especially the declarative part of frame systems, have played a central role in the evolution of AI formalisms. Feature logics were developed as the constraint logic part of so-called unification grammars, such as head-driven phrase structure grammar (HPSG). For details, see http://hpsg.stanford.edu/. Modal and description logics (DL) appear to be the most appealing logic-based formalisms for framework like ours. In fact, some results for description logics were found by translating results from variations of modal logics (propositional dynamic logics, μ-calculus) into description logics [37,38]. For the purpose of this work we'll be using description logics and its multi-values attributes capability. A brief definition of key DL concepts and its $\mathcal{ALC}$ extension are introduced in Appendix A.

3.2. Description logics extensions for the web ontology language (OWL)

In order to build models that address the challenges identified in Section 2.2 and satisfy the requirements of Section 2.3, CPS applications need to be backed by ontologies that have well-defined semantics and support for formal reasoning. DLs provide these formal foundations to the web ontology language (OWL) [39,40]. In fact, the semantics of the OWL language can be defined through a translation into an expressive DL. However, as pointed out by Baader and co-workers [41], the $\mathcal{ALC}$ extensions (see Appendix A) are incapable of efficiently supporting OWL because important pieces are missing. Bridging this gap requires a certain number extensions including support for role Hierarchy ($\mathcal{H}$), Nominals ($\mathcal{O}$), Inverse and transitive roles ($\mathcal{I}$), Cardinality/Number restriction ($\mathcal{N}$), Qualified number restrictions ($\mathcal{Q}$), Role restrictions ($\mathcal{R}$) and Concrete domains. These extensions are briefly defined along with illustrative examples in Appendix B.

Fig. 3 shows how these extensions to $\mathcal{ALC}$ DL can be organized and mapped to semantics for the OWL sub-languages. To that extend, OWL 2 (standardized in 2009) overcomes a number of weaknesses (e.g., relational expressivity, syntax deficiencies, species definitions) in OWL 1 [42]. Tapping into this potential for efficient modeling and decision support for CPS-based applications requires effective and decidable reasoning capabilities as enablers. We briefly introduce in the next section the reasoning infrastructure needed to that aim.

3.3. Reasoning support for $\mathcal{SROIQ}$-based ontologies

When the relevant set of axioms are applied to a specific DL-based ontology, the result is a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ for the domain being modeled. However, this is half of what we need for our framework. This foundation needs to be completed with a reasoner that can derive, through inferencing, additional facts about the concepts of the domain of interest. Among the key reasoning services needed, are satisfiability, subsumption, equivalence and disjointness. These services are formally defined in Appendix C. Also, with regard to the $\mathcal{SROIQ}$-DL which is mapped to OWL 2 (see Fig. 3) there is a need to formally establish the decidability of this DL. Thus, Proposition 1 builds on the definitions introduced to establish the satisfiability of the TBox while Lemma 1 ensures the elimination of the ABox for the purpose of simplifying the complexity of the reasoning process. Horrocks and co-authors [43] use these preliminary results to construct and describe an algorithm that decides the satisfiability and subsumption of $\mathcal{SROIQ}$ as stated by Theorem 2. Hence, given the mapping in Fig. 3, this theorem ensures the decidability of OWL 2 DL, the language of development of our ontological framework that we introduce in the next section.

4. Framework for modeling CPS knowledge and reasoning support

4.1. System overview

The global context for the design and construction of this framework is knowledge-enabled models for complex heterogeneous systems, such as CPS. The pathway for this task involves many steps including automated data acquisition, transformation of data to knowledge, and finally the creation of models that are reusable, provable and executable. The first potential use of these models is for system behavior and safety analysis. They can also act as middleware for CPS systems. To achieve these purposes, the CPS-KMoDS framework relies on the composition of domain-specific ontologies (DSO) along with corresponding knowledge bases (DSKB) on one hand and, domain-specific semantics extensions, an integrator and the CPS application on the other hand. The components of the framework are organized into layers as shown in Fig. 4. Thus, we describe in the next section the different layers of this architecture, and how the elements interact together to produce decidable CPS models with regard to the requirements identified in Section 2.3.

4.2. From data to knowledge: domain-specific ontologies and semantic support

The domains layer is a modular piece at the center of the CPS-KMoDS architecture. It covers the participating domains and disciplines (see the columns of Fig. 2), thus, represents concepts relevant to the CPS under study. To completely capture and represent the domain knowledge for CPS, we go beyond simple standalone ontologies toward an architectural structure spanning the bottom two layers of Fig. 4. The elements of these layers are: (1) domain-specific ontologies (DSO), (2) a data repository, and (3) semantic extensions and computation support. The details are as follows:

Domain-specific ontologies (DSO): Each domain is formally defined and described by a light, modular, and reusable basic ontology that captures its core concepts and properties. Then, it is extended with application-oriented concepts and properties.

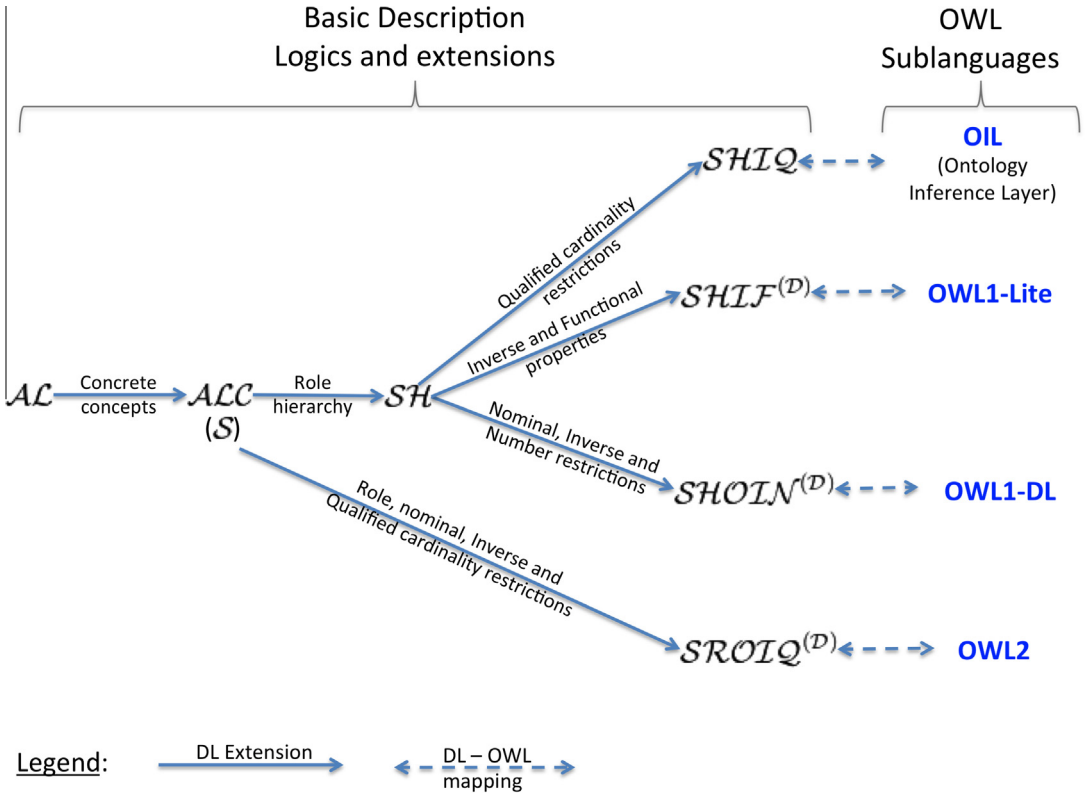

Fig. 3. Description Logics formalism extensions for the Web Ontology Language.

This approach to ontology specification is consistent with the TBox definitorial as introduced in Section 3.3 and formally defined in Appendix A. In the absence of instances these ontologies are reusable across applications. Laws and constraints of the domains are captured and translated as rules in domain-related *rules engines*. In order to provide support for complex computations and also to enforce semantics of a given domain, an *interface* to the relevant computational platform is needed by the reasoner.

Our framework employs three types of domain-specific ontologies:

1. Physical ontologies. These are ontologies of physical subsystems involved in the CPS of interest, for example, an automobile, a building or an aircraft.
2. Cyber ontologies. These ontologies describe the cyber part of the CPS are under this category. A software is an example of such domain.
3. Meta ontologies. Meta concepts such as time, space, or privacy that are relevant to the system are captured and described by this category. Because of their cross-cutting nature, they can apply to either the physical or cyber worlds, or both. In a previous work [44] we have demonstrated the critical importance of time and temporal semantics for MBSE of CPS.

Data repository: The data repository contains instances of the concepts defined in the ontologies. As the assertion component of the architecture, instances are interpreted as the ABox in the DL formalism. It is important to note that this interpretation operates under an "open world assumption" as opposed to a "close world assumption" of databases. Thus, the reasoner is prevented from drawing erroneous and invalid conclusions from the facts in the knowledge base. Control mechanisms embedded in the rules engine ensure that any data available in the repository is correct. These measures are particularly important, as CPS are safety critical systems and the decision made has to be the right one (always) in order to guarantee system safety.

Semantic extensions and computation support: As shown in Section 3.2, the $\mathcal{SROIQ}$ DL is equipped with appropriate formalisms to handle concrete domains – these are pre-defined interpretation domains for which semantics of datatypes are invariant (i.e., the same no mater the interpretation). Therefore, the development of this framework with an ontology language backed by this DL or equivalent will provide similar support.

Unfortunately, supported concrete sets such as real, integer or boolean that are computation friendly can miss essential information (e.g., dimensions and units) from the physics of the domain of interest. Support for reasoning with physical quantities can be made as needed, and made available to the reasoner through its interface. Hence, the corresponding computational platform will be able to process physical quantities-based datatypes. These orientations put our framework at the forefront of the efforts for a more "physicalization of the cyber world" in the sense of Lee [7].

4.3. From knowledge to model: system integration and CPS knowledge model build-up

The pathway from knowledge to models is defined by a systematic build-up of knowledge models from domain-specific ontologies. Ontologies from disparate domains need to be merged and integrated with ontologies that represent concepts from cross-cutting concerns, such as time.

Assembling ontologies: Domain-specific ontologies along with rules engine and semantic support are good foundations to domain-oriented system modeling, as described in the previous section. In the context of CPS modeling, however, stand alone formalizations of a sub-domain do nothing more than provide a

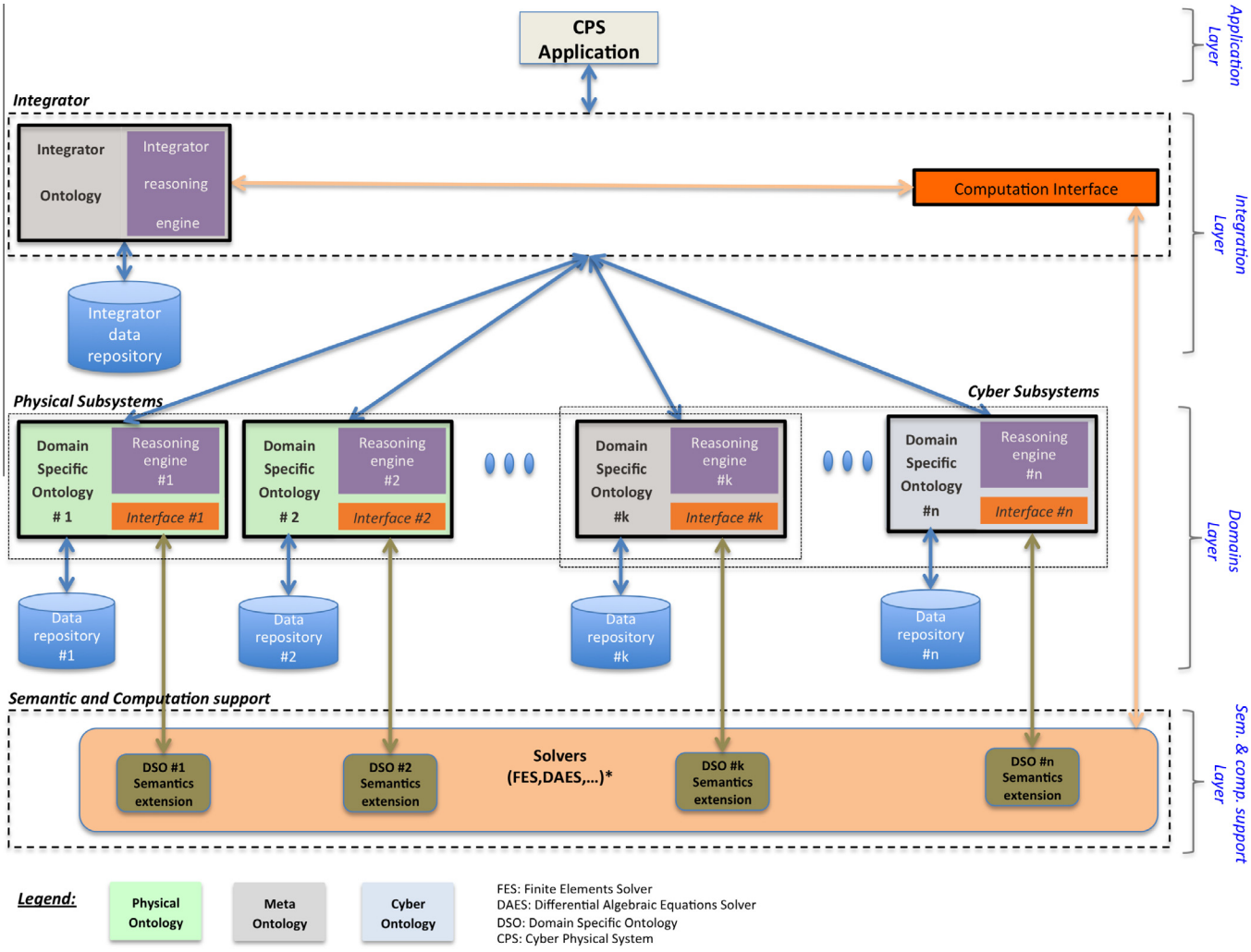

Fig. 4. Architecture of the CPS-KMoDS.

formal description of the domain and means for the designer to test proper low-level interactions between the different modules at the domain level. Even though the latter step is very important, it is not enough. There is a need to reuse the various domain ontologies in a coherent and correct assembly. This has to be done in a way that properly renders the CPS of interest while preserving the decidability of the underlining DL formalism. Several researchers [22,45] indicate that the following techniques provide a pathway for moving forward:

1. Merging: Ontologies for similar domains are merged into one single coherent ontology.
2. Alignment: Complementary domains ontologies are linked, resulting in two or more ontologies.
3. Integration: Ontologies from different domains are merged in one single ontology.

The categorization of DSO (as shown in Section 4.2) prevents the designer from introducing overlaps between ontologies during their development. This is not always guaranteed as concepts and properties can be repeated in different ontologies. Still, the CPS model has to be viewed as a unified domain, thus the need for a single ontology backing the model. This leaves us with "ontology integration" as the appropriate pathway for assembling individual ontologies in the CPS-KMoDS framework. The CPS ontology is created by merging all ontologies (including the integrator) under a single umbrella ontology that is checked for consistency before any further use.

Integrator ontology and extensions: The integrator is created to capture, represent and translate CPS properties and concepts that are not part of a specific subdomain. Concepts in this ontology are mostly from the individual DSO. The integrator has its own rules engine that translates the constraints and laws applicable to the CPS of interest. It also handles system metrics and control parameters, including decision rules capable of determining system safety state at any point in time. There is no need for semantic extension to support this ontology as it's not related to a specific concrete domain. Depending on the problem, one might need the system rules engine to interface with external solvers, for example, to handle complex calculations such as differential algebraic equations (DAE) or finite element analysis (FEA). A computation interface augmented with proper semantic translation capabilities is charged with linking both modules. However, the effectiveness of such computation platforms are dependent on the performance of the implementation hardware. This could significantly affect on-going decision tree exploration in the rules engine in a context where timing is a design correctness criterion, as seen in Section 2.2. For those cases where the granularity of modeled time

is not appropriate for the selected computation support platform, solvers can be replaced by set of lookup tables. These lookup tables will encode, with a high level of precision, solutions to the system's equations.

4.4. Reasoning for decision support

Reasoning is concerned with the use of inferencing techniques to draw conclusions from a set of premises. In the proposed framework, decision trees are translated into sets of logical rules that can be evaluated through the use of reasoning strategies.

Choosing a reasoning approach: Generally speaking, reasoning techniques can be of three types which are logical [46,47], heuristic [48,49], ethical [50,51]. Because of their weak underlying formalisms thus, high risk of undecidability, the last two approaches are weak are not good candidates for our framework. Therefore, moving forward, we only consider automated mathematical logic-based reasoning approaches, with a bias towards those enabling automated theorem proving. We would like every logical inference to be checked all the way back to the fundamental mathematical axioms in order to ensure model provability. Thus, some critical capabilities are needed to the rule-based reasoning approaches that we adopt for our CPS-KMoDS framework.

Rule-based reasoning for CPS-KMoDS: Rule-based reasoning is the classical approach to logic-based reasoning, where the knowledge-based system is developed by deduction, induction, abduction or choices from a starting set of data and rules. The main components of a rule-based reasoning system are: (1) the rule base, (2) the inference engine, and (3) a variety of miscellaneous integration components.

The rule base corresponds to a set of rules applicable to the (sub)system of interest. Rules are of the form of "*If…then…*" statements. Each rule is made of a *body* that contains all premises or conditions and a head that states the conclusion(s) when the conditions are satisfied. In CPS-KMoDS, rules are written and used in the following three ways:

1. Forward chaining (materialization): The rule base is scanned and heads are precomputed and stored. Conditions are evaluated one at a time, from left to right. The evaluation stops any time a condition is not satisfied and the rule is not fired. The CPS-KMoDS Integrator rules engine uses this method to hook to and access external lookup tables when needed.
2. Backward chaining (query-rewriting): The computation of the head of the rule is done on-demand with a minimal index storing. When the head of the rule is not called in an instance of an execution chain of the rule base, the given rule is not evaluated. This approach appears to be the most indicated for writing rules in our framework, especially when it's used as middleware.
3. Hybrid chaining: This approach combines the previous ones in complex rules designed to take full advantage of both methods.

Inference engine provides mechanisms supporting the use of the ontology language and allowing for additional facts to be inferred from available data in the repository and class definitions. Within the context of CPS-KMoDS, semantic reasoners are the actual concrete code objects that perform the inferencing task. The chosen ontology language for our framework is OWL, which is $\mathcal{SROIQ}$-backed and decidable, as shown in Section 3.2 and Appendix C. For practical applications of reasoning, there also exists numerous OWL DL reasoners, such as Pellet [52] and RACER, the Renamed ABox and Concept Expression Reasoner. They can be plugged into actual implementations of the CPS-KMoDS framework.

Miscellaneous components for the reasoning system include a temporary working memory to store information that is in-transit between different computation cycles, and connections to other parts of the framework. The latter are essentially internal links to TBox, ABox and possibly external connections to semantic extensions and computation support systems through interfaces (when needed), as shown in Fig. 4.

In our framework, reasoning engines are implemented and tested for each of the DSOs involved in the CPS of interest. They are integrated into a system rules engine along with the integrator rules engine. This operation mimics the above-mentioned ontologies integration process. Fig. 5 employs the architectural component shown in Fig. 4 and synthetize the development process for modeling and analysis of CPS behavior and properties in the CPS-KMoDS framework. The next section is a case study that exercises the key architectural elements of our framework.

5. Case study: A traffic light time-based reasoning system

5.1. Overview of the case and set up

The purposes of this case study are to: (1) show how the proposed CPS-KMoDS development chart in Fig. 5 can be used to build the architecture in Fig. 4, and (2) illustrate how the underlying reasoning structure can be used to support decision making and, consequently improve system level safety. Our focus will be on the domains layer of the architecture and its immediate parent (integration) and child (extension and computation support). This case study problem builds upon our previous work illustrating the implementation of the application layer, and the problem of a self-driving car traveling through a traffic intersection controlled by a smart traffic light system [44,53]. In the latter, vehicles (i.e., the physical system) interact with the light (i.e., the cyber system) with the objective of maximizing traffic throughput, while ensuring vehicle crossings are safe at the intersection. Each entity is equipped with its computation platform as per Fig. 1. This problem setup enables us to depart from the current traditional master–slave relationship between the vehicles and the light and move to a cooperative relationship enabled by a bidirectional communication between these entities. Also, we keep the physics of the vehicle simple (constant speed) and assume that computation times are insignificant enough to not affect the effectiveness of the decision process. The framework implementation employs Jena, a Java-based framework for the implementation of Semantic Web technologies that support the development of needed ontologies (i.e. car, light and time ontologies) as well as reasoning process. Let's formulate and analyze the problem domain as per the left-hand side of Fig. 5.

5.2. Dilemma zone problem, decision and reasoning problems formulation

Twilight zone. The "twilight zone" of a traffic intersection – also called the dilemma zone (DZ) – is the area where drivers are indecisive on whether to stop or cross at the onset of a yellow light. It claims around 2000 lives at stop light intersections and billions of dollars worth of damages in the United States every year [54]. Fig. 6 shows a simplified representation of the problem setup. The core safety requirement of the system car-light that should be valid all the times is that "No vehicle is allowed to cross the intersection when the light is red". This is a non-functional requirement, a hard constraint which violation is the driving force behind the multiple accidents at intersections. As shown on the Figure, the continuous dynamic of the vehicle (a) and discrete behavior of the light (b) illustrate the very different nature of both entities. This complicates the ability of the system to satisfy the safety requirement at the onset or in the presence of the yellow light.

