Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Review:

A survey of model-driven techniques and tools for cyber-physical systems*

Bo LIU†1, Yuan-rui ZHANG[1], Xue-lian CAO[1], Yu LIU[1], Bin GU†‡2,3, Tie-xin WANG†‡4

[1]*RISE, School of Computer and Information Science, Southwest University, Chongqing 400715, China*

[2]*Beijing Institute of Control Engineering, Beijing 100190, China*

[3]*School of Computer Science, Northwestern Polytechnical University, Xi'an 710029, China*

[4]*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China*

†E-mail: liubocq@swu.edu.cn; gubinbj@sina.com; tiexin.wang@nuaa.edu.cn

Received June 29, 2020; Revision accepted Sept. 29, 2020; Crosschecked Oct. 22, 2020

Abstract: Cyber-physical systems (CPSs) have emerged as a potential enabling technology to handle the challenges in social and economic sustainable development. Since it was proposed in 2006, intensive research has been conducted, showing that the construction of a CPS is a hard and complex engineering process due to the nature of integrating a large number of heterogeneous subsystems. Among other approaches to dealing with the complex design issues, model-driven design of CPSs has shown its advantages. In this review paper, we present a survey of research on model-driven development of CPSs. We are concerned mainly with the widely used methods, techniques, and tools, and discuss how these are applied to CPSs. We also present comparative analyses on the surveyed techniques and tools from various perspectives, including their modeling languages, functionalities, and the challenges which they address in CPS design. With our understanding of the surveyed methods, we believe that model-driven approaches are an inevitable choice in building CPSs and further research effort is needed in the development of model-driven theories, techniques, and tools. We also argue that a unified modeling platform is needed. Such a platform would benefit research in the academic community and practical development in industry, and improve the collaboration between these two communities.

Key words: Cyber-physical systems; Model-driven approach; System modeling; Software engineering
https://doi.org/10.1631/FITEE.2000311 **CLC number:** TP311

1 Introduction

Cyber-physical systems (CPSs) have emerged in recent decades as one of the most attractive realms in both academic and industrial information and communication technology (ICT) communities. The term CPS was coined in 2006 by Helen Gill at the National Science Foundation (NSF) in the USA, which can be simply interpreted in the past as networks of distributed embedded systems. CPSs are now combined with the emerging ICTs including Internet of Things (IoT), cloud computing, and big data. CPS is also considered as a key enabling technology to the major strategic plans in industrial countries, such as the "Industrial Internet of Things (USA)," "Industry 4.0 (Europe)," and "Made in China 2025."

As pointed out in Gill (2008), a CPS is architecturally seen as an integration of physical, biological, and engineered systems, whose operations are monitored and controlled by a computational core. In such an integration, the computing and

‡ Corresponding authors

* Project supported by the Special Foundation for Basic Science and Frontier Technology Research Program of Chongqing, China (No. cstc2017jcyjAX0295), the Capacity Development Foundation of Southwest University, China (No. SWU116007), and the National Natural Science Foundation of China (Nos. 62032019, 61732019, 61672435, and 61811530327)

ORCID: Bo LIU, https://orcid.org/0000-0002-9026-2543

communication capabilities are embedded in all types of objects and structures in the physical environment (Rajkumar et al., 2010). Applications of CPSs are of full range of industrial, social, economic, human living, and scientific applications, such as CPS-based manufacturing systems, health and medical systems, intelligent transportation systems, emergency relief systems, renewable energy systems, distributed robotic systems, and aerospace systems (Rajkumar et al., 2010; Lee J et al., 2015).

With big data technology and cloud computing platforms, CPSs nowadays are formed as integrations of cross-domain subsystems or even (sub-)CPSs. Their scales are growing beyond the traditional ICT systems. Furthermore, the subsystems and components in the integration are developed in different technology paradigms, controlled and managed in their local environment. Therefore, interoperability integration and control of these heterogeneous systems and components are extremely challenging. It is easy to understand that the trial-and-error approach will not be able to deal with the increasingly chronic complexity of scale and complex heterogeneity, and at the same time to meet the requirements for trustworthiness, performance, and quality of service (QoS), including safety, security, reliability, robustness, adaptivity, and realtime. Indeed, we do see that industrial CPS applications are designed and running, but they are designed and maintained in ad-hoc processes using best-effort approaches (Rajkumar et al., 2010; Liu ZM et al., 2019). They are very costly, not reliable, and not easy to maintain. Their construction processes are not repeatable, and their components are not reusable. Therefore, new theories and methods of modeling and design are required (Lee EA, 2008; Liu ZM and Chen, 2016).

In recent decades, a wide consensus has been reached that model-driven approaches will be the most systematic and effective in engineering CPSs (Lee EA, 2008; Sha et al., 2008; Rajkumar et al., 2010; Derler et al., 2012; Sangiovanni-Vincentelli et al., 2012). The essence of model-driven development is that the whole process of system construction is based on model construction, transformations, and model analysis. This has been proven to be successful in traditional software-intensive systems development. For a thorough historical discussion about model-driven software development, we refer readers to Liu ZM and Chen (2016).

A model produced at a time in a model-driven development process provides the abstraction to reduce the complexity of a system (or a problem) and allow people to focus on the problems of the design at that time. Therefore, a large number of models for different concerns and at different levels of abstraction are constructed and analyzed during the development process. Fig. 1 shows the sorts of models usually produced with modeling tools, manually or automatically, in a model-driven software development process.

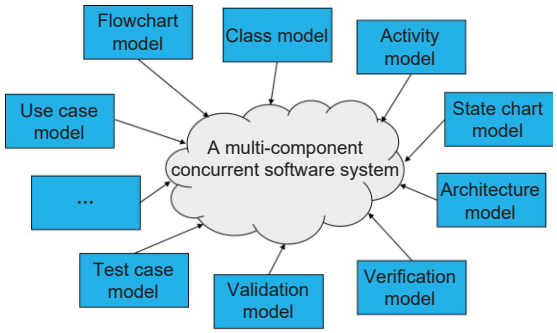

Fig. 1 Abstraction of model-driven approaches

Model-driven development of CPSs involves modeling local functionalities, performance, QoS of systems, and interaction protocols with their environments, along with composition of systems and their composed functionalities, performance, QoS of systems, and interaction protocols of the composite systems. Modeling and composition have to be applied to the models at different levels of abstractions. The main challenge here is to achieve interoperability among the models of heterogeneous subsystems.

Since 2006 when CPS was proposed, there has been extensive research on model-driven approaches to CPS development (Derler et al., 2012; Gunes et al., 2014; Simko et al., 2014; Hehenberger et al., 2016; Saeedloei and Gupta, 2016; Liu J et al., 2019). However, there is little systematic analysis and discussion about these methods, techniques, and tools in particular. Mohamed et al. (2020) surveyed the use of model-driven approaches in developing CPS applications, but they provided little analysis about techniques and tools, and challenges that they addressed.

In this paper, we present a survey on the state-of-the-art model-driven approaches from a system engineering perspective. We will introduce the most

representative model-driven techniques and tools, and discuss their features and solutions to the challenges they support. We argue that model-driven development is a key method to successful development of CPSs, and propose that more research efforts should be put into the development of techniques and tools. Our contributions include (1) a systematic review of model-driven approaches via a general overview of this domain for researchers of this realm, (2) an analysis of model-driven approaches used in practical development of CPSs, which can be regarded as a guidance for CPS builders, and (3) a proposal for promising research on building unified modeling platforms for CPS engineering.

The remainder of this paper is organized as follows: In Section 2, we present an overview of CPS where we summarize the historical development and key challenges related to CPS construