Modelling of Cyber-Physical Systems through Domain-Specific Languages: Decision, Analysis, Design

Marcus Gerhold
m.gerhold@utwente.nl
Formal Methods and Tools
University of Twente
Enschede, The Netherlands

Aliaksei Kouzel
a.kouzel@student.utwente.nl
Technical Computer Science
University of Twente
Enschede, The Netherlands

Haroun Mangal
h.mangal@student.utwente.nl
Technical Computer Science
University of Twente
Enschede, The Netherlands

Selin A. Mehmed
s.a.mehmed@student.utwente.nl
Technical Computer Science
University of Twente
Enschede, The Netherlands

Vadim Zaytsev
vadim@grammarware.net
Formal Methods and Tools
University of Twente
Enschede, The Netherlands

Abstract

Cyber-Physical Systems (CPS) integrate computational algorithms and physical components, requiring sophisticated modelling techniques to address complex interactions and dynamics. This paper explores the creation of Domain-Specific Languages (DSLs) tailored for CPS, focusing on the initial three critical phases: decision, analysis, design. We present four key aspects to address in the decision phase, design an ontology as a domain model for the analysis phase, and collect some advice for the design phase. By systematically addressing these phases, we provide a comprehensive framework for developing DSLs that can efficiently model CPS, facilitating improved design, verification, and deployment of these intricate systems.

CCS Concepts: • **Software and its engineering** → **Domain specific languages**; *Interoperability*; *Design languages*; • **Information systems** → **Ontologies**; • **Networks** → Network protocol design.

Keywords: Cyber-Physical Systems, Ontological Analysis, Domain-Specific Languages

ACM Reference Format:
Marcus Gerhold, Aliaksei Kouzel, Haroun Mangal, Selin A. Mehmed, and Vadim Zaytsev. 2024. Modelling of Cyber-Physical Systems through Domain-Specific Languages: Decision, Analysis, Design. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), 22–27 September 2024, Linz, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3652620.3688348

MODELS Companion '24, 22–27 September 2024, Linz, Austria

1 Introduction

Cyber-Physical Systems (CPS) represent collaborations of computational algorithms and physical components [46], creating a network where digital systems monitor and control physical processes through sensors and actuators. These systems form a feedback loop where sensor data informs computational decisions, and actuators execute these decisions to affect the physical environment. Examples of CPS include smart grids, autonomous vehicles, and advanced medical monitoring systems [50, 64]. CPS hold substantial potential across diverse domains such as smart manufacturing, robotics, healthcare, intelligent transportation, and smart cities, offering benefits like enhanced automation, improved safety, and optimised resource usage [23, 29, 46, 55].

Realising the full potential of CPS poses significant challenges due to their inherent complexity and heterogeneity [64]. Integrating the continuous, concurrent physical world with the discrete, sequential cyber world often leads to non-deterministic behaviours, complicating the development of reliable and dependable models. Model-Driven Engineering (MDE) addresses these complexities by breaking down CPS into manageable components, yet the intricacies of such systems demand specialised modelling approaches [13].

Domain-Specific Languages (DSLs) offer a promising solution for CPS modelling by providing tailored notations and constructs specific to the domain [14, 68]. Unlike general-purpose languages (GPLs), DSLs can encapsulate domain-specific knowledge, simplifying the modelling process and enhancing communication between domain experts and developers [44]. For instance, UML and VHDL are well-known DSLs in their respective domains.

It is known from prior work that DSL development can be conceptually split into phases of **decision**, **analysis**, **design** and **implementation** [45]. By addressing these phases, we wish to provide a comprehensive framework for developing DSLs that can efficiently model CPS. Essentially we want the answers to the following research questions:

RQ1: What are critical aspects that DSLs must address to ensure that modelling and implementation of CPS to be most effective?

RQ2: How can we analyse and model the CPS domain to understand the foundational concepts and relationships that underpin these systems?

RQ3: What aspects are important to address for the design of the DSLs for CPS?

Thus, the paper focuses on the creation of DSLs specifically for the CPS domain, emphasising the initial three phases: decision, analysis and design. For RQ1 and the *decision phase*, we identify four critical aspects that DSLs for CPS must address to ensure effective modelling and implementation. RQ2 and the *analysis phase* involve an ontological examination of the CPS domain to understand the foundational concepts and relationships that underpin these systems. Finally, for RQ3 and the *design phase* we list a few pieces of DSL design advice that seem applicable, and evaluate the impact of various computational models on the structure and functionality of DSLs. We skip over the *implementation phase* because it is (1) not that different for CPS compared to other domains; and (2) either not that different from implementation of GPLs, or (3) is done with language workbenches which are relatively well studied elsewhere [9, 15–17].

This paper is but a first humble step in the desired direction, and even though we will constantly bring up existing DSLs to highlight effective practices and methodologies, for the moment we avoid passing any judgement on them with respect to applicability, effectiveness, compatibility and other aspects, since that requires deeper examination and solid experiments. Ultimately, with this paper we make a contribution to the field of language engineering by offering insights and methodologies for creating specialised languages that enhance the modelling and development of CPS.

2 Decision Phase (RQ1)

Cyber-physical systems are complex and multidisciplinary, and thus challenging to build, however desirable they might be as a solution. Models are a promising solution to addressing this as they make it possible to break down complex systems and make concerns understandable and analysable. Models in software engineering can serve various purposes from code generation, system documentation, construction of the system, exploration of solution possibilities etc [13]. This phase helps to assess if creating a DSL is justified by providing evaluation criteria.

Important requirements for cyber-physical systems are interoperability, predictability, reliability, and dependability [23]. *Interoperability* refers to the ability of different components to work together. The other three are related to each other: *dependability* refers to the property of a system to perform functions without degradation in performance and outcome, *predictability* refers to the degree of being able to foresee a system's behaviour, and *reliability* refers to the degree of correctness in functioning. All three are related to the overall functioning of the system.

However, due to the complexity and heterogeneity of cyber-physical systems, modelling itself presents several difficulties. Physical processes and computational elements require different ways of modelling, timing becomes more crucial, various distributed behaviours arise, and components are heterogeneous and interconnected. A model must meet the requirements of reliability, predictability, dependability and interoperability all while correctly expressing the properties of the system.

The first issue in modelling CPS stems from the fact that the physical world is continuous and concurrent where many things are happening all at once, whereas the cyber one is discrete and sequential [30]. Thus, different modelling methods are utilised for the two parts of the system. For example, continuous-time models of dynamics are good at modelling physical processes, while state machines are good for modelling computations. However, integrating these two modelling paradigms is difficult [37] because it can lead to incompatibilities during the separate design processes or non-deterministic behaviour [14, 38]. Therefore, this is an issue that a CPS model must be able to address. For example, a CPS can be modelled as a hybrid system [14, 30, 72].