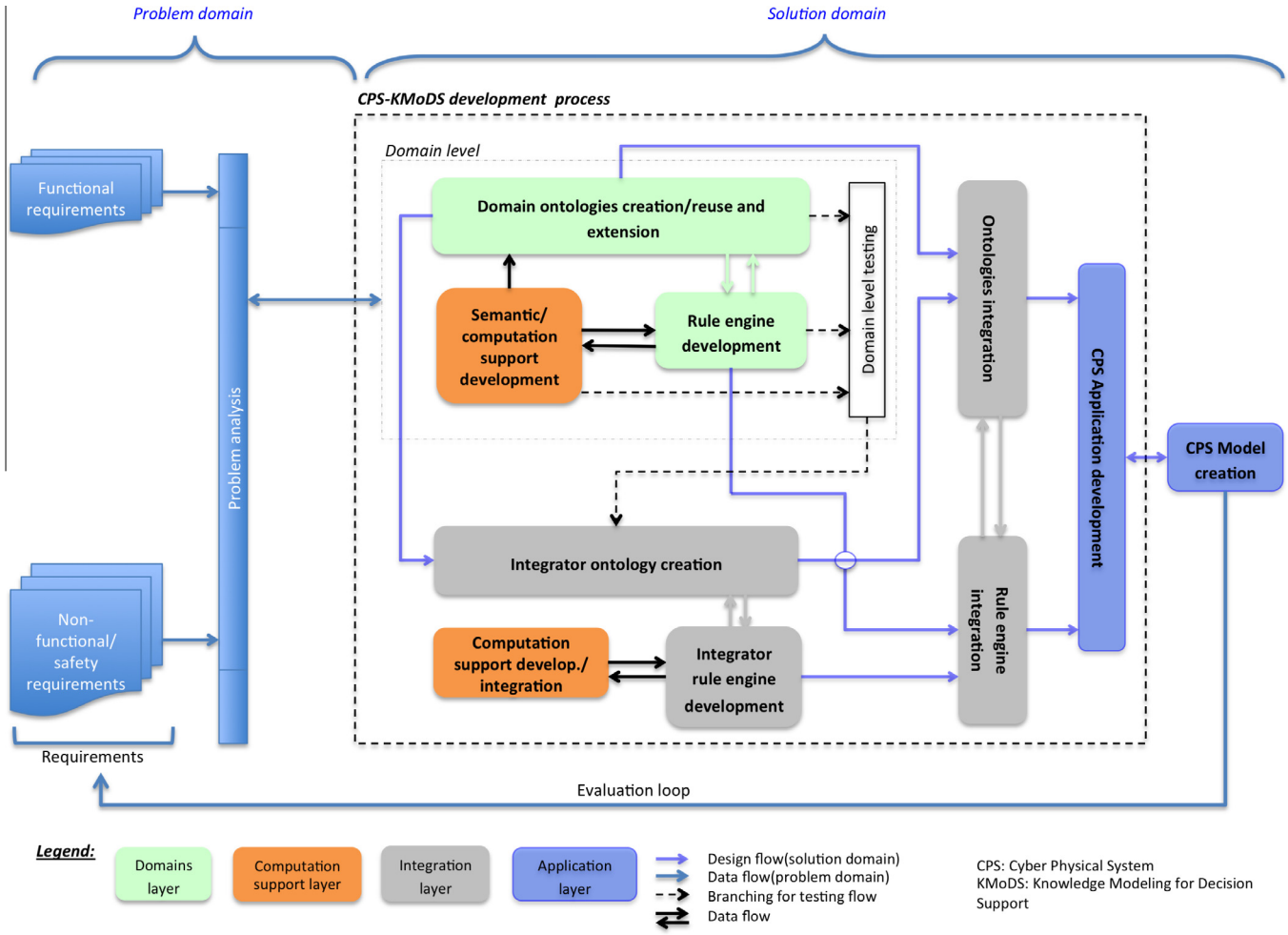

Fig. 5. Proposed flow chart for development of the CPS-KMoDS framework.

However, an analysis of the problem shows that there is a way forward.

Translating safety requirement satisfaction into a decision problem. Understanding the mechanisms by which system-level safety is achieved or violated is critical in addressing the dilemma zone challenge. Decision trees appear to be the most suitable analysis tool to explore the different possible paths the system could follow and characterizing the resulting state as safe or unsafe. Petnga and Austin [44] have shown that the probability of making the right course of action increases when the car has three key information at decision time: (1) Duration Θ_Y of the yellow light before it turns red; (2) Vehicle stopping distance XS, and (3) Travel duration Θ_B or distance to light XB. A smart car will be able to detect the light and accurately compute XS, Θ_B and XB on one hand, and take advantage of the bidirectional communication with the light to obtain Θ_Y from the stop light. Thus, it will be able to make a more informed decision as shown by the system decision tree (c) in Fig. 6. In [44], we show that, compared to actual human driven systems, this approach significantly improves system level safety and throughput by reducing the number of decision paths that lead to unsafe states. However, all paths of the decision tree still do not lead to good decisions as shown by the decision tree in Fig. 6. Thus, the system will not be 100% safe solely by relying on the smartness of the car.

Reasoning support to preventing unsafe system configurations. The configuration of the system for which there is no good decision despite the car smartness highlights the prominent role of the physics in the overall system safety. One illustration is the situation where the speed and/or condition of the vehicle along with the one of the road do not allow it to stop safely before the stop light or cross it before it turns red. Thus, the system will enter an unsafe state, the vehicle physics preventing the safety requirement from being satisfied. In such situations, we make use of the bidirectional relationship and reasoning capabilities of both entities (and an intermediary traffic supervisory controller) to resolve this issue before it materializes. If the traffic light learns that a vehicle cannot possibly pass through the intersection safely, it will reconfigure its operations for instance by lengthening the duration of the yellow light by just the amount of time needed ie $\Delta\Theta$ (determined by the supervisory controller) for the car to cross safely. The additional time will be taken from the duration of the red light in the same cycle, making its length unchanged. This will result in a safe crossing of the intersection as shown by The mechanics behind this reasoning process is rendered possible by the critical role of temporal semantics and Allen's temporal intervals [55] as we demonstrated in [44]. Further analyses (outside the scope of this work) identify four unsafe regions (I, II, III, IV) in the system decision space corresponding to situations where there is no good solution to the driveless car. We illustrate such situations in the context of a "Leader and Follower" configuration of the system in [56].

5.3. Jena-based modeling of the Traffic System as a CPS: system architecture

The implementation of our architectural framework makes use of Semantic Web technologies [39] and the Jena Application

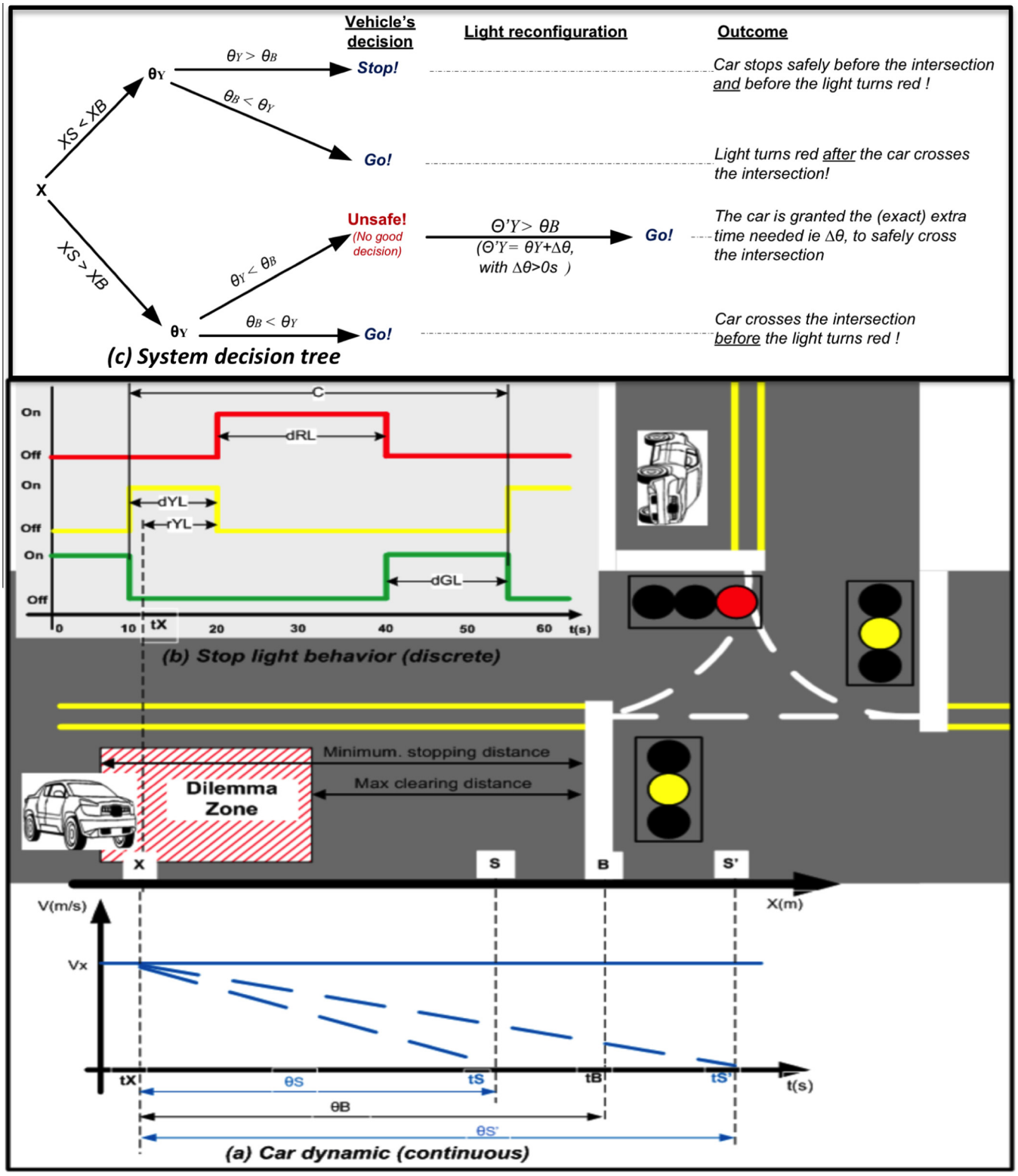

Fig. 6. Schematic of traffic intersection "dilemma zone" problem and corresponding simplified decision tree.

