Domain-Specific Languages for Cyber-Physical Systems: A Survey

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

ABSTRACT

Cyber-physical systems are systems that are interactions of computation with physical processes. They are incredibly complex and multidisciplinary, and thus benefit from models in order to achieve requirements such as interoperability and reliability. A model in the form of a domain-specific language (DSL) presents itself as an attractive solution to managing the complexity of CPS as it is geared towards a more specific application domain and can solve domain-specific issues. This paper looks at concerns a CPS model should address, proposes necessary features a DSL for CPS should have, and analyses existing DSLs and the concrete features they implement (such as task scheduling). Its aim is to be a starting point for creating DSLs by identifying some of the overarching things that it should be able to do, present concrete features and capabilities, and give an overview of existing work.

KEYWORDS

Cyber-physical systems, Domain-specific languages, MDE, Survey

1 INTRODUCTION

Cyber-physical systems are interactions of computation and physical processes. More specifically, cyber systems control or monitor physical processes, which requires the use of components like sensors and actuators. Sensors sense information about the physical environment and relay it back to the cyber system, whereas actuators affect the physical environment in some way upon being required to so by the computational elements of the system. This leads to a feedback loop between the physical and cyber.

Cyber-physical systems have many potential application domains and benefits [18, 34, 38]. Such domains are smart manufacturing, robotics, healthcare and medicine, intelligent transportation, smart cities etc. For example, in healthcare and medicine specifically some potential applications are "proton therapy machines, electro-anatomic mapping and intervention, bio-compatible and implantable devices, and robotic prosthetics" [43]. The potential benefits of CPS cannot be overstated; fully autonomous vehicles could transport passengers safely with near zero fatalities, reduced car accidents, and less traffic, smart power grids could lead to better energy distribution and less blackouts, manufacturing could become highly automated etc [38, 43].

However, there are many obstacles to fully harnessing the benefits of CPS. The main issue is the inherent complexity and heterogeneity of CPS due to the integration of the physical and cyber worlds. Models and model-driven engineering (MDE) are currently utilised to deal with the complexity as models make it possible to break down complex systems and make concerns understandable and analysable [11]. Despite this, many difficulties in modelling remain to be addressed. For example, the physical world is continuous and concurrent, whereas the cyber one is discrete and sequential; integrating them is not straightforward and can lead to non-deterministic behaviour in models [12, 30]. This does not lead to reliable and dependable models because the outcome cannot be predicted.

One approach to modelling is domain-specific languages (DSL). Difficulties in modelling CPS can be mitigated by using more specialised, domain-specific models and modelling languages with well-defined semantics [12]. Domain-specific languages are languages meant to handle a specific type of problem or be used in a specific domain. SQL (for databases) and HTML (for creating web pages) are well-known and widely used domain-specific languages [48]. The advantage comes from their ability to specify domain-specific notations and constructs, and more easily implement actions necessary in that domain rather than going through the effort of finding a general-purpose abstraction.

This paper seeks to answer the research question of what the minimum necessary features that a DSL for CPS should have are. Based on concerns CPS models should address, it identifies some overarching features that a DSL should have. Following this, it goes through existing DSLs that have been created and identifies concrete features that they have. Its aim is to be a starting point for creating a DSL by identifying some of the overarching things that it should be able to do such as having timing semantics, present concrete features and capabilities such as real-time task scheduling, and give an overview of existing work.

The paper is structured as follow. Section 2 gives a definition for CPS and an overview of the difficulties when modelling CPS. Section 3 describes the advantages and disadvantages, and development of DSLs, and additionally gives a brief justification for creating a DSL for CPS and a proposed approach for doing so. Section 4 analyses features that a DSL for CPS might have, gives motivation for implementing such features, and primarily looks at the different

TScIT 41, July 5, 2024, Enschede, The Netherlands
© 2022 University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ways in which existing DSLs incorporates those features. Lastly, section 5 looks at related work.

2 CYBER-PHYSICAL SYSTEMS

2.1 Definition

Various definitions for cyber-physical systems have been proposed. Lee [29] describes them as "interactions of computation with physical processes". Gunes et al. [18] summarise various definitions, such as "physical and engineered systems, whose operations are monitored, coordinated, controlled, and integrated by a computing and communicating core", "embedded systems together with their physical environment", and "physical, biological, and engineered systems whose operations are integrated, monitored, and/or controlled by a computational core. Components are networked at every scale. Computing is deeply embedded into every physical component, possibly even into materials. The computational core is an embedded system, usually demands real-time response, and is most often distributed". What is common across definitions is that there is a cyber part and a physical part, and the cyber part can control or monitor the physical part in some way. Thus, the definition that we stick to in this paper is that CPS are a combination of computational and physical elements and processes where the computational parts control or monitor in some way the physical ones.

More concretely, a CPS consists of "the physical world, interfaces, and cyber systems" [18]. The physical world refers to all the physical phenomena that are to be monitored and controlled, the cyber systems are the devices that process data, and the interfaces are what allow the physical world and the cyber systems to communicate. Examples of interfaces are sensors, actuators, analog-to-digital converters (ADC), digital-to-analog converters (DAC) etc.

2.2 Modelling

Cyber-physical systems are complex and multidisciplinary, and thus challenging to build. This is an issue because the number of potential applications of CPS is staggering - from healthcare to agriculture to city design, the possible societal impact and economic benefit cannot be overstated. Models are a promising solution to addressing this as they make it possible to break down complex systems and make concerns understandable and analysable [11]. Models in software engineering can serve various purposes from code generation, system documentation, construction of the system, exploration of solution possibilities etc. [11].

Important requirements for cyber-physical systems are interoperability, predictability, reliability, and dependability [18]. Interoperability refers to the ability of different components to work together. Dependability, reliability, and predictability 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 various 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 when it comes to modelling CPS is due to 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 [22]. Thus, different modelling methods are utilised for the two parts of the system. For example, continuous-time models of dynamics model physical processes and state machines model computations. However, integrating these two modelling paradigms is difficult [29] because it can lead to incompatibilities during the separate design processes or non-deterministic behaviour [12, 30]. Therefore, this is an issue that a CPS must address. For example, a CPS can be modeled as a hybrid system [12, 22].

An essential problem for CPS is that most models are unequipped to deal with timing semantics. Such semantics are crucial [22, 28, 30] due to the aforementioned 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 [22, 28] - 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, a C program by itself provides no meaningful time 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 of the key requirements for a CPS model is that it must be predictable, which means it needs to have exact expressions of time rather than delegating them to the implementation (as one would with a C program).

Furthermore, 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 [12]. This also necessitates solutions such as communication semantics, synchronous or asynchronous message transmissions and timestamps.

CPS are also very complex and made up of multiple components all interacting together. Components are highly interconnected and models of a CPS might grow more complex with time [12]. In order to meet the previously mentioned requirements, a model should make it possible to ensure those components work together as intended. Furthermore, a given model must be reliable when small deviations from the expected operation occur [28]. Dealing with unexpected deviations can happen at higher or lower levels, and a CPS model must contain enough knowledge in order to be reliable [27] especially since many CPS are safety-critical systems [22]. The most obvious example of safety-critical systems are medical devices.

3 DOMAIN SPECIFIC LANGUAGES

3.1 Advantages and Disadvantages

There are various benefits to using a domain-specific language (DSL) over a general-purpose programming language (GPL). DSLs trade versatility for expressiveness in a specific domain. They offer domain-specific notations, incorporate domain-specific constructs in a way that allows for easier definition and traversal, and make it easier to program tasks that might be tedious and require workarounds in a GPL by offering the needed code generation [33]. All of this reduces the amount of domain knowledge and programming expertise needed when working with the domain in question, which in turn leads to increased productivity and reduced maintenance costs.

The main drawback of a DSL is the *initial* development process. In order to design and implement a DSL, both extensive domain knowledge and language development expertise is needed [33, 42]. Since the use of a DSL is by definition more limited than a GPL, it might further put in question whether the development costs are worth it. Furthermore, careful examination is needed whether a DSL offers the benefits that make it preferable to using a GPL with its associated workarounds for any arising limitations. Ultimately, the benefits and drawbacks must be evaluated on a case by case basis and might not be immediately apparent.