Another essential problem for CPS is that most models are unequipped to deal with timing semantics. Such semantics are crucial [30, 36, 38] due to the interaction with the real world, where time cannot be abstracted away as in the sequential cyber world. At the moment, models generally focus on increasing performance at the expense of predictability [30, 36] — caches, for example, are unpredictable but lead to faster execution times. This is perfectly acceptable for sequential programs but not so ideal for CPS. For example, if an executable model is a C program, then this program by itself provides no meaningful timing semantics, and the programmer must find ways around that obstacle, but methods for increased performance are plentiful. This is a failure of abstraction as the model is unequipped to deal with behaviour essential to the system. One key requirement for a CPS model is that it must be predictable, meaning it must have precise expressions of time rather than depending on the implementation (like a C program) to provide them.

The interaction between the physical and cyber parts of the system itself takes time. Data is not transmitted in zero time, there are network delays, components are separated in space, and computations take time [14]. This also necessitates solutions such as communication semantics, synchronous or asynchronous message transmissions and timestamps.

CPS are also complex and made up of multiple interacting components. Components are highly interconnected, and models of a CPS might grow more complex with time [14]. To be able to meet the previously mentioned requirements, a model should make it possible to ensure those components

work together as intended. A given model must also be reliable when small deviations from the expected operation occur [36]. Dealing with unexpected deviations can happen at higher or lower levels, and a CPS model must contain enough knowledge for it to be reliable [35] especially since many CPS are safety-critical systems [30]. The most obvious example of safety-critical systems are medical devices.

For the remainder of this section, we focus on four important aspects, one per subsection, providing motivation for them, as well as exemplifying them with existing languages that already implement them.

2.1 Specify How Components Work Together

Interoperability is one of the crucial requirements for CPS [23]. CPS are complex and intertwined systems; in order for a CPS to function correctly, its interconnected components must work together as intended. A DSL for CPS should make it possible to achieve this by implementing the ability to specify how the parts should work together and interact.

With a focus on interoperability specifically, there is aDSL [6]. The language models all the systems that a CPS is composed of, which themselves can be further broken down into systems or concrete parts. Parts and systems both have different requirements. For example, speed can be a requirement defined for a certain tractor part. Systems and parts only operate if the requirements are met. The DSL implements a way to define components recursively — a system can be composed of subsystems which might be composed of subsubsystems etc. — and constraints for the systems.

Chariot [54] is a DSL focused on clear separation of concerns between computation and communication, along with explicit definitions for system goals, objectives, and functionalities. Given that communication is a major challenge in CPS [14], Chariot supports heterogeneous communication middleware by maintaining a distinct separation between communication and computation logic. Chariot can express independent communication patterns, the system's overall state, available resources, known faults, as well as corresponding goals, objectives, and functionalities.

For the integration of sensors specifically, there is SensOr Interfacing Language (SOIL) [7]. It is a graphical domain-specific programming language for defining sensor interfaces. It models them as trees and specifies the information physically sensed by the sensor, any data required for operation, and functions that trigger tasks or change the internal state of the sensor. SOIL allows for the easy definition of required interactions between different components and the communication of measurement results.

MuScADeL [8] is a DSL for the deployment of multi-scale systems. Multi-scale systems are highly heterogeneous systems and are composed of various components and families of components that interact together. While not directly related to CPS, both are complex systems of many components that must interact as required. In a DSL we should be able to list components, the dependencies of each component, as well as any constraints that need to be satisfied. A less relevant feature is the ability to define probes, which collect data about the system for the purpose of deployment.

Chauhan et al [11] developed a framework for creating CPS with modelling languages. They aim to address issues such as complexity due to CPS consisting of various entities like sensors and actuators, differences in platforms that components run on, and various types of interactions components can have. Their framework allows for specifying domain-specific constructs like sensors, actuators, tags, and storage. Sensors can be periodic (sample data at regular intervals), event-driven (have activation triggers), or request-based (responding to users). It also supports computational services for generating results from measurements, issuing system requests, and executing commands. It also includes options for user interactions, such as notifications, and deployment specifications. Overall, it enables defining components, sensor types, computation methods, system interactions, and deployment options.

On a more concrete level, common features in DSLs implementing the ability to specify how components work together are constraints and requirements, the ability to define the individual components and their characteristics or functionalities, communication protocols, states and descriptions of how changes in state are triggered, and interactions between components. Structures such as graphs might also be used to better model interactions between components of the system.

2.2 Define Flow of Operation

In a CPS, computational elements control or monitor physical processes. A DSL can facilitate this by making it possible to specify what to control or monitor and the appropriate responses to changes in the physical world. For example, if some part of the system reaches a certain temperature, a model can define an appropriate action (such as shutting down the system to prevent overheating) or implement constraints (such as the safety limit).

AMon [66] is a DSL that monitors different states and provides definitions for the data flow within the CPS. It allows modellers to define various rules for the system, sample data with given frequency, specify which devices check for which rules and monitor what data, limit rules to certain devices or the entire system, etc. AMon is ideal for adaptive monitoring of CPS and defining the general flow of data between components.

Hoyos et al [26] developed a DSL for context-aware systems that interpret the context and modify the system based on it. This is done with context sensors, which brings context-aware systems into CPS. The proposed language models entities (people or objects) and their context, with attributes (such as the location and time) and the source of that context (e.g., GPS or a clock, active with certain accuracy). Rules

define which actions to take based on context facts. The DSL can also deal with the problem of sensors not being fully reliable with a notion of context quality.

Another option is task-oriented programming. Steenvoorden et al [61] give a formalisation of task-oriented programming. Tasks are interactive units of work based on information sources. Koopman et al [33] present an example DSL. It uses lightweight threads that produce immediate results after each evaluated step. There is a well-defined evaluation order of tasks, which can communicate via shared data sources. Tasks can be delayed, executed simultaneously, be sequentially ordered, or act based on the output of other tasks. The last part means the DSL is capable of reacting to the physical environment. The delays give the language some very basic timing semantics.

There are various features languages in this category implement. Examples include rules to changes in the physical world with the accompanying action to execute, definitions of states, specifying which components monitor what, and tasks and their timing of execution.