Programming Interface (API) [57]. The latter is a Java-based framework for the development of Semantic Web applications. As such, it provides a variety ofAPIs for accessing and handling standardized technologies such as the Resource Description Framework (RDF), triple stores and OWL platforms. Jena architectural framework supports the deployment of architectures that are consistent with the general architecture in Fig. 4. That's what the construction process of the solution domain (as per Fig. 5) of the dilemma zone problem does. The various layers of the CPS-KMoDS architecture in Fig. 4 are individually implemented and programmatically assembled bottom up as per the architecture using the capabilities of Jena API. In the next sections we describe each layer of the

CPS-KMoDS architecture for our traffic system example and their Jena-based assembly following the flow chart in Fig. 5.

5.4. Domains layer: light, car and time ontologies semantic blocks

From a CPS perspective and as suggested by the architecture, our traffic system model is partitioned into subdomains. We keep the space domain simple (reduced to a point), thus, there is no need for a separate ontological description for this meta domain for this application. For each of the three foundational subdomains (i.e., physical, meta and cyber), a corresponding domain specific ontology – car, time and light – is created along with domain rules. For instance, a car is defined in term of families such as #LightTruck, #SUV with properties like #hasWeight and #hasFinalDriveRatio. These properties are common to all types of car. Similarly, #hasColor and #hasCycleDuration are properties common to all stoplights. As for time, we employ a simplified version of the OWL-Time ontology [58] that is consistent with Allens temporal interval calculus. Concepts such as #Instant, #ProperTimeInterval and properties like #beginsAt and #intMeets serve as the foundation for the domain. Extensions are programmatically added using the Jena ontology API. This results in the development of subsystem ontologies that provide a better definition of the subsystem for efficient future use. For example, datatype properties such as #hasSpeed and #hasStoppingDistance are added to the car ontology because of their relevance to a formal quantification of the vehicle dynamics. This, in turn, is critical to the decision making strategies that solve the dilemma zone problem.

The excerpt below shows a description of the #intEquals object property in XML format. It formally defines the notion of equality between two entities of type #ProperTimeInterval.

```
<owl:ObjectProperty rdf:about="&time-entry;
  intEquals">
  <rdf:type rdf:resource="&owl;
  ReflexiveProperty"/>
  <rdf:type rdf:resource="&owl;
  SymmetricProperty"/>
  <rdfs:domain rdf:resource="&time-entry;
  ProperTimeInterval"/>
  <rdfs:range rdf:resource="&time-entry;
  ProperTimeInterval"/>
</owl:ObjectProperty>
```

A set of rules is created for each domain-specific ontology and encoded in the corresponding rules engine. For example, the fragment of code:

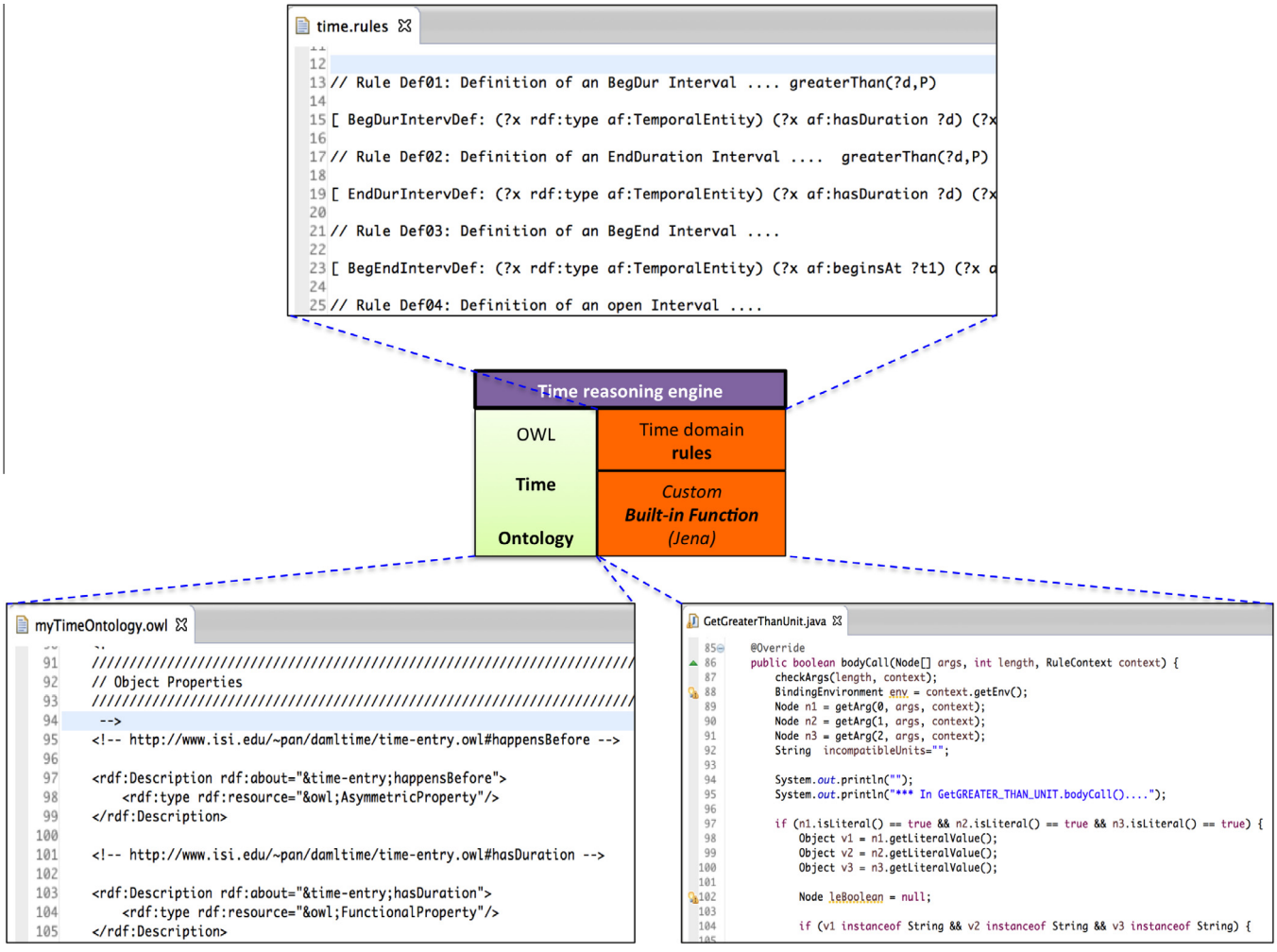

Fig. 7. Time reasoning engine architectural block and implementation.

```
// Rule #1: Propagate class hierarchy relationships
  ....
[rdfs01: (?x rdfs:subClassOf ?y), notEqual(?x,?y) ->
[(?a rdf:type ?y) <- (?a rdf:type ?x)]]
// Rule #2: Inferring an "Instant" from the
  definition
//          and property of a temporal entity...
[Instant: (?x rdf:type te:TemporalEntity) (?x te:
  hasTime ?t)
noValue(?x rdf:type te:Instant) -> (?x rdf:type te:
  Instant)]
// Rule #3: Duration of a "Proper" time interval ...
[GetDurationPropInterv: (?x rdf:type te:
  BegEndTimeInt)
(?x te:beginsAt ?t1) (?x te:endsAt ?t2) getDurInt
  (?t1,?t2,?d)
noValue(?x te:hasDuration ?d) -> (?x te:
  hasDuration ?d)]
```

shows how the Jena rules engine relies on hybrid and forward chaining techniques respectively to propagate relationships among classes in a hierarchy (#1), define an entity (#2), and compute and infer new statements, possibly with the help of built-in functions (#3). Fig. 7 shows the time reasoning engine architectural block and excerpts of the implementation of its various modules for our Dilemma Zone application.

5.5. Semantics support layer: handling of physical quantities and units

The framework enables the branching of semantics extensions to domains ontological structures wherever it's needed. In the case of this application there is a need for our reasoner to properly handle physical quantities. Dimensions (length and time) and units carried by data characterizing physical and meta properties such as #hasCarSpeed and #hasDuration in Car and Time ontologies. This is critical in keeping the reasoning and the ontologies consistent and unambiguous. Both flaws have the potential to lead to undecidable reasoning. To that aim, we use the Jscience [59] package to capture and handle the representation, conversion and computation of physical quantities related operations. This enables the reasoner to properly represent and distinguish, during processing and rule checking, both dimensions and units. These semantic services are provided to the reasoner by calls of Jscience functionality within custom built-ins functions where needed. Given the current inability of Jena to directly process dimensions and units, we wrap them into String datatypes as illustrated on the left-hand side of Fig. 8.

With this step completed we can proceed to "local" testing of individual domain level as per Fig. 5 by populating individual ontologies with valid instances. The verification of the proper integration of the Jscience and rules engine is of high interest here. A successful verification clears the path toward the integration of various blocks to form the integration layer for our traffic system.

5.6. Integration layer: integrator semantic block, control strategy and system level reasoning

Traffic system integrator. As shown in Fig. 8, the traffic system integrator defines relationships between subdomain entities. It's a meta sub-domain of the traffic system that cross-cuts the various cyber, physical and other meta domain making up the system and specifies crosscutting system-level properties. This includes properties related to the metrics used to help characterize the decision space. Its a separate ontology that simply uses elements of subsystem ontologies to enable a system-level view of the traffic system. As a case in point, the excerpt in Fig. 8 shows how Jena API is used to create and add a new ObjectProperty $\#hasTSCarInt_tXB$ to the Integrator ontology using elements of the Car ontology (#CarEntity) as domain and Time ontology (#ProperTimeInterval) as range. This property defines and associates a proper (close) time interval to the period of time that a car travels from location X (when the decision is being made) to the stop light (location B).

It is important to observe that the integrator operates like a traffic system "semantic controller" with its own rules engine encoding and enforcing system-level rules and calculations that affect the domains involved in the CPS behavior i.e. car, time and light. For instance, the excerpt:

```
[TimeSynchronization: (?x rdf:type ti:
  TSController)
(?y rdf:type le:CarLight) (?z rdf:type ce:
  CarEntity)
(?x ti:hasTSCurrentTime ?t) noValue(?y le:
  hasLightTime ?t)
noValue(?z ce:hasCarTime ?t)
-> (?y le:hasLightTime ?t) (?z ce:hasCarTime ?t)]
```

shows how the Integrator synchronizes time in the physical (car) and cyber (light) clocks.

Traffic system: model, control and reasoning strategies. The efficient reasoning on the system, as a whole requires the integration of the various ontologies. The overall traffic-system model is constructed from the merging of individual ontologies, including the Integrator. We opt for a dynamic import of ontologies to manage the stream of data in the system. Thus, domain and integrator ontologies are added to the empty traffic system ontology as submodels, with their top classes as disjoint subclasses of a #TrafficSystemEntity class. A system-level TS rules engine is constructed by way of union of domain rules engines in a unique file with integrator rules serving both as controller and systems integration glue. Its configuration mirrors the various branches of the system decision tree. A predefined Jena reasoner is used to perform inferencing because of its support for user-defined rules as well as forward, backward and hybrid chaining execution strategies. The integration of the units package Jscience with Jena thus as described in Section 5.5, enables the processing of physical quantities by the reasoner. This approach to construction of the TS model has the advantage of preserving the CPS view of the system while enabling deep insight in the connections and relationships between the domains. This is critical to uncovering and understanding mechanisms through which unsafe situations within the dilemma zone occur, while also providing support for efficient decision making.

5.7. Application layer: instantiation and testing the traffic system reasoning framework

In order to evaluate the effectiveness of the traffic system framework, we test it as a stand alone platform. We instantiate the ontological structure by populating the system with car and light entities and minimal data characterizing their basic properties. We are particularly interested in configurations of the system for which it reaches one of the four unsafe states. We verify that the reasoner is able to accurately: (1) predict this occurrence, and (2) reconfigure itself (actually the light) to enable safe crossing of the intersection when the car doesn't have a viable solution

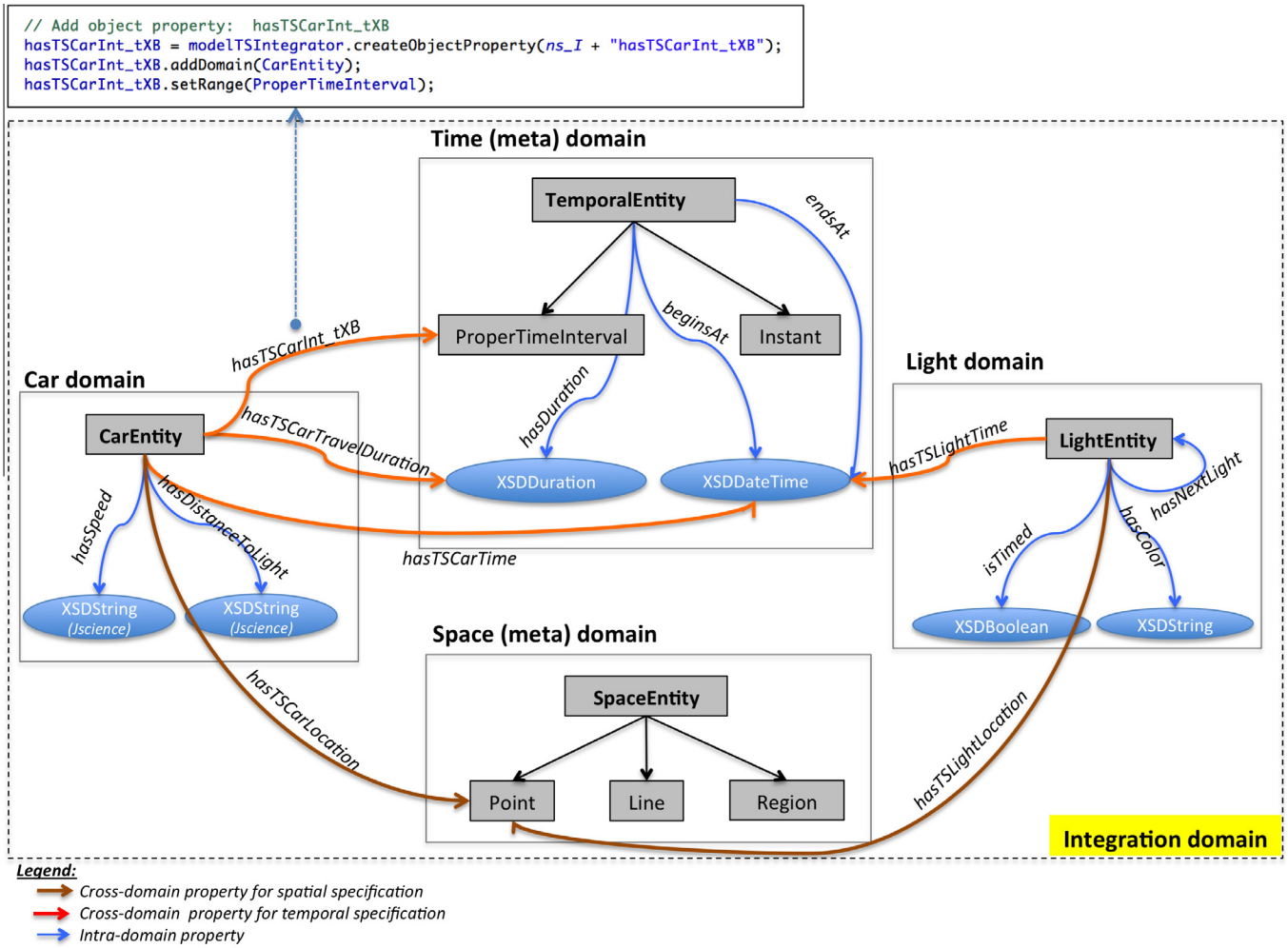

Fig. 8. Illustration of the construction mechanism of the traffic system integrator ontology.

(NO_GOOD). To exercise the system, we pick a #2004FordTaurusSES (Sedan) weighing around 1.5 ton and approaching an intersection at 30 m/s. The remaining duration of the yellow light at the time the decision is taken is $r_{YL} = 9$ s on a total duration of $d_{YL} = 15$ s. Combined with other parameters (e.g., stopping distance, braking force, other lights durations, etc.), the traffic system reasoner is able to infer that the vehicle system will enter an unsafe state, region IV in this case. The screen capture in Fig. 9 shows how the traffic system controller improves decision making in the dilemma zone by allocating extra time i.e., $\Delta\Theta = 2$ s in the length of the yellow light, which is the time needed by the car to cross the intersection safely. The new system metrics are calculated to account for the change and ensure the integrity of the duration of the cycle of the stop light. Therefore, the car is no longer projected to violate the red light when it reaches the intersection, it's now in region V which is a safe spot in the decision space.

6. Conclusions and future work

In this paper we have introduced a new ontological-based knowledge and reasoning framework for decision support for CPS. The framework enables the development of determinate CPS models with metadomain (e.g., temporal, spatial, etc.) and domain-specific semantics strengthening the model-driven approach to CPS design. Key features of the framework include: (1) a common data and information processing layer, which helps in overcoming the barrier of heterogeneity of domains and physics in CPS, and (2) the use of logic-based Description Logics (DL) semantics and rules-based reasoning. Together, these capabilities enable the conversion of data to knowledge, their effective use for efficient decision making and the realization of system-level properties such as safety. An investigation into the structure of basic DLs has led us to the identification of semantic extensions needed to efficiently support the web ontology language (OWL) as the ontological language for our framework. The resulting DL ie $\mathcal{SROIQ}$ has been found to be the most appropriate as it maps to OWL 2 and ensures its decidability. Thus, the formal description of the knowledge system for the CPS of interest as well as correct, stable, complete and terminating reasoning algorithms are guaranteed with this $\mathcal{SROIQ}$-backed language. Also, the framework is modular and, thus, provides the designer with flexibility and ease of configuration. Finally, it provides interfaces for semantic extensions and computational support, including the ability to handle quantities for which dimensions and units are semantic parameters in the physical world.

Future work needs to consider the development of a component modeling framework (front end) and its mapping to the ontological framework (back end) as systems engineers and modelers are not familiar with knowledge infrastructures such as the one introduced in this paper. Also, there is a need to develop a more general and inclusive approach to requirements elicitation that can account for safety concerns associated with human–machine interfaces, human reliability, delays, and failure mechanisms in software intensive systems, thus, broadening the scope of safety

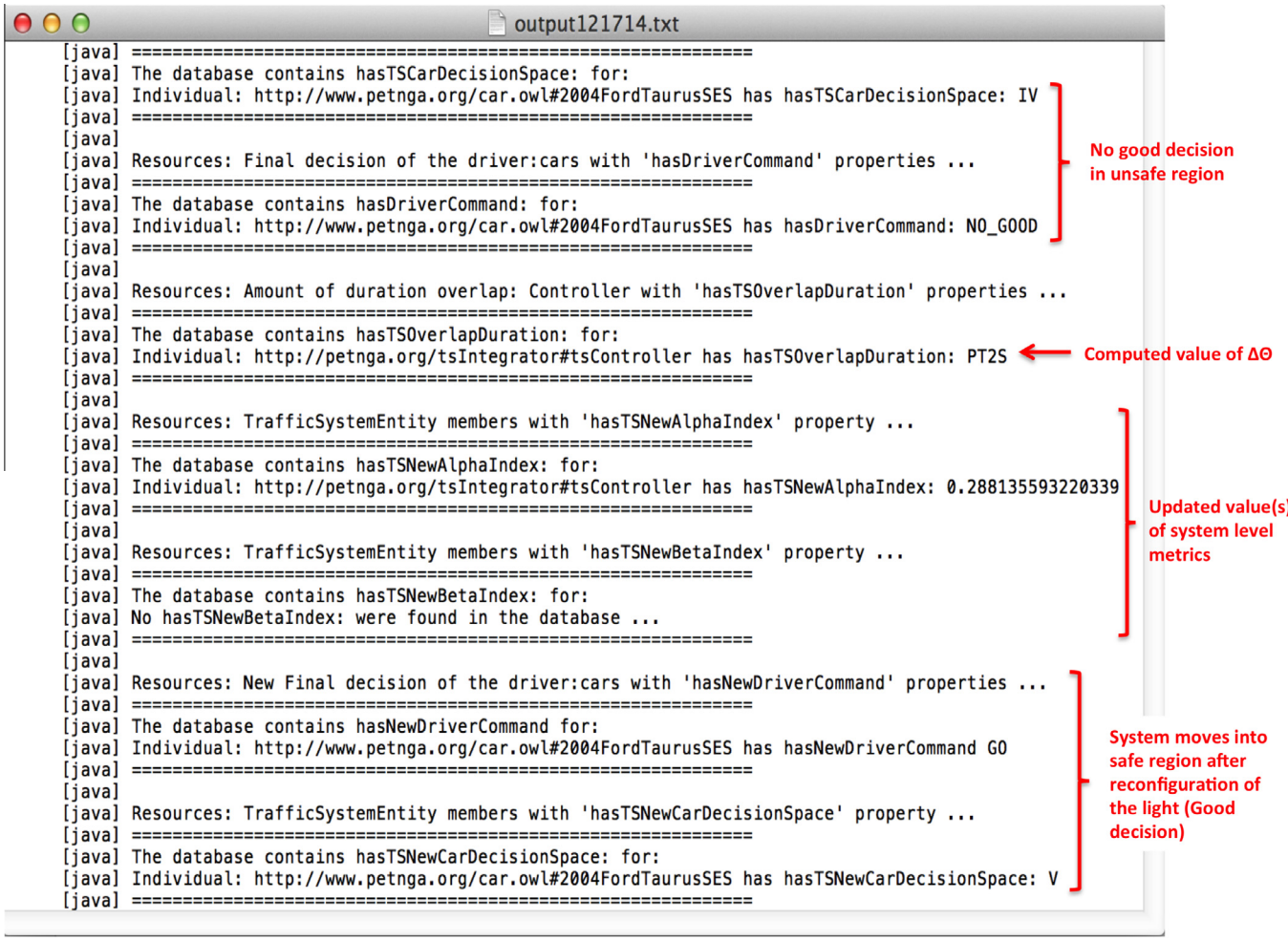

Fig. 9. Reconfiguration of the light to get the car out of an unsafe region.

requirements beyond a single one as we did in the case study. Systems-theoretic accident model and process and/or systems-theoretic process analysis (STAMP/STPA) [60] is a good supportive approach. We should also account for uncertainties in reasoning processes [61,62] especially data, reasoning mechanisms and timing uncertainties. Support for probabilistic reasoning [63,64], Fuzzy logic reasoning or truth maintenance [65] are possible pathways forward.

Acknowledgement

The work reported here is part of a US National Institute of Science and Technology (NIST) funded program dedicated to the development of standards for CPS design, modeling, construction, verification and validation.

Appendix A. Description logics and $\mathcal{ALC}$ extension

Basic description logics: Description logics are a family of logic-based knowledge representation formalisms that can describe domain in terms of concepts (e.g., classes in OWL), roles (e.g., properties, relationships) and individuals (e.g., objects). As a subset of first-order logics (FOL), they provide well-defined semantics supporting decidability and development of efficient reasoning algorithms. The acronym $\mathcal{AL}$ stands for attribute language (see Appendix 1 of Baader [36] for details on naming scheme for DLs). When a basic DL serves as a foundation for knowledge representation, many other DLs may be constructed through the addition of specific extensions. One such extension is the attribute language concepts ($\mathcal{ALC}$). The benefit of this extension mechanism is that is allows for the specification of languages supporting new features. For example, atomic concepts (A) can be extended to support arbitrary concepts (C), thereby enabling the description of any domain of interest. A second important extension is the number restriction N which leads to ALCN DL." This is a subset of the frame-based DL $\mathcal{FL}$ and is equivalent to $\mathcal{AL}$, but without atomic negation, inverse, transitive roles and subroles or concrete domains [66,67]. As we will soon see, these extensions and restrictions are needed to make the language decidable with low complexity, a strategy that is supported by Lutz [68], who identifies $\mathcal{ALC}$ as the most appropriate DL for reasoning with concrete domains.

The $\mathcal{ALC}$ description logics: In this DL, the operators universal ($\forall$), existential ($\exists$), intersection ($\sqcap$), Union ($\sqcup$), negation ($\neg$) can be properly applied to atomic (A, A_i,..), arbitrary (C, D,..), top ($\top$) (i.e., All concepts names) and bottom ($\bot$) (i.e., Empty concept) concepts. Primitive relations (r, s, ...) as well as existential restriction ($\exists r.C$) and value restriction ($\forall r.C$) on concepts are other key constructors used to formally define a domain of interest. The complete set of defined concepts of the basic $\mathcal{ALC}$ system can be represented by the following grammar:

$$C := \top | \bot | A | \neg C | C \sqcap D | C \sqcup D | \forall r.C | \exists r.C$$

For instance, the statement *"A woman who is single and whose children are either boy or girl"* can be expressed in DL using a minimal number of concepts as follows.

Human $\sqcap \neg$ Male $\sqcap \forall$hasChild.(Boy $\sqcup$ Girl).

In DL, semantics are defined by interpretations. In the case of $\mathcal{ALC}$, an interpretation $\mathcal{I}$ is formally defined as follows [69]:

Definition 1 (*Interpretation*). An interpretation $\mathcal{I} = (\triangle^{\mathcal{I}}, .^{\mathcal{I}})$ consists of a non-empty set $\triangle^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a function $.^{\mathcal{I}}$ that maps every $\mathcal{ALC} - concept$ to a subset of $\triangle^{\mathcal{I}}$, and every role name to a subset of $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ such that, for all $\mathcal{ALC} - concepts$ C, D and all role names r,

$$\top^{\mathcal{I}} = \triangle^{\mathcal{I}}, \bot^{\mathcal{I}} = \varnothing$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \neg C^{\mathcal{I}} = \triangle^{\mathcal{I}} \setminus C^{\mathcal{I}},$$
$$(\exists r.C)^{\mathcal{I}} = \{x \in \triangle^{\mathcal{I}} | \exists y \in \triangle^{\mathcal{I}} \text{ with } < x, y > \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$$
$$(\forall r.C)^{\mathcal{I}} = \{x \in \triangle^{\mathcal{I}} | \forall y \in \triangle^{\mathcal{I}}, \text{ if } < x, y > \in r^{\mathcal{I}}, \text{ then } y \in C^{\mathcal{I}}\}$$

x and y are *instances* of C in the interpretation $\mathcal{I}$.

Concept descriptions are used to build statements in a DL knowledge base in accordance to the semantics provided by the interpretation.

Fig. A.10 shows that the knowledge base (KB) is typically made up of two parts: (1) A terminological part or TBox, and (2) An assertional part called ABox.

Definition 2 (*TBox*). A TBox $\mathcal{T}$ is a finite set of general concept inclusion (GCI). A GCI is of the form $C \sqsubseteq D$ where C, D are $\mathcal{ALC} - concepts$. When $C \equiv D$ the corresponding pair of GCI $C \sqsubseteq D$ and $D \sqsubseteq C$ are symmetrical. If C is a concept name, then the axiom $C \equiv D$ is called a *definition*. An interpretation $\mathcal{I}$ is a model of a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if it is a model of every CGI in $\mathcal{T}$.

A TBox $\mathcal{T}$ can be definitorial (also called an acyclic TBox), i.e., it contains only definitions along with certain restrictions. In this case, concept names in left-hand side of $\mathcal{T}$ are "defined concepts" while concepts in the other side are "primitive concepts."

Definition 3 (*ABox*). An ABox $\mathcal{A}$ is a finite set of assertional axioms of the form x: C or (x,y): r, where C is an $\mathcal{ALC} - concept$, r is an $\mathcal{ALC} - role$, and x and y are individual names. An interpretation $\mathcal{I}$ is a model of an assertional axiom x: C if $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and $\mathcal{I}$ is a model of an assertional axiom (x,y): r if $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$; $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ if it is a model of every axiom in $\mathcal{A}$.

These definitions equip us with the necessary elements to formally define the notion of knowledge base introduced above.

Definition 4 (*Knowledge base*). A knowledge base (KB) is a pair $(\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a model of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if $\mathcal{I}$ is an interpretation of $\mathcal{T}$ and $\mathcal{I}$ is a model of $\mathcal{A}$.

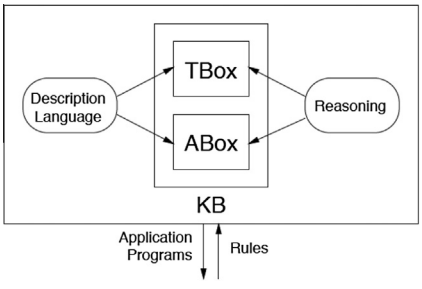

Fig. A.10. High level architecture of a knowledge representation system based on description logics. *Source:* [36].

We write $\mathcal{I} \models \mathcal{K}$, ($\mathcal{I} \models \mathcal{T}, \mathcal{I} \models \mathcal{A}$) to denote that $\mathcal{I}$ is a model of a KB $\mathcal{K}$ (respectively, TBox $\mathcal{T}$, ABox $\mathcal{A}$).

A summary of the main DL concept constructors is shown in Fig. A.11.

Appendix B. DL extensions for OWL2

1. **Role hierarchy ($\mathcal{H}$):** Hierarchies between roles are allowed in this extension. This results into the $\mathcal{ALCH}$ or $\mathcal{SH}$ DL formalism that is a translation of foundational OWL. In turn, there are three OWL sublanguages with increasing expressiveness: OWL-Lite, OWL-DL and OWL-Full (no syntactic constraints). More precisely, the DL TBox along with the role hierarchy extension map to the OWL (Lite or DL) ontology. In OWL, the domain of interest is defined in term of *classes* related to each other by *properties*. These entities correspond respectively to concepts and roles in $\mathcal{SH}$ DL.
   As an illustration, the DL statement *hasColor.CarColor* $\sqsubseteq$ *hasCarDescriptor* can be translated into OWL as follows:

```
<owl:ObjectProperty rdf:about="#hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasCarDe
  scriptor"/>
  <rdfs:range rdf:resource="#CarColor"/>
</owl:ObjectProperty>
```

The properties here are of type object, but they could also be of type data depending on the domain and application need.

2. **Nominal ($\mathcal{O}$):** In this DL extension, use of the nominal constructor {} allows for the definition of singleton sets (i.e., as concepts) from individual names. The corresponding restriction in OWL is achieved with the object property elements *owl:oneOf* and *owl:hasValue*.
   Let us suppose that we are given an "individual" V6. We can use this extension to define all cars that are equipped with this particular engine type as follows.

   $Car \sqcap \exists hasEngine.\{V6\}$

   This can be translated in OWL using the constructor *owl:hasValue* as follows.

```
<owl:Class rdf::about="#Engine">
 <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasEngine"/>
    <owl:hasValue rdf:resource="#V6"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

An important limitation of nominals [69] is that is can dramatically increase the complexity of reasoning processes.

3. **Inverse and transitive roles ($\mathcal{I}$):** This extension is needed to increase the expressiveness of the DL. Inverse and transitive roles are expressed in OWL using the object properties *owl:inverseOf* and *owl:TransitiveProperty*.
   For instance, $makeCar \equiv hasMaker^{-}$ and can be expressed in OWL as follows.