3.2 Development

The development of a DSL requires multiple steps. This section presents a rough guideline. First, a decision must be made about the trade offs of developing a DSL versus other existing methods. If a DSL has significant advantages over other methods, the problem domain must be identified and analysed. Following this is the design and implementation stages.

Before creating the DSL, the domain knowledge needs to be captured and analysed. Domain analysis involves gathering domain knowledge and constructing a domain model including the scope, terminology, features, concepts, dependencies of said concepts, semantics etc. [33]. Building an ontology is one established way of doing so [9]. An ontology consists of the representative vocabulary of the domain (such as concepts and features), i.e. a set of terms defining domain concepts, and a body of knowledge of the domain using this representative vocabulary, i.e. a collection of facts about the domain [9]. Tairas et al. evaluate the usefulness of an ontology in defining DSLs [42]. Walter et al. [47] propose an ontology-based framework for defining DSLs.

There are a few choices to make in regards to DSL design and implementation. One can base the design of a DSL on an existing programming language and create an internal DSL, or choose not to do so and create an external DSL [33]. The former option is the simpler one, but the latter offers more novelty. Furthermore, a design might be formal or informal. An informal design is represented by natural language or some form of illustration, whereas a formal one includes specifications and semantics such as a grammar. A formal design simplifies the implementation process later on and can bring potential problems to light earlier. To make the transition between ontology and grammar, there is the following approach [35]. In terms of implementation, various implementation approaches exist [26]. For example, a DSL might be embedded in another language such as Haskell or some compiler/interpreter approach used.

A language has both syntax and semantics. Syntax refers to the way sentences are constructed in a language [19, 48]. In the context of programming languages, this could be the grammar of the language. Semantics on the other hand is the study of meanings in the language [19, 48]. Semantics consist of a semantic domain and semantic mapping, where the domain is the collection of elements and concepts that make up the language's meanings and the mapping is what relates the syntax to the domain [19]. This presents an alternative approach to creating a DSL - that is, one can begin by constructing the semantic domain of the language and working backwards from there [14]. For example, a calendar application would offer the ability to define appointments at particular times. This means that both "times" and "appointments" are essential concepts of the semantic domain and also that they must be mapped to each other.

3.3 Justification

While many approaches to modelling CPS exist, a DSL is an attractive one for various reasons. First, difficulties in modelling CPS can be mitigated by using more specialised, domain-specific models and modelling languages with well-defined semantics [12]. Furthermore, working with cyber-physical systems requires working with a low-level environment as high-level languages lack crucial features especially related to timing [49], which makes it more complicated for developers and requires extensive domain knowledge. A DSL would make it possible to implement such features on a higher level of abstraction, simplifying the development process.

3.4 Proposed Approach

The first step in developing a DSL for CPS is domain analysis. As mentioned previously, ontologies are an established way to do so and there are works on approaches for cyber-physical systems specifically. Petnga and Austin [36] present a framework for creating models and domain-specific semantics for CPS. First they separate CPS into three ontologies: physical, cyber, and meta ontology (for any concepts that don't fall under either physical or cyber such as time and space). The ontologies are then integrated using an "integrator ontology" and logic-based rules. Voinov and Senokosov [46] propose a high-level CPS ontology where various interactions are defined. They include definitions for concepts such as resources, space, time, events etc. Hildebrandt et al. [20] have developed a domain-centric approach for developing a CPS ontology. It consists of requirements specification and lightweight ontology building by domain experts and end users, and heavyweight ontology building by ontology experts. In the heavyweight ontology building phase, the ontology is explicitly defined in Ontology Web Language (OWL). Lastly Dillon et al. [13] analyse the key requirements that must be met by CPS systems and propose semantics based on said requirements. They focus on the semantics of sensors and events, integrating them through semantic frameworks and models.