2.3 Express Timing Semantics

Timing is of crucial importance to cyber-physical systems. Tasks must execute and finish at the correct time and order. Unlike software systems, a process taking too long does not just impact the performance of the application but might very well be incorrect behaviour for the system. For example, it is critical that a self-driving car applies the brakes at just the right time and not too late; failing to do so might well be catastrophic. In a CPS that directly controls some physical process a delay in time is in many circumstances unacceptable, especially in a safety-critical system. This means that a DSL must have some form of timing semantics to introduce things like delays, deadlines, actions happening simultaneously, and just general task scheduling.

Triton [69] is a DSL with real-time scheduling. It defines scheduling blocks which contain tasks and are parametrised by time. It additionally implements constraints and defines the appropriate action in case a violation of the constraint occurs. For example, using the DSL one can schedule a task to happen in 4 milliseconds. However, in case the thermometer reaches a certain value, the task can be permanently stopped from executing or skipped until the temperature is within normal range again.

Lohstroh et al [40] proposed a language that implements timing semantics. The language accomplishes this by taking into account the relationship between logical time and physical time and specifying program behaviour by this relationship. It makes use of timestamps to create a "logical timeline" to deal with the problem of clock synchronisation that leads to a different "physical timeline" for different components in a system. Furthermore, periodic and once-off timers can be specified to trigger certain functions, delays can be induced, actions can be scheduled, and deadlines put in place for some events.

Goknil and Peraldi-Frati [19] present another DSL for specifying four types of timing requirements: delay requirements, synchronisation requirements, repetition requirements, and periodic requirements. All timing requirements interact with certain events or state changes. For example, a delay requirement describes how occurrences of a target event are placed relative to a source event. This means that a target event happens a certain amount of time after a source event, i.e. it is delayed. Synchronisation requirements refer to how close events can happen to each other (e.g. at the same time), repetition requirements give some limits to how often events can occur, and period requirements describe how often certain events are repeated. The language also addresses aspects of timing requirements such as time base, dimension, equations and variables and allows for their explicit modelling.

There are many different ways to implement timing constraints. Possible concrete features in this category are task scheduling, timelines, timestamps, and different ways to time something (whether periodically, a certain amount of time after some event, at the same time as some event, etc). Timing semantics are very closely related to the data flow feature because timing semantics arise precisely due to interactions with the physical world [10], especially when basing timing on a certain event in the physical world.

2.4 Combine Discrete with Continuous

A DSL must have a way to capture what is happening in the physical part of the system. However, the physical world is continuous and must be modelled as such. Unfortunately, this leads to incompatibilities with the model of the rest of the system which is discrete. Thus, a DSL must capture and model the physical world in a way that avoids this issue — by appropriately modelling the whole system as a hybrid one for example. This is a complex task but there are many options to choose from when constructing a solution.

CREST [32] is a DSL for hybrid systems modelling. It is created specifically for modelling CPS whose components "primarily interact through the exchange of physical resource flows such as water, heat or electricity" — that is, continuous resource flows. It accomplishes this through the use of modelling techniques such as hybrid automata, data-flow languages, and architecture description languages. CREST defines both diagrams for visual representation, and an internal DSL based on Python.

Another solution is xSHS [21], an executable domain-specific language that models the hybrid behaviour of cyber-physical systems. In this example, states in the model are captured also by ordinary differential equations in order to model the continuous behaviour of physical processes. It also has semantics for representing transitions between states and physical environment variables.

Diderot [31] is a DSL for scientific visualisation and image analysis. Its relevance comes from the fact that it supports the abstractions of continuous scalars. Similarly to CPS, most general-purpose programming language do not have the necessary abstractions for anything non-discrete and Diderot serves as a useful starting point to creating abstractions of more complicated mathematical operations.

Overall, representing a continuous, physical world in a DSL is complicated and requires the use of formalisms. Formalisms are mathematical objects consisting of abstract syntax and a formal semantics, of which the languages are a concrete implementation [10]. For example, xSHS made use of ordinary differential equations [21] and CREST made use of hybrid automata [32]. Implementing this last feature requires expertise on modelling physical systems as opposed to concrete features that can be described semantically.

2.5 Takeaways for RQ1

The most important aspects that we could identify, as interoperability (§ 2.1), explicit flow of operations (§ 2.2), having timing semantics (§ 2.3) and combining the discrete with the continuous (§ 2.4). These seem to correspond to both examples in the existing literature and the current wishes of our industrial partners. Further investigation is needed in the form of both detailed interviews with domain experts, as well as systematic literature reviews, to validate and refine this set of aspects.

3 Analysis Phase (RQ2)

Once the architectural decisions have been taken, the creation of a DSL proceeds by analysing the relevant domain [45]. This domain must be captured by the language vocabulary of the DSL such that all domain constructs can be expressed. From this point on, it is useful to start splitting the language into its syntax and semantics: the *syntax* prescribing what symbols are allowed in expressions of the language and how they can be combined into well-formed constructs; and the *semantics* defining the meaning and/or the behaviour of these symbols and their combinations. Some approaches also explicitly split semantics into the *semantic domain* of all possible meanings that can potentially be created with this language and the *semantic mapping* from syntax to this semantic domain [24].

The semantic domain is usually modelled by a domain model [3], which can be an analysis model [12], a conceptual model [49], a megamodel [73], or just a model in a domain-modelling language like DSVL [60]. In this paper, we will model the semantic domain using an **ontology**, which is of the mature technologies portable across domains. Within the context of model-based engineering, an ontology is a representation of domain knowledge [56]. Generally speaking, an ontology is denoted with (domain) concepts and the relationships between these concepts [22]. Numerous papers have been written about using an ontology for the development of a DSL. To name a few, Lyadova et al described a framework for developing DSLs by letting domain experts develop an ontology upon which DSL developers will base the language on [41]; Tairas et al constructed an ontology for air traffic communication and proposed a subsequent context-free grammar for the DSL design [63]; Utilin and Babkin discussed the evolution of an ontology and its DSL by adding new rules to the DSL which subsequently also adds new concepts and relationships to the ontology [65].

Four different kinds of anomalies can occur when mapping an ontology to a construct (which in this paper is a DSL) [47]:

- *Construct deficiency* means there is no construct for an ontological concept.
- *Construct overload* is when a single notation maps to multiple ontological concepts.
- *Construct redundancy* is when multiple notations map to the same concept.
- *Construct excess* is when a notation construct does not map to an ontological construct.

In case there is a construct deficiency, then the DSL is said to be *ontologically incomplete*. If any of the three other cases occur, then the DSL is *ontologically unclear* [47]. The goal of a good DSL is to have a one-to-one mapping from ontological concepts to the language vocabulary.