```
<owl:ObjectProperty rdf:about="#makeCar">
  <owl:inverseOf rdf:resource="#hasMaker"/>
</owl:ObjectProperty>
```

Name	Syntax	Semantics	Symbol
Top	$\top$	$\Delta^{\mathcal{I}}$	$\mathcal{AL}$
Bottom	$\bot$	$\emptyset$	$\mathcal{AL}$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$\mathcal{AL}$
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\mathcal{C}$
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.\ (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	$\mathcal{AL}$
Existential quant.	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	$\mathcal{E}$
Unqualified number restriction	$\geqslant n\,R$ $\leqslant n\,R$ $= n\,R$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert = n\}$	$\mathcal{N}$
Qualified number restriction	$\geqslant n\,R.C$ $\leqslant n\,R.C$ $= n\,R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert = n\}$	$\mathcal{Q}$
Role-value-map	$R \subseteq S$ $R = S$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}$ $\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \leftrightarrow (a,b) \in S^{\mathcal{I}}\}$	
Agreement and disagreement	$u_1 \doteq u_2$ $u_1 \neq u_2$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}}.\ u_1^{\mathcal{I}}(a) = b = u_2^{\mathcal{I}}(a)\}$ $\{a \in \Delta^{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta^{\mathcal{I}}.\ u_1^{\mathcal{I}}(a) = b_1 \neq b_2 = u_2^{\mathcal{I}}(a)\}$	$\mathcal{F}$
Nominal	I	$I^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ with $\lvert I^{\mathcal{I}}\rvert = 1$	$\mathcal{O}$

Fig. A.11. Summary of description logic concepts constructors [36].

The procedure for expressing that an object property is transitive is as follows:

```
<owl:ObjectProperty rdf:about="#hasFollower">
  <rdf:type rdf:resource="#TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Car"/>
  <rdfs:range rdf:resource="#Car"/>
</owl:ObjectProperty>
```

4. **Cardinality/number restriction ($\mathcal{N}$):** This extension allows for the formal expression of the number of relationships that individuals of specific types can have among them, a feature that is particularly relevant to CPS modeling.
   For example, the statement *A car has at most one engine* can be written as follows.

   $$Car \sqsubseteq \leqslant 1 hasEngine$$

   Additional syntax elements and their corresponding semantics are shown in Fig. A.11.
5. **Qualified number restrictions ($\mathcal{Q}$):** This extension is similar to the previous one with the difference that we can describe individual types that are counted by a given number of expressions, which allows for representation of the notion of a "data interval."
   To see how this works in practice, we can extend the definition of a car to allow for two through five doors. The corresponding logical expression is:

   $$Car \equiv Vehicle \sqcap \leqslant 1 hasEngine \sqcap (\geqslant 2 hasDoor \sqcap \leqslant 5 hasDoor)$$

6. **Role restrictions ($\mathcal{R}$):** This extension completes the $\mathcal{I}$ DL by providing role inclusion axioms as well as support for reflexivity, symmetry and roles disjointness. In OWL, these features show up as the property characteristics *owl:reflexive*, *owl:irreflexive*, *owl:symmetry*, *owl:functional* and *owl:disjointWith*. The fragment of code:

```
<owl:ObjectProperty rdf:about="#hasFollower">
  <rdf:type rdf:resource="#TransitiveProperty"/>
  <rdf:type rdf:resource="#IrreflexiveProperty"/>
  <rdf:type rdf:resource="#AsymmetricProperty"/>
  <owl:disjointWith rdf:resource="#hasPredeces
  sor"/>
  <rdfs:domain rdf:resource="#Car"/>
  <rdfs:range rdf:resource="#Car"/>
</owl:ObjectProperty>
```

   illustrates the use of these characteristics for a more precise specification of the aforementioned `hasFollower` object property.
7. **Concrete domains:** This extension provides support for the handling of *concrete sets* (real numbers, integers, strings, etc.) and *concrete predicates* (numerical comparisons, string comparisons and comparisons with constants) on these sets.

Appendix C. Reasoning services for $\mathcal{SROIQ}$-based ontologies

The $\mathcal{SROIQ}$ description logics that support OWL 2 are introduced in Krotzsch [70] and are thoroughly detailed in Horrocks [43]. Here, $\mathcal{SR} = \mathcal{ALC}$ + role chains ($\mathcal{R}$), $\mathcal{O}$ = nominals (closed classes), $\mathcal{I}$ = support for inverse rules, and $\mathcal{Q}$ = qualified cardinality restrictions. We note the extension of the grammar to include *role expressions* $\mathbf{R} := \mathbf{U} | \mathbf{N_R} | \mathbf{N_R^-}$ where $\mathbf{N_R}$ is the set of role names and $\mathbf{U}$ is the universal role. Also, alongside the TBox and ABox, the *RBox* is an integral part of $\mathcal{SROIQ}$ axioms.

Thus, from the grammar

$$C := N_C | \mathbf{C} \sqcap \mathbf{C} | \mathbf{C} \sqcup \mathbf{C} | \neg \mathbf{C} | \top | \bot | \forall \mathbf{R}.\mathbf{C} | \exists \mathbf{R}.\mathbf{C} | \geqslant n\mathbf{R}.\mathbf{C} | \leqslant n\mathbf{R}.\mathbf{C} | \exists \mathbf{R}.Self | \{N_I\}$$

where n is a non-negative integer, $\mathbf{C} \sqcap \mathbf{C}$ representing expressions of the form $C \sqcap D$ with C, D $\in \mathbf{C}$ and $\{N_I\}$ are individual names, $\mathcal{SROIQ}$ axioms are defined as follows.

$$\begin{array}{ll} \text{ABox}: & \mathbf{C}(N_I) \quad \mathbf{R}(N_I, N_I) \quad N_I \approx N_I \quad N_I \not\approx N_I \\ \text{TBox}: & \mathbf{C} \sqsubseteq \mathbf{C} \quad \mathbf{C} \equiv \mathbf{C} \\ \text{RBox}: & \mathbf{R} \sqsubseteq \mathbf{R} \quad \mathbf{R} \equiv \mathbf{R} \quad \mathbf{R} \circ \mathbf{R} \sqsubseteq \mathbf{R} \quad Disjoint(\mathbf{R}, \mathbf{R}) \end{array}$$

When applied to any given $\mathcal{SROIQ}$-based ontology, this set of axioms creates a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ for the domain being modeled. However, when applicable, the following reasoning tasks are required with regard to the TBox $\mathcal{T}$ and ABox $\mathcal{A}$ [71]:

Definition 5 (*Satisfiability*). A concept C is satisfiable *w.r.t. a TBox* $\mathcal{T}$ if there exists an interpretation $\mathcal{I} \models \mathcal{T}$ such that $C^I \neq \varnothing$.

Similarly, an ABox $\mathcal{A}$ is satisfiable *w.r.t. a TBox* $\mathcal{T}$ if there exists an interpretation $\mathcal{I} \models \mathcal{T} \cup \mathcal{A}$.

Definition 6 (*Subsumption*). A concept C is subsumed by D ($C \sqsubseteq_{\mathcal{T}} D$) with C, D $\in \mathbf{C}$ if for all interpretations $\mathcal{I}$, if $\mathcal{I} \models \mathcal{T}$ then $C^I \subseteq D^I$.

Definition 7 (*Equivalence*). Two concepts C and D (C, D $\in \mathbf{C}$) are equivalent with respect to $\mathcal{T}$ if for all interpretations $\mathcal{I}$, if $\mathcal{I} \models \mathcal{T}$ then $C^I = D^I$.

Definition 8 (*Disjointness*). Two concepts C and D (C, D $\in \mathbf{C}$) are disjoint with respect to $\mathcal{T}$ if for all interpretations $\mathcal{I}$, if $\mathcal{I} \models \mathcal{T}$ then $C^I \cap D^I = \varnothing$.

The reasoner should be able to systematically decide on the existence and satisfaction of these characteristics and assert (or infer) new facts and statements that are added to the knowledge base $\mathcal{K}$. However, reasoning over $\mathcal{K}$ in its wholeness is very inefficient. Fortunately, it has been proven that there are ways to reduce the complexity of reasoning to polynomial order through elimination of ABox and TBox axioms/concepts. This advance is formulated in the following results [43,71]:

Proposition 1 (*Satisfiability w.r.t. TBox*). *Subsumption, equivalence, and disjointness with respect to* $\mathcal{T}$ *are reducible to testing (un) satisfiability w.r.t.* $\mathcal{T}$.

Lemma 1 (*ABox Elimination*). *$\mathcal{SROIQ}$ concept satisfiability with respect to ABoxes, RBoxes, and TBoxes is polynomially reducible to $\mathcal{SROIQ}$ concept satisfiability with respect to RBoxes and TBoxes only.*

Similar result is formulated for the elimination of both the TBox and Universal Role thus, the following theorem addressing reduction.

Theorem 1 (*Reduction*).

1. *Satisfiability and subsumption of $\mathcal{SROIQ}$-concepts w.r.t. ABoxes, RBoxes, and TBoxes are polynomially reducible to (un) satisfiability of $\mathcal{SROIQ}$-concepts w.r.t. RBoxes.*
2. *Without lost of generality, we can assume that RBoxes do not contain role assertions of the form Irr(R), Tra(R), or Sym(R), and that the universal role is not used.*

This result reduces the standard $\mathcal{SROIQ}$ (concepts and ABoxes) inference problem to the one of determining the consistency of a $\mathcal{SROIQ}$-concept with respect to a reduced RBox where all role assertions are of the form Ref(R) or Dis(R,S). Krotzsch [70] also points out the need for "structural restrictions" on $\mathcal{SROIQ}$-based ontologies as a whole in order to guarantee the existence of correct and terminating algorithms to support inferencing. We note that the first restriction, simplicity is concerned with non-simple roles resulting from roles composition. Second, regularity is concerned with RBox axioms. The main goal of such restrictions is to limit the occurrence of cyclic dependencies between complex roles and inclusion axioms (i.e., see the OWL constructor owl:SuperPropertyOf (chain)). Horrocks et al. [43] build on these results to develop and describe a terminating, sound, and complete tableau-based algorithm that decides the consistency of a $\mathcal{SROIQ}$-concepts with respect to a reduced RBox.

Theorem 2 (*Decidability*). *The tableau algorithm decides satisfiability and subsumption of $\mathcal{SROIQ}$-concepts with respect to ABoxes, RBoxes, and TBoxes.*

References

[1] P. Marwedel, Embedded System Design: Embedded systems Foundations of Cyber-Physical Systems, second ed., Springer, 2011.
[2] C. Myers, A Modeling and Verification of Cyber-Physical Systems, Design Automation Summer School, University of Utah, 2011.
[3] NIST, Strategic R&D Opportunities for 21st Cyber-physical Systems: Connecting Computer and Information Systems with the Physical World, National Institute of Science and Technology (NIST), Gaithersburg, MD, USA, 2013.
[4] NIST, Strategic Vision and Business Drivers for 21st Century Cyber-physical Systems, National Institute of Science and Technology (NIST), Gaithersburg, MD, USA, 2013.
[5] P. Derler, E.A. Lee, A.L. Sangiovanni-Vincentelli, Addressing Modeling challenges in Cyber-Physical Systems, Technical Report N0. UCB/EECS-2011-17, Electrical Engineering and Computer Sciences University of California Berkeley, 2011.
[6] J. Sztipanovits, T. Bapty, G. Karsai, S. Neema, Model-Integration and Cyber Physical Systems: A Semantic Perspective, Keynote at FM 2011, Limerick, Ireland, 2011.
[7] E.A. Lee, CPS Foundations, DAC10, Anaheim, California, USA, 2010.
[8] E.A. Lee, Cyber-physical systems: a rehash or a new intellectual challenge?, in: Invited Talk in the Distinguished Speaker Series, Design Automation Conference (DAC), Austin, TX, 2013.
[9] J. Doyle, Feedback Control Theory, Class Notes, CDS 212, Fall 2011, 2011. <https://www.cds.caltech.edu/wiki/index.php/>.
[10] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng, Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II), Tech. Rep. ECB/EECS-2008-28, Department Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, April 2008.
[11] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, Berlin, 1992.
[12] A. Cataldo, E.A. Lee, X. Liu, E. Matsikoudis, H. Zheng, A Constructive Fixed-Point Theorem and the Feedback Semantic of Timed Systems, Workshop on Discrete Event Systems (WODES), Ann Arbor, Michigan, 2006. <http://ptolemy.eecs.berkeley.edu/publication/papers/06/constructive/>.
[13] J.C. Jensen, D.H. Chang, E.A. Lee, A model-based design methodology for cyber-physical systems, in: IEEE Workshop on Design, modeling, and Evaluation of Cyber-Physical Systems (CyPhy), Istanbul, Turkey, 2011.
[14] A. Bhave, B. Krogh, D. Garlan, B. Schmerl, Multi-domain Modeling of Cyber-Physical Systems Using Architectural Views, Dept. of Electrical & Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15217, 2010.
[15] N. Cutland, Computability: An Introduction to Recursive function Theory, Cambridge University Press, Cambridge, MA, 1997.
[16] E.A. Lee, The Problem with Threads, EECS Department, University of California at Berkeley, Berkeley, CA, 2006.
[17] B. Lampson, Getting computers to understand, J. Assoc. Comput. Mach. (JACM) 50 (2003) 70–72.
[18] A.M. Turing, On computable numbers, with application to the Entscheidungsproblem, Proc. Lond. Math. Soc. 2 (42) (1936) 230–265.
[19] M. Sipser, The halting problem, in: Introduction to the Theory of computation, second ed., PWS Publishing, 2005, pp. 173–182. ISBN 0-534-94728-X.
[20] H. Graves, Current state of Ontology in Engineering Systems, OMG: Ontology Action Team, 2012. <www.omgwiki.org/MBSE/oku.php?id>.
[21] M. Yoshiokaa, Y. Umedab, H. Takedac, Y. Shimomurad, Y. Nomaguchie, T. Tomiyama, Physical concept ontology for the knowledge intensive engineering framework, Adv. Eng. Inform. (2004).
[22] F. Song, G. Zacharewicz, D. Chen, An ontology-driven framework towards building enterprise semantic information layer, Adv. Eng. Inform. (2013).
[23] J. Nanda, T.W. Simpson, S.R. Kumara, S.B. Shooter, A methodology for product family ontology development using formal concept analysis and web ontology language, J. Comput. Inform. Sci. Eng. ASME 6 (2006) 103–113.

[24] C.E. Bock, X. Zha, H. Suh, J.H. Lee, Ontological product modeling for collaborative design, Adv. Eng. Inform. (2010).
[25] G. La Rocca, Knowledge-based engineering: between AI and CAD. Review of a language-based technology to support engineering design, Adv. Eng. Inform. (2012).
[26] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, J. Dickinson, R. Thomas, A. Pardasani, H. Xue, Systems integration and collaboration in architecture, engineering, construction, and facilities management: a review, Adv. Eng. Inform. (2010).
[27] M. Quillian, Semantic memory, in: M. Minsky (Ed.), Semantic Information Processing, MIT Press, 1968, pp. 227–270.
[28] J.F. Sowa, A. Borgida, Principles of semantic networks: explorations in the representation of knowledge, in: John F. Sowa (Ed.), 1991.
[29] J. Allen, A. Frisch, What's in a Semantic Network, 20th. Annual Meeting of ACL, Toronto, 1982, pp. 19–27.
[30] M. Minsky, A Framework for Representing Knowledge, Technical Report, MIT-AI Laboratory, Massachusetts Institute of Technology Cambridge, MA, USA, 1974.
[31] R.J. Brachman, H.J. Levesque, The tractability of subsumption in frame-based description languages, in: 4th National Conference of the American Association for Artificial Intelligence (AAAI-84). Austin, TX, 1984, pp. 34–37.
[32] P.J. Hayes, The logic of frames, in: D. Metzing (Ed.), Frame Conceptions and Text Understanding, deGruyter, Berlin, 1980, pp. 46–61.
[33] J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984. ISBN 978-0-201-14472-7.
[34] M. Pavlic, A. Mestrovic, A. Jakupovic, Graph-based formalisms for knowledge representation, in: 17th World Multi-Conference on Systemics, Cybernetics and Informatics, July 9–12, Orlando, Florida, USA, 2013.
[35] M. Chein, M.-L. Mugnier, Graph-Based Knowledge Representation: Computational Foundations of Conceptual Graphs, first ed., Springer Publishing Company, Incorporated, 2008.
[36] F. Baader, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, The Description Logic Handbook: Theory, implementation, and applications, Cambridge, 2003.
[37] G. De Giacomo, M. Lenzerini, Concept language with number restrictions and fixpoints, and its relationship with mu-calculus, in: 11th European Conference on Artificial Intelligence (ECAI), John Wiley and Sons, Ltd., 1994.
[38] K. Schild, Terminological cycles and the propositional with mu-calculus, in: J. Doyle, E. Sandewall, P. Torasso (Eds.), 4th Int. Conference on the Principle of Knowledge Representation and Reasoning (KR-94), 1994, pp. 509–520.
[39] Semantic Web Activity, 2013. <http://www.w3.org/2001/sw/>.
[40] P.F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, Recommendation, W3C, 2004. <http://www.w3.org/TR/owl-semantics/>.
[41] F. Baader, I. Horrocks, U. Sattler, Description logics as ontology languages for the semantic web, in: Dieter Hutter, Werner Stephan (Eds.), Mechanizing Mathematical Reasoning: Essays in Honor of Jrg Siekmann on the Occasion of His 60th Birthday, Lecture Notes in Artificial Intelligence, vol. 2605, Springer, 2005, pp. 228–248.
[42] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: the next step for OWL, J. Web Semant.: Sci. Serv. Agents World Wide Web 6 (4) (2008) 309–322.
[43] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible SROIQ, in: 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), AAAI Press, 2006, pp. 57–67.
[44] L. Petnga, M.A. Austin, Ontologies of time and time-based reasoning for MBSE of cyber-physical systems, in: 11th Annual Conference on Systems Engineering Research (CSER 2013), Georgia Institute of Technology, Atlanta, GA, 2013.
[45] N. Choi, I.Y. Song, H. Han, A Survey on Ontology Mapping, SIGMOD Rec, 2006, pp. 34–41.
[46] T. Menzies, Applications of abduction: knowledge-level modeling, Int. J. Human Comput. Stud. 45 (1996) 305–335.
[47] L. Console, D.T. Dupre, P. Torasso, On the relationship between abduction and deduction, J. Logic Programm. 1 (1991) 661–690.
[48] P.K. Praitosh, The Heuristic Reasoning Manifesto, Qualitative Reasoning Group, Electrical and Computer Science, Northwestern University, Evanston IL 60208, USA, 2006.
[49] P.R. Cohen, M.R. Grinberg, A theory of heuristic reasoning about uncertainty, AI Mag. 4 (2) (1983) 17–24.
[50] L. Elder, R. Paul, Thinkers Guide to Understanding the Foundations of Ethical Reasoning, Foundation for Critical Thinking, 2013.
[51] A.A.C.U, Ethical Reasoning Value Rubric, Association of American Colleges & Universities (AACU), 2010.
[52] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, Pellet: a practical OWL-DL reasoner, Web Semant.: Sci. Serv. Agents World Wide Web 5 (2) (2007) 51–53.
[53] L. Petnga, M.A. Austin, Semantic platforms for cyber-physical systems, in: 24nd Annual International Symposium of The International Council on Systems Engineering (INCOSE 2014), Las Vegas, Nevada, 2014.
[54] D. Hurwitz, The "Twilight Zone" of Traffic Costs Lives at Stoplight Intersections, Oregon State University, Corvallis, Oregon, USA, 2012.
[55] J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[56] L. Petnga, M.A. Austin, Cyber-physical architectures for modeling and enhanced operations of connected-vehicle systems, in: 2nd International Conference on Connected Vehicles (ICCVE 2013), Las Vegas, Nevada, 2013.
[57] Apache Jena, 2013. <http://www.jena.apache.org>.
[58] Time Ontology in OWL, 2006. <http://www.w3.org/TR/owl-time/>.
[59] Jscience, 2013. <http://www.jscience.org>.
[60] N. Leveson, A systems-theoretic approach to safety in software-intensive systems, IEEE Trans. Dependable Secure Comput. 1 (2004) 66–86.
[61] T. Zuk, S. Carpendale, Visualization of uncertainty and reasoning, in: A. Butz et al. (Eds.), SG 2007, LNCS, 4569, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 164–177.
[62] J.Y. Halpern, Reasoning About Uncertainty, second ed., Ain A. Sonin, 2005.
[63] S.J. Russell, P. Norvig, Artificial Intelligence, first ed., A Modern Approach, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1995.
[64] P. Klinov, Practical Reasoning in Probabilistic Description Logic, Ph.D. Thesis, School of Computer Science, University of Manchester, 2011.
[65] M. Huntbach, Notes on Reasoning with Uncertainty, Class Notes, Artificial Intelligence I, Queen Mary and Westfield College, London, 1996.
[66] F. Baader, P. Hanschke, A scheme for integrating concrete domains into concept languages, in: J. Doyle, E. Sandewall, P. Torasso (Eds.), 12th International Joint Conference on Artificial Intelligence (IJCAI91), 1991, pp. 452–457.
[67] I. Horrocks, U. Sattler, A description logic with transitive and inverse roles and role hierarchies, J. Logic Comput. 9 (3) (1999) 385–410.
[68] C. Lutz, The Complexity of Reasoning with Concrete Domains, LTCS-Report 99-01, Aachen university of Technology Research group for Theoretical Computer Science, 1999.
[69] F. Baader, I. Horrocks, U. Sattler, Description logics, in: F. van Harmelen, V. Lifschitz, B. Porter (Eds.), Handbook of Knowledge Representations, Elsevier, 2008, pp. 135–180 (Chapter 3).
[70] M. Krtzsch, F. Simanck, I. Horrocks, A Description Logic Primer, Department of computer Science, University of Oxford, UK, 2013. Version 1.2.
[71] N. Olivetti, Artificial Intelligence: Introduction to Description Logics, INCA-LSIS, Paul Cezanne university, Marseille, France, 2009.