Building an ontology might be aided by considering domain ontologies that experts use but also ontologies of modelling languages that describe existing languages and tasks that they solve [32]. Analysing languages helps identify trends and patterns can provide a better foundation for building a CPS ontology. In the next section we shift our focus to the analysis of languages.

4 ENGINEERING A CPS LANGUAGE

This section gives an overview of some important features, motivation for these features, and existing languages that implement them. These features are as follows: a DSL should be able to specify how the components work together, be able to define the flow of operation, have timing semantics, and have a way to model both the physical and cyber parts of the system (i.e. integrate the continuous and discrete). These features are not fully separable - timing is directly related to the flow of operation rather than being totally distinct from it for instance - so there might be overlaps between existing work. Nonetheless, this section will focus on them as mostly distinct.

4.1 Specify How Components Work Together

Interoperability is one of the crucial requirements for CPS [18]. CPS is a complex and intertwined system; in order for it to function correctly the separate components must work together as intended. A DSL for CPS can 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 [44]. 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 [37] is a DSL with a focus on clean separation-of-concerns between computation and communication aspects and explicit definitions for systems goals, objectives and associated functionalities. As mentioned previously, communication is also tricky in a CPS [12]. Chariot enforces a clean separation between communication logic and computation logic in order to support heterogeneous communication middleware. Different communication patterns can be modelled without having to worry about the middleware that will support these interactions. It can also model the state of the entire system and different resources available, well known faults, system goals, objectives and corresponding functionalities.

For the integration of sensors specifically, there is SensOr Interfacing Language (SOIL) [5]. 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 [6] 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. The DSL can list components, the dependencies of that component i.e. what other components that component depends on, and any constraints that must be true. A less relevant feature is the ability to define probes, which collect data about the system for the purpose of deployment.

Chauhan et al. [10] develop a framework for programming CPS with modelling languages. They aim to deal with issues such as complexity due to CPS consisting of various entities like sensors and actuators, differences in platforms that components run on, and the different types of interactions components can have. In the framework, one can specify domain-specific constructs such as sensors, actuators, tags (any physical object that can act as identification), and storage. Sensors produce measurements with specified datatypes and can be periodic, event-driven, or request-based. Period sensors sample results periodically every few seconds as specified, event-based sensors sample results when a specific event occurs, and request-based sensors sample data only when the user makes a request. One can furthermore specify computational services. This includes using a certain measurement in order to generate some result (such as computing average temperature), issuing requests to access something in the system, commands (such as setting a certain temperature). Lastly, one can define user interactions (for example, sending notifications to the user) and deployment specifications. Overall, the framework has features to define individual components, types of sensors, how to perform computations, various interactions between system components and between user and system, 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 trees might also be used to better model interactions between components of the system.

4.2 Define Flow of Operation

In a CPS, computational elements control or monitor some 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 DSL can define the appropriate action (such as shutting down the system in order to prevent overheating) or implement constraints (the maximum temperature that is safe).

AMon [45] is a DSL that monitors different states and provides definitions for the data flow within the CPS. It makes it possible to define various rules for the system. For example, if the battery voltage level falls under a certain level specified actions can be defined and executed as appropriate. Another feature of the language is the ability to specify which devices check for which rules and monitor what data. Furthermore, rules can apply to certain devices or the entire system. AMon is ideal for adaptive monitoring of cyber-physical systems and defining the general flow of data between components. It implements rules to certain changes in the physical environment and what part of the system should be affected, and makes it possible to specify how data is processed and how often it's sampled.

A DSL for context-aware systems is developed in [21]. Context-aware systems are systems that interpret the context and modify the system based on it. Their similarity to CPS comes from the fact that they make use of context sensors. In both cases, sensors sense information about the environment and the system must take appropriate action based on that. The proposed language achieves this by modelling entities (people or objects) and their context with the option to specify attributes of that context (such as the location and time) and the source of that context (for example GPS or a clock). The sources of context have a provider with certain accuracy, units, and so on which implements methods for getting a specific attribute. Lastly, it can specify situations in which based on context facts some action can be undertaken. Additionally, the DSL can deal with the problem of sensors not being 100 percent reliable. It has a notion of context quality and helps select the most reliable sources and trigger the appropriate behaviour as a result.