There are different kinds of formal notations to describe ontologies. For example, the Ontology Web Language (OWL) is a formal ontology for which Pereira et al created OWL2DSL, an OWL to DSL converter [51]. However, OWL is mostly used in the context of the Semantic Web. Bunge-Wand-Weber (BWW) is a different kind of formal notation for an ontology and is one of the leading ontology frameworks used [47, 67].

The proposed ontology in this paper uses the following concepts of the BWW ontology to describe CPS: *Thing* (an elementary unit), *Property* (an attribute belonging to a Thing), *State* (the values of all attributes of a Thing), *Event* (a change in State), *History* (all Events of a Thing), *Coupling* (whether the History of two Things are independent or not), *System* (Things which are connected to each other and have dependent Histories), *Composition* (all Things inside a System), *Environment* (all Things outside a System that interact with Things inside the System), *Structure* (the Coupling among the components of the Systems and the Environment), *Subsystem* (a System whose Composition and Structure are a subset of another System), *Input* (a Thing in a System acted upon by an Environmental Thing), and *Output* (a Thing in a System acting on an Environmental Thing) [20, 67].

While shaping the ontology, we refine our understanding of a CPS specification from a selection of available definitions from prior literature and a set of generic guidelines, into a more concrete definition. We have quite some Things from the physical world, such as sensors and actuators which are

connected to some network and send data in some format towards either edge devices or remote computers. There are also many concepts from the cyber part, like an algorithm that is being used to process the data, usually by performing tasks or operations, which need to be properly scheduled to be completed as well. Previous researchers commented that there should be an unambiguous division between the physical and cyber parts of the system [48], but the mutual influence is undeniable.

This leads us to one of our main contributions of this paper, the following ontology, also visualised on Figure 1:

- **Sensor** represents a physical sensor that takes measurements of some part of the physical world;
- **Process** represents a continuous process which influences the sensor (e.g., temperature changes);
- **Actuator** represents a physical actuator that can be used to exert influence on the physical world;
- **Protocol** represents a discrete protocol which is used to communicate to the actuator;
- **Network** represents a carrier of information among other CPS entities; it is a CPS by itself and can be viewed as a stack of its own protocols [27] or at least as a stateful connection enabling the contact with Sensors and Actuators;
- **Format** represents essentially the metamodel of the data produced by a sensor or consumed by an actuator;
- **Edge** represents an edge device that receives information from sensors and issues commands to actuators, while being operated by some agent;

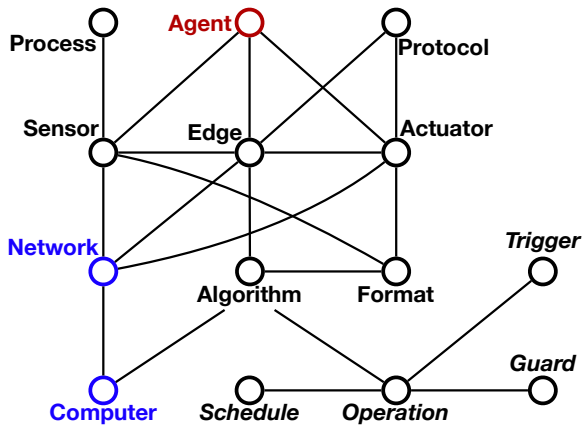

Figure 1. A graphical view of a CPS ontology. Nodes represent BWW [67] Things (in regular font) and Events (in italics), lines show Couplings. Black labelled nodes are a part of the System, blue nodes may or may not be considered to be a part of the System (depending on whether the focus is on embedded systems or on pervasive computing), red nodes are traditionally viewed as a part of the Environment.

- **Computer** represents a remote device accessible "in the cloud" through a network connection and used to carry on computations;
- **Algorithm** represents the purely cyber entity that models a computation;
- **Operation** is an Event representing one task or part of an algorithm that can be carried out separately;
- **Schedule** represents some management of operations in time on available hardware;
- **Trigger** represents an operation that enables and initiates another operation;
- **Guard** represents an operation that prevents execution of another operation until a certain condition is met;
- **Agent** represents an out-of-system entity that operates the system or interacts with it in some other way, up to and including communicating with Sensors and Actuators.

One can notice that we have followed our own advice from two sections ago: interoperability (§ 2.1) is guaranteed by making the Protocol and the Format explicit; the flow of operation (§ 2.2) is guided by Guards and Triggers in addition to normal Operations; the timing semantics (§ 2.3) resides in the Schedule; and the hybrid nature (§ 2.4) of the CPS comes to life with the distinction between a Process and a Protocol.

3.1 Takeaways for RQ2

Figure 1 is our answer to model the domain of CPS. Armed with this ontology, we can still make very different decisions about the constructs we want to have prominent in the DSL (collectively known as an "abstract syntax"), and even with more diversity make decisions about the way we want to write these things down textually or graphically (similarly known as the "concrete syntax"), but all those decisions can be guided or at least informed by this ontology as means of checking compatibility and conformance to the chosen domain. The BWW-based ontology we have presented here, can be compared to existing ontological frameworks for CPS such as those based on description logics [52] and also further refined and formalised inside frameworks like UFO [1] and thus engage in validation activities before the syntax of the language has crystallised.

4 Design Phase (RQ3)

Before designing a DSL, it is important to understand its application in the domain. A well-designed DSL capable of expressing diverse and heterogeneous CPS in a reliable, valid and diagnosable way, should exist within some model-driven framework, where the system's behaviour and health is inferred by its compositional model in comparison with observed inputs and outputs [53]. Specifically, a DSL is used to model the system and transform it into the primary language

of the application, which is then used for further development of diagnostic algorithms.

Feature-wise, based on the DSL's purpose and existing approaches to model-based design, the language can potentially support the following features:

(1) Detection of faults and their root causes [5, 6, 53].
(2) Choosing CPS sensors [4, 5].
(3) Evaluating diagnosis accuracy [4–6].
(4) Evaluating diagnosis costs [53].
(5) Model extraction from existing data [5].
(6) Model visualisation [4, 6].

However, the exact features would depend on the specific requirements of CPS stakeholders. Depending on the chosen feature set, the design can significantly vary. For instance, determining the root causes of CPS faults would require the mappings between anomalies and fault indicators to be embedded in the DSL design.

To determine the principles behind the DSL design, it is necessary to understand its intentions [71]. In the context of CPS diagnostics, the DSLs are expected to improve productivity during the development, maintenance, and utilisation of diagnostic software by facilitating error detection, system modification, and program understanding [17, 45]. The latter is especially important as code comprehension can be time-consuming and may require more than half of the time allocated for software maintenance [70]. Another principal intention behind DSLs is to enhance interaction with domain specialists, as they are not always familiar with GPL concepts such as algorithms and data structures [17]. Taking this into account, we recall the following design principles based on the existing work of Fowler and Parsons [17], Hermans et al [25], Karsai et al [28], Zaytsev [71], Wąsowski and Berger [68]:

(1) Use concise and simple syntax to facilitate communication with stakeholders.
(2) Use domain-specific terminology in the syntax and the semantic model to improve understandability for domain experts.
(3) Use common conventions familiar to everyday coding practices.
(4) Avoid ambiguity in definitions and reasoning.
(5) Avoid resembling a natural language, as this introduces syntactic sugar that obscures the semantics.
(6) Separate the DSL's semantic model and syntax, allowing their independent evolution.
(7) Implement automatic migration among DSL versions.
(8) Implement testing of the DSL's parser, scripts, and the semantic model.

Let us zoom in on one more principle which is important for DSL design and crucial for CPS: choosing the right computational model. This model determines the framework used to describe the computational processes and define the language semantics [17]. Most popular GPLs, such as Java, Python, and C++, utilise an imperative approach where the program consists of statements executed step by step [59]. They provide selection statements, iterative statements, perhaps support for object-oriented programming, and other constructs [59]. For many domains, this approach is known to be badly suitable and unnecessarily complex for domain experts, which is why many DSLs explore alternative computation models like a decision table, a state machine, or a production rule system, instead [17].

Imagine a thermostat system, one of the simplest possible cyber-physical systems. Within this system, the diagnosis is based on conditions such as temperature readings (T), sensor status (S), and error codes (E). Also, the system adheres to the following rules consecutively:

- If $E = 1$, then the output is "system failure".
- If $S \neq \text{OK}$, then the output is "sensor failure".
- If $T > 30$, then the output is "high temperature".
- Otherwise the system functions normally.

If we translate this into a decision table, it will look similar to the following:

```
T>30 ; S=OK  ; E=0 ; D=high_temperature
T=_  ; S!=OK ; E=0 ; D=sensor_failure
T=_  ; S=_   ; E=1 ; D=system_failure
```

As can be observed, this approach is efficient in combining the outputs of multiple interacting conditions. It is also well understood by both software engineers and domain experts [17]. This model can be used to define the correct system behaviour in diagnostics as a set of conditions leading to either fault/non-fault states or the probabilities of failures. An example of this model's usage can be found in the work by Barbini et al [4], which introduces a model-based approach for computing the system's diagnosability by generating Bayesian networks. However, the drawback of this model is that defining input conditions can be time-consuming, especially for complex systems [17].

Another alternative is translating the thermostat example to a state machine, which defines the system as a set of states and transitions between them, as shown below.

```
normal -> sensor_failure
    when E = 0 and S != OK
normal -> high_temperature
    when E = 0 and S = OK and T > 30
normal -> system_error
    when E = 1
```

This approach can be used to describe CPS diagnostics with "normal" states and transitions that lead to "faulty" states. Its application can be found in DSLs such as SHIFT [2], which focuses on describing complex systems, and Facile [58], which is used for micro-architecture simulations.

The last alternative is the production rule system. It is similar to the decision table, but the difference is that it focuses on the behaviour of individual rules rather than the whole table [17].

```
E = 1   => D = system_failure
S != OK => D = sensor_failure
T > 30  => D = high_temperature
```

This model is more compact than the decision table, but engineers should also consider how rules interact with each other.

Besides these three, other computational models also exist, and can also turn out to be suitable for the domain of CPS diagnostics. For example, Petri Nets [18] are known to be suitable for the description and analysis of systems characterised by concurrency, synchronisation, and resource sharing; or fault trees [57] for representing and analysing the causes of system failures through a visually representable hierarchical decomposition. Formal temporal logics (recall § 2.3) such as LTL have also been reported to be effective when used together with powerful system modelling languages like SysML [39].

4.1 Takeaways for RQ3

There are many principles and good practices in DSL design, and most of them can be made applicable to DSLs for CPS. When assigning priority to them, one should remember the results we collected for previous phases: for instance, the ontology from Figure 1 can help to stick to the expected terminology, and the main aspects of § 2 can guide the designers in choosing the right underlying computation model — which, as we have demonstrated, can have tremendous immediate effect on the overall design of a DSL.

5 Conclusion

In conclusion, the development of Domain-Specific Languages for reliable and diagnosable Cyber-Physical Systems presents a challenge with many aspects. Addressing it requires careful consideration of domain-specific requirements, computational models, as well as stakeholder needs, which we plan to do in the scope of a larger project with five industrial partners. The design of such DSLs must balance between providing expressive power for capturing intricate system behaviours and maintaining simplicity for domain experts to effectively utilise their knowledge and interpret the models written in the DSL. Key features such as interoperability, flow of operation, timing semantics, and hybrid system modelling are critical in ensuring that DSLs accurately represent CPS complexities while facilitating efficient diagnostics and system analysis.

In this paper we have traversed some existing literature on the topic, which, albeit not exhaustive, provided us with a number of concrete insights which informed the next steps. We have also proposed a conceptual model of this domain in a form of Bunge-Wand-Weber-based ontology, and explored various computational models including decision tables, state machines, and production rule systems, each offering distinct advantages in modelling CPS diagnostics. Decision tables excel in combining multiple conditions into clear diagnostic outputs, state machines provide visual clarity on system states and transitions, while production rule systems offer concise rule-based logic closer to the rules domain experts use in their daily lives. The choice of computational model should align with the specific diagnostic requirements and the expertise of stakeholders involved.

Leveraging formal ontologies such as Bunge-Wand-Weber provided us with a structured approach to defining CPS components and their relationships, ensuring consistency and clarity in future DSL design. We believe that this ontology-driven approach facilitates validation and refinement of DSLs before their implementation, enhancing their utility in real-world CPS applications.

Overall, the upcoming design and implementation (as the next two phases) of DSLs for CPS diagnostics will require a systematic approach that integrates domain knowledge, computational modelling techniques, and validation methodologies. By addressing these aspects comprehensively, DSLs can effectively support the development, deployment, maintenance, operation and optimisation of CPS systems, ultimately enhancing their reliability, performance and safety in diverse application domains.

Acknowledgments

This publication is part of the project ZORRO with project number KICH1.ST02.21.003 of the research programme Key Enabling Technologies (KIC) which is (partly) financed by the Dutch Research Council (NWO). [34, 42, 43, 62]

References

[1] João Paulo A. Almeida, Giancarlo Guizzardi, Tiago Prince Sales, and Ricardo A. Falbo. 2019. gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO). http://purl.org/nemo/doc/gufo.