Another option is task-oriented programming. Steenvoorden et al. [41] give a formalisation of task-oriented programming. Tasks are interactive units of work based on information sources. Koopman et al. [25] present an example DSL. It uses light-weight 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.

4.3 Have 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 [49] is a DSL with real-time scheduling. It defines scheduling blocks which contain tasks and are parameterised 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. Triton also fits feature 2 as it's another language that implements tasks and constraints while also having basic timing semantics.

Lohstroh et al. [31] propose 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 one-time 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 [15] present another DSL for specifying different timing requirements is developed. It supports 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 refers 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 furthermore 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 feature in 4.2 because timing semantics arise precisely due to interactions with the physical world [8], especially when basing timing on a certain event in the physical world.

4.4 Model Hybrid Systems

A DSL must have a way to capture what is happening in the physical part of the system. However, as mentioned previously, 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 some solutions exist.

CREST [24] is a DSL 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 [17], an executable domain-specific language that models the hybrid behaviour of cyber-physical systems. For example, states in the model are captured also by ordinary differential equations (ODEs) in order to model the continuous behaviour of physical processes. It also has semantics for representing transitions between states and physical environment variables.

Diderot [23] 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, which languages are a concrete implementation of [8]. For example, xSHS made use of ODEs [17] and CREST made use of hybrid automata [24]. Implementing this last feature requires expertise on modelling physical systems as opposed to concrete features that can be described semantically.

5 RELATED WORK

Beyond DSLs, timed automata are a formalism to modelling time in applications. Timed automata are finite-state machines extended with clock variables which allows one to capture quantitative continuous-time properties [8] that arise in CPS. UPPAAL [4] specifically is a language that implements the timed automata formalism and it models a system as a network of timed automata in parallel. It also makes it possible to specify discrete variables as in a regular programming language, and the values of these variables can be used to define the state of the system.

There are other helpful resources to keep in mind when designing a DSL for CPS. Broman et al. [7] give an overview of the challenges and other existing approaches in regards to time in CPS. Shrivastava et al. [40] explain timing-related challenges in CPS development and give insights as to the limitations of current approaches. Baillieul and Antsaklis [3] describe issues involved in designing successful networked real-time systems. Sanfelice [39] gives a hybrid systems approach to the analysis and design of CPS.

Modelling a CPS using a DSL is not the only approach. Graja et al. [16] give an overview on various modelling techniques of CPS. Many different architectures for CPS also exist, depending on system requirements and application details [1]. Some examples are given by Yu et al. [50] and Ahmed et al. [2].

6 CONCLUSION

The goal of this paper was to identify relevant information about domain-specific languages for cyber-physical systems. We began with introducing CPS and some of the difficulties of modelling CPS in order to understand what problems a model should address. A CPS model must meet the requirements of interoperability, dependability, reliability, and predictability, and must be able to handle timing and the integration of the continuous physical world and the discrete cyber one. We similarly gave an overview of DSLs - what the advantages and disadvantages are, how one develops a DSL, some existing approaches to developing DSLs for CPS. We found that DSLs often domain-specific notations and constructs, but are difficult to develop as they require domain expertise. Their development consists of a decision, domain analysis, design, and implementation phases. We investigated the domain analysis phase in particular and explored ontologies, and talked about some CPS-specific frameworks. Following this we identified overarching necessary features that a DSL for CPS should have - namely, it should make it possible to specify how components interact together because CPS are composed of various components and subsystems, define the flow of operation in regards to how the system reacts to the physical environment, have timing semantics, and integrate the continuous physical world with the discrete cyber one. Various features appeared in a lot of languages, such as constraints and rules, scheduling the order and timing of tasks by inducing delays and periodic tasks, and ability to specify existing components and their dependencies.