[2] Marco Antoniotti and Aleks Göllü. 1997. SHIFT and SMART-AHS: A Language for Hybrid System Engineering Modeling and Simulation. In *Proceedings of the Conference on Domain-Specific Languages (DSL)*. USENIX Association, Santa Barbara, CA, USA, 171–182. https://www.usenix.org/conference/dsl-97/shift-and-smart-ahs-language-hybrid-system-engineering-modeling-and-simulation

[3] Colin Atkinson and Thomas Kühne. 2008. Reducing Accidental Complexity in Domain Models. *Journal of Software & Systems Modeling* 7, 3 (01 Jul 2008), 345–359. https://doi.org/10.1007/s10270-007-0061-0

[4] Leonardo Barbini, Carmen Bratosin, and Thomas Nägele. 2021. Embedding Diagnosability of Complex Industrial Systems Into the Design Process Using a Model-Based Methodology. In *PHM Society European Conference*. PHM Society, 9. Issue 6. https://doi.org/10.36001/phme.2021.v6i1.2806

[5] Anup Barve. 2005. *Model-Based Diagnosis — An ASML Case Study*. Master's thesis. Delft University of Technology, The Netherlands. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6435e31cee1b1a7c4b1d481abc5fd8a41e27c51f

[6] Freek van den Berg, Vahid Garousi, Bedir Tekinerdogan, and Boudewijn R. Haverkort. 2018. Designing Cyber-Physical Systems with aDSL: a Domain-Specific Language and Tool Support. In *Proceedings of the 13th Annual Conference on System of Systems Engineering (SoSE)* (Paris, France). IEEE, 225–232. https://doi.org/10.1109/SYSOSE.2018.8428770

[7] M. Bodenbenner, M. P. Sanders, B. Montavon, and R. H. Schmitt. 2021. Domain-Specific Language for Sensors in the Internet of Production. In *Production at the Leading Edge of Technology*, Bernd-Arno Behrens, Alexander Brosius, Wolfgang Hintze, Steffen Ihlenfeldt, and Jens Peter Wulfsberg (Eds.). Springer, Berlin, Heidelberg, 448–456. https://doi.org/10.1007/978-3-662-62138-7_45

[8] Raja Boujbel, Sam Rottenberg, Sébastien Leriche, Chantal Taconet, Jean-Paul Arcangeli, and Claire Lecocq. 2014. MuScADeL: A Deployment DSL Based on a Multiscale Characterization Framework. In *Proceedings of the 38th IEEE International Computer Software and Applications Conference Workshops*. IEEE, 708–715. https://doi.org/10.1109/COMPSACW.2014.120

[9] Mark van den Brand. 2023. A Personal Retrospective on Language Workbenches. *Journal of Software and System Modeling* 22, 3 (2023), 847–850. https://doi.org/10.1007/S10270-023-01101-9

[10] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. 2012. Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems. In *Proceedings of the Sixth International Workshop on Multi-Paradigm Modeling* (Innsbruck, Austria) *(MPM)*. ACM, New York, NY, USA, 49–54. https://doi.org/10.1145/2508443.2508452

[11] Saurabh Chauhan, Pankesh Patel, Flávia C. Delicato, and Sanjay Chaudhary. 2016. A Development Framework for Programming Cyber-Physical Systems. In *Proceedings of the Second International Workshop on Software Engineering for Smart Cyber-Physical Systems* (Austin, Texas) *(SEsCPS)*. ACM, New York, NY, USA, 47–53. https://doi.org/10.1145/2897035.2897039

[12] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes. 1994. *Object-Oriented Development: The Fusion Method.* Prentice Hall.

[13] Benoît Combemale, Robert W. France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim Steel, and Didier Vojtisek. 2016. *Engineering Modeling Languages: Turning Domain Knowledge into Tools.* CRC Press. https://doi.org/10.1201/b21841

[14] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni Vincentelli. 2012. Modeling Cyber–Physical Systems. *Proc. IEEE* 100, 1 (2012), 13–28. https://doi.org/10.1109/JPROC.2011.2160929

[15] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël D. P. Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad A. Vergu, Eelco Visser, Kevin van der Vlist, Guido Wachsmuth, and Jimi van der Woning. 2015. Evaluating and Comparing Language Workbenches: Existing Results and Benchmarks for the Future. *Computer Languages, Systems and Structures* 44 (2015), 24–47. https://doi.org/10.1016/J.CL.2015.08.007

[16] Martin Fowler. 2005. Language Workbenches: The Killer-App for Domain Specific Languages? MartinFowler.com. https://martinfowler.com/articles/languageWorkbench.html

[17] Martin Fowler and Rebecca Parsons. 2007. *Domain-Specific Languages.* Addison-Wesley Professional.

[18] Claude Girault and Rüdiger Valk. 2003. *Petri Nets for Systems Engineering — A Guide to Modeling, Verification, and Applications.* Springer. https://doi.org/10.1007/978-3-662-05324-9

[19] Arda Goknil and Marie-Agnès Peraldi-Frati. 2012. A DSL for Specifying Timing Requirements. In *Proceedings of the Second IEEE International Workshop on Model-Driven Requirements Engineering (MoDRE)*. IEEE, 49–57. https://doi.org/10.1109/MoDRE.2012.6360074

[20] Boryana Goncharenko and Vadim Zaytsev. 2016. Language Design and Implementation for the Domain of Coding Conventions. In *Proceedings of the Ninth International Conference on Software Language Engineering (SLE)*, Tijs van der Storm, Emilie Balland, and Dániel Varró (Eds.). ACM, 90–104. https://doi.org/10.1145/2997364.2997386

[21] Chunlin Guan, Yi Ao, Dehui Du, and Frédéric Mallet. 2018. xSHS: An Executable Domain-Specific Modeling Language for Modeling Stochastic and Hybrid Behaviors of Cyber-Physical Systems. In *Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 683–687. https://doi.org/10.1109/APSEC.2018.00090

[22] Nicola Guarino. 1995. Formal Ontology, Conceptual Analysis and Knowledge Representation. *International Journal of Human Computer Studies* 43, 5-6 (1995), 625–640. https://doi.org/10.1006/IJHC.1995.1066

[23] Volkan Gunes, Steffen Peter, Tony Givargis, and Frank Vahid. 2014. A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems. *KSII Transactions on Internet and Information Systems* 8, 12 (Dec. 2014), 4242–4268. https://doi.org/10.3837/tiis.2014.12.001

[24] David Harel and Bernhard Rumpe. 2000. *Modeling Languages: Syntax, Semantics and All That Stuff Part I: The Basic Stuff.* Technical Report MCS00-16. Weizmann Institute. https://www.se-rwth.de/staff/rumpe/publications/Modeling-Languages-Syntax-Semantics-and-All-That-Stuff.pdf

[25] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2009. Domain-Specific Languages in Practice: A User Study on the Success Factors. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (LNCS, Vol. 5795)*, Andy Schürr and Bran Selic (Eds.). Springer, 423–437. https://doi.org/10.1007/978-3-642-04425-0_33

[26] José R. Hoyos, Davy Preuveneers, Jesús J. García-Molina, and Yolande Berbers. 2011. A DSL for Context Quality Modeling in Context-aware Applications. In *Ambient Intelligence-Software and Applications: 2nd International Symposium on Ambient Intelligence (ISAmI)*. Springer, 41–49. https://doi.org/10.1007/978-3-642-19937-0_6

[27] ISO 07498 1994. ISO/IEC 7498-1:1994. Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. https://www.iso.org/standard/20269.html.

[28] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2014. Design Guidelines for Domain Specific Languages. https://arxiv.org/abs/1409.2378. arXiv:1409.2378 [cs.SE]

[29] Siddhartha Kumar Khaitan and James D. McCalley. 2015. Design Techniques and Applications of Cyberphysical Systems: A Survey. *IEEE Systems Journal* 9, 2 (2015), 350–365. https://doi.org/10.1109/JSYST.2014.2322503

[30] Kyoung Dae Kim and Panganamala R. Kumar. 2013. An Overview and Some Challenges in Cyber-Physical Systems. *Journal of the Indian Institute of Science* 93, 3 (July 2013), 341–352. https://journal.iisc.ac.in/index.php/iisc/article/view/1693

[31] Gordon Kindlmann, Charisee Chiw, Nicholas Seltzer, Lamont Samuels, and John Reppy. 2016. Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 867–876. https://doi.org/10.1109/TVCG.2015.2467449

[32] Stefan Klikovits and Didier Buchs. 2021. Pragmatic Reuse for DSML Development. *Software and Systems Modeling* 20, 3 (01 Jun 2021), 837–866. https://doi.org/10.1007/s10270-020-00831-4

[33] Pieter Koopman, Mart Lubbers, and Rinus Plasmeijer. 2018. A Task-Based DSL for Microcomputers. In *Proceedings of the Real World Domain Specific Languages Workshop* (Vienna, Austria) *(RWDSL2018)*. ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3183895.3183902

[34] Aliaksei Kouzel. 2024. *Developing a DSL Design Methodology for CPS Diagnostics.* Bachelor's thesis. Universiteit Twente, Enschede, The Netherlands. http://purl.utwente.nl/essays/100776

[35] Edward A. Lee. 2007. *Computing Foundations and Practice for Cyber-Physical Systems: A Preliminary Report.* Technical Report UCB/EECS-2007-72. Electrical Engineering and Computer Sciences; University of California at Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-72.html

[36] Edward A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 363–369. https://doi.org/10.1109/ISORC.2008.25

[37] Edward A. Lee. 2010. CPS Foundations. In *Proceedings of the 47th Design Automation Conference* (Anaheim, California) *(DAC)*. ACM, New York, NY, USA, 737–742. https://doi.org/10.1145/1837274.1837462

[38] Edward A. Lee. 2015. The Past, Present and Future of Cyber-Physical Systems: A Focus on Models. *Sensors* 15, 3 (2015), 4837–4869. https://doi.org/10.3390/s150304837

[39] Marcos V Linhares, Romulo S. de Oliveira, Jean-Marie Farines, and Francois Vernadat. 2007. Introducing the Modeling and Verification Process in SysML. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (EFTA)*. IEEE, 344–351. https://doi.org/10.1109/EFTA.2007.4416788

[40] Marten Lohstroh, Christian Menard, Alexander Schulz-Rosengarten, Matthew Weber, Jeronimo Castrillon, and Edward A. Lee. 2020. A Language for Deterministic Coordination Across Multiple Timelines. In *Proceedings of the Forum for Specification and Design Languages (FDL)* (Kiel, Germany). IEEE, 1–8. https://doi.org/10.1109/FDL50818.2020.9232939

[41] Lyudmila N. Lyadova, Alexander O. Sukhov, and Marsel R. Nureev. 2021. An Ontology-Based Approach to the Domain Specific Languages Design. In *2021 IEEE 15th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, Baku, Azerbaijan, 1–6. https://doi.org/10.1109/AICT52784.2021.9620493

[42] Haroun Mangal. 2024. *CSPL: A Domain-Specific Language for Modelling the Behaviour of Cyber-Physical Systems*. Bachelor's thesis. Universiteit Twente, Enschede, The Netherlands. https://purl.utwente.nl/essays/101433

[43] Selin Mehmed. 2024. *Domain-Specific Language for Cyber-Physical Systems: A Survey*. Bachelor's thesis. Universiteit Twente, Enschede, The Netherlands. http://purl.utwente.nl/essays/100883

[44] Josh G. M. Mengerink., Bram van der Sanden., Bram C. M. Cappers., Alexander Serebrenik., Ramon R. H. Schiffelers., and Mark G. J. van den Brand. 2018. Exploring DSL Evolutionary Patterns in Practice — A Study of DSL Evolution in a Large-scale Industrial DSL Repository. In *Proceedings of the Sixth International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. INSTICC, SciTePress, 446–453. https://doi.org/10.5220/0006605804460453

[45] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-Specific Languages. *Comput. Surveys* 37, 4 (Dec. 2005), 316–344. https://doi.org/10.1145/1118890.1118892

[46] Mustafa Abshir Mohamed, Geylani Kardas, and Moharram Challenger. 2021. Model-Driven Engineering Tools and Languages for Cyber-Physical Systems —A Systematic Literature Review. *IEEE Access* 9 (2021), 48605–48630. https://doi.org/10.1109/ACCESS.2021.3068358

[47] Daniel Moody. 2009. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* 35, 6 (Nov. 2009), 756–779. https://doi.org/10.1109/TSE.2009.67

[48] Dmitry Morozov, Mario Lezoche, and Hervé Panetto. 2018. Multi-Paradigm Modelling of Cyber-Physical Systems. *IFAC-PapersOnLine* 51, 11 (2018), 1385–1390. https://doi.org/10.1016/j.ifacol.2018.08.334 Proceedings of the 16th IFAC Symposium on Information Control Problems in Manufacturing (INCOM).