REFERENCES

[1] Mohamed Anis Aguida, Samir Ouchani, and Mourad Benmalek. 2020. A Review on Cyber-Physical Systems: Models and Architectures. In *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 275–278. https://doi.org/10.1109/WETICE49692.2020.00060

[2] Syed Hassan Ahmed, Gwanghyeon Kim, and Dongkyun Kim. 2013. Cyber Physical System: Architecture, applications and research challenges. In *2013 IFIP Wireless Days (WD)*. 1–5. https://doi.org/10.1109/WD.2013.6686528

[3] John Baillieul and Panos J. Antsaklis. 2007. Control and Communication Challenges in Networked Real-Time Systems. *Proc. IEEE* 95, 1 (2007), 9–28. https://doi.org/10.1109/JPROC.2006.887290

[4] Gerd Behrmann, Alexandre David, and Kim G. Larsen. 2004. *A Tutorial on Uppaal.* Springer Berlin Heidelberg, Berlin, Heidelberg, 200–236. https://doi.org/10.1007/978-3-540-30080-9_7

[5] 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, Berlin, Heidelberg, 448–456.

[6] 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 *2014 IEEE 38th International Computer Software and Applications Conference Workshops*. 708–715. https://doi.org/10.1109/COMPSACW.2014.120

[7] David Broman, Patricia Derler, and John Eidson. 2013. Temporal Issues in Cyber-Physical Systems. *Journal of the Indian Institute of Science* 93 (07 2013), 389–402.

[8] David Broman, Edward A. Lee, Stavros Tripakis, and Martin Törngren. 2012. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling* (Innsbruck, Austria) *(MPM '12)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/2508443.2508452

[9] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. 1999. What are ontologies, and why do we need them? *IEEE Intelligent Systems and their Applications* 14, 1 (1999), 20–26. https://doi.org/10.1109/5254.747902

[10] Saurabh Chauhan, Pankesh Patel, Flávia C. Delicato, and Sanjay Chaudhary. 2016. A development framework for programming cyber-physical systems. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems* (Austin, Texas) *(SEsCPS '16)*. Association for Computing Machinery, New York, NY, USA, 47–53. https://doi.org/10.1145/2897035.2897039

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

[12] 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

[13] Tharam Dillon, Elizabeth Chang, Jaipal Singh, and Omar Hussain. 2012. Semantics of Cyber-Physical Systems. In *Intelligent Information Processing VI*, Zhongzhi Shi, David Leake, and Sunil Vadera (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–12.

[14] Martin Erwig and Eric Walkingshaw. 2012. Semantics First!. In *Software Language Engineering*, Anthony Sloane and Uwe Aßmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 243–262.

[15] Arda Goknil and Marie-Agnès Peraldi-Frati. 2012. A DSL for specifying timing requirements. In *2012 Second IEEE International Workshop on Model-Driven Requirements Engineering (MoDRE)*. 49–57. https://doi.org/10.1109/MoDRE.2012.6360074

[16] Imen Graja, Slim Kallel, Nawal Guermouche, Saoussen Cheikhrouhou, and Ahmed Hadj Kacem. 2020. A comprehensive survey on modeling of cyber-physical systems. *Concurrency and Computation: Practice and Experience* 32, 15 (2020), e4850. https://doi.org/10.1002/cpe.4850 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4850 e4850 cpe.4850.

[17] 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 *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 683–687. https://doi.org/10.1109/APSEC.2018.00090

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

[19] David Harel and Bernhard Rumpe. 2004. Meaningful modeling: What's the semantics of "semantics"? *Computer* 37 (11 2004), 64 – 72. https://doi.org/10.1109/MC.2004.172

[20] C. Hildebrandt, S. Törsleff, B. Caesar, and A. Fay. 2018. Ontology Building for Cyber-Physical Systems: A domain expert-centric approach. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. 1079–1086. https://doi.org/10.1109/COASE.2018.8560465

[21] 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*, Paulo Novais, Davy Preuveneers, and Juan M. Corchado (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 41–49.

[22] K.-D Kim and Panganamala Kumar. 2013. An Overview and Some Challenges in Cyber-Physical Systems. *Journal of the Indian Institute of Science* 93 (07 2013), 341–352.