[49] Mark A. Musen, Lawrence M. Fagan, David M. Combs, and Edward H. Shortliffe. 1987. Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool. *International Journal of Man-Machine Studies* 26, 1 (1987), 105–121. https://doi.org/10.1016/S0020-7373(87)80039-1

[50] Sascha Julian Oks, Albrecht Fritzsche, and Kathrin M. Möslein. 2017. An Application Map for Industrial Cyber-Physical Systems. In *Industrial Internet of Things: Cybermanufacturing Systems*, Sabina Jeschke, Christian Brecher, Houbing Song, and Danda B. Rawat (Eds.). Springer, Cham, 21–46. https://doi.org/10.1007/978-3-319-42559-7_2

[51] Maria João Varanda Pereira, João Fonseca, and Pedro Rangel Henriques. 2016. Ontological Approach for DSL Development. *Computer Languages, Systems & Structures* 45 (April 2016), 35–52. https://doi.org/10.1016/j.cl.2015.12.004

[52] Leonard Petnga and Mark Austin. 2016. An Ontological Framework for Knowledge Modeling and Decision Support in Cyber-Physical Systems. *Advanced Engineering Informatics* 30, 1 (2016), 77–94. https://doi.org/10.1016/j.aei.2015.12.003

[53] Jurryt Pietersma, Arjan J. C. van Gemund, and A. Bos. 2005. A Model-Based Approach to Sequential Fault Diagnosis. In *IEEE Autotestcon*. IEEE, 621–627. https://doi.org/10.1109/AUTEST.2005.1609208

[54] Subhav M. Pradhan, Abhishek Dubey, Aniruddha Gokhale, and Martin Lehofer. 2015. CHARIOT: A Domain Specific Language for Extensible Cyber-Physical Systems. In *Proceedings of the Workshop on Domain-Specific Modeling* (Pittsburgh, PA, USA) *(DSM)*. ACM, 9–16. https://doi.org/10.1145/2846696.2846708

[55] Ragunathan Rajkumar. 2012. A Cyber–Physical Future. *Proc. IEEE* 100, Special Centennial Issue (2012), 1309–1312. https://doi.org/10.1109/JPROC.2012.2189915

[56] Michael Rosemann, Iris Vessey, Ron Weber, and Boris Wyssusek. 2004. On the Applicability of the Bunge-Wand-Weber Ontology to Enterprise Systems Requirements. In *Proceedings of ACIS*. AISeL, 10 pages. https://aisel.aisnet.org/acis2004/78

[57] Enno Ruijters and Mariëlle Stoelinga. 2015. Fault Tree Analysis: A Survey of the State-of-the-Art in Modeling, Analysis and Tools. *Computer Science Review* 15 (2015), 29–62. https://doi.org/10.1016/j.cosrev.2015.03.001

[58] Eric C. Schnarr, Mark D. Hill, and James R. Larus. 2001. Facile: A Language and Compiler for High-Performance Processor Simulators. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation* (Snowbird, Utah, USA) *(PLDI)*. ACM, New York, NY, USA, 321–331. https://doi.org/10.1145/378795.378864

[59] Robert W. Sebesta. 2001. *Concepts of Programming Languages*. Pearson.

[60] Jonathan Sprinkle and Gabor Karsai. 2004. A Domain-Specific Visual Language for Domain Model Evolution. *Journal of Visual Languages & Computing* 15, 3 (2004), 291–307. https://doi.org/10.1016/j.jvlc.2004.01.006

[61] Tim Steenvoorden, Nico Naus, and Markus Klinik. 2019. TopHat: A Formal Foundation for Task-Oriented Programming. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming* (Porto, Portugal) *(PPDP)*. ACM, New York, NY, USA, Article 17, 13 pages. https://doi.org/10.1145/3354166.3354182

[62] Mariëlle Stoelinga et al. 2023. Zorro: Zero Downtime in Cyber-Physical Systems. https://zorro-project.nl.

[63] Robert Tairas, Marjan Mernik, and Jeff Gray. 2009. Using Ontologies in the Domain Analysis of Domain-Specific Languages. In *Models in Software Engineering*, Michel R. V. Chaudron (Ed.). LNCS, Vol. 5421. Springer, Berlin, Heidelberg, 332–342. https://doi.org/10.1007/978-3-642-01648-6_35

[64] Amit Kumar Tyagi and N. Sreenath. 2021. Cyber Physical Systems: Analyses, Challenges and Possible Solutions. *Internet of Things and Cyber-Physical Systems* 1 (2021), 22–33. https://doi.org/10.1016/j.iotcps.2021.12.002

[65] Boris Ulitin and Eduard Babkin. 2017. Ontology and DSL Co-evolution Using Graph Transformations Methods. In *Perspectives in Business Informatics Research*, Björn Johansson, Charles Møller, Atanu Chaudhuri, and Frantisek Sudzina (Eds.). LNBIP, Vol. 295. Springer, Cham, 233–247. https://doi.org/10.1007/978-3-319-64930-6_17

[66] Michael Vierhauser, Rebekka Wohlrab, Marco Stadler, and Jane Cleland-Huang. 2023. AMon: A Domain-Specific Language and Framework for Adaptive Monitoring of Cyber–Physical Systems. *JSS* 195 (2023), 111507. https://doi.org/10.1016/j.jss.2022.111507

[67] Yair Wand and Ron Weber. 1990. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering* 16, 11 (Nov. 1990), 1282–1292. https://doi.org/10.1109/32.60316

[68] Andrzej Wąsowski and Thorsten Berger. 2023. *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*. Springer. https://doi.org/10.1007/978-3-031-23669-3

[69] Bradley Wood and Akramul Azim. 2021. Triton: a Domain Specific Language for Cyber-Physical Systems. In *Proceedings of the 22nd IEEE International Conference on Industrial Technology (ICIT)*, Vol. 1. IEEE, 810–816. https://doi.org/10.1109/ICIT46573.2021.9453575

[70] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering* 44, 10 (2018), 951–976. https://doi.org/10.1109/TSE.2017.2734091

[71] Vadim Zaytsev. 2017. Language Design with Intent. In *Proceedings of the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, Don Batory, Jeff Gray, and Vinay Kulkarni (Eds.). IEEE, 45–52. https://doi.org/10.1109/MODELS.2017.16

[72] Vadim Zaytsev. 2017. Megamodelling with NGA Multimodels. In *Proceedings of the 2nd International Workshop on Comprehension of Complex Systems*, Christoph Bockisch and Michael L. Van De Vanter (Eds.). ACM, 1–6. https://doi.org/10.1145/3141842.3141843

[73] Vadim Zaytsev and Anya Helene Bagge. 2014. Parsing in a Broad Sense. In *Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014) (LNCS, Vol. 8767)*, Jürgen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfran (Eds.). Springer, 50–67. https://doi.org/10.1007/978-3-319-11653-2_4