[23] 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

[24] Stefan Klikovits and Didier Buchs. 2020. Pragmatic reuse for DSML development: Composing a DSL for hybrid CPS modeling. *Software and Systems Modeling* 20, 3 (Oct. 2020), 837–866. https://doi.org/10.1007/s10270-020-00831-4

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

[26] Tomaž Kosar, Pablo E. Martı ́nez López, Pablo A. Barrientos, and Marjan Mernik. 2008. A preliminary study on various implementation approaches of domain-specific language. *Information and Software Technology* 50, 5 (2008), 390–405. https://doi.org/10.1016/j.infsof.2007.04.002

[27] Edward Lee. 2007. Computing Foundations and Practice for Cyber Physical Systems: A Preliminary Report. (01 2007).

[28] 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)*. 363–369. https://doi.org/10.1109/ISORC.2008.25

[29] Edward A. Lee. 2010. CPS foundations. In *Proceedings of the 47th Design Automation Conference* (Anaheim, California) *(DAC '10)*. Association for Computing Machinery, New York, NY, USA, 737–742. https://doi.org/10.1145/1837274.1837462

[30] 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

[31] 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 *2020 Forum for Specification and Design Languages (FDL)*. 1–8. https://doi.org/10.1109/FDL50818.2020.9232939

[32] 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)*. 1–6. https://doi.org/10.1109/AICT52784.2021.9620493

[33] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.* 37, 4 (dec 2005), 316–344. https://doi.org/10.1145/1118890.1118892

[34] Sascha Julian Oks, Albrecht Fritzsche, and Kathrin M. Möslein. 2017. *An Application Map for Industrial Cyber-Physical Systems*. Springer International Publishing, Cham, 21–46. https://doi.org/10.1007/978-3-319-42559-7_2

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

[36] 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

[37] 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 2015)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/2846696.2846708

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

[39] Ricardo G. Sanfelice. 2015. Analysis and Design of Cyber-Physical Systems: A Hybrid Control Systems Approach. https://api.semanticscholar.org/CorpusID:29069010

[40] Aviral Shrivastava, Patricia Derler, Ya-Shian Li Baboud, Kevin Stanton, Mohammad Khayatian, Hugo A. Andrade, Marc Weiss, John Eidson, and Sundeep Chandhoke. 2016. Time in cyber-physical systems. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Pittsburgh, Pennsylvania) *(CODES '16)*. Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/2968456.2974012

[41] 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 '19)*. Association for Computing Machinery, New York, NY, USA, Article 17, 13 pages. https://doi.org/10.1145/3354166.3354182

[42] 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.). Springer Berlin Heidelberg, Berlin, Heidelberg, 332–342.

[43] 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

[44] 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 *2018 13th System of Systems Engineering Conference, SoSE 2018 (2018 13th System of Systems Engineering Conference, SoSE 2018)*. Institute of Electrical and Electronics Engineers Inc., United States, 225–232. https://doi.org/10.1109/SYSOSE.2018.8428770

[45] 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. *Journal of Systems and Software* 195 (2023), 111507. https://doi.org/10.1016/j.jss.2022.111507

[46] Artem Voinov and Ilya Senokosov. 2021. Ontological models of cyber physical systems. *Journal of Physics: Conference Series* 1889, 2 (apr 2021), 022064. https://doi.org/10.1088/1742-6596/1889/2/022064

[47] Tobias Walter, Fernando Silva Parreiras, and Steffen Staab. 2009. OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In *Model Driven Engineering Languages and Systems*, Andy Schürr and Bran Selic (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 408–422.

[48] Andrzej Wąsowski and Thorsten Berger. 2023. *Building Modeling Languages*. Springer International Publishing, Cham, 25–46. https://doi.org/10.1007/978-3-031-23669-3_2

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

[50] Chengyuan Yu, Song Jing, and Xuan Li. 2012. An Architecture of Cyber Physical System Based on Service. In *2012 International Conference on Computer Science and Service System*. 1409–1412. https://doi.org/10.1109/CSSS.2012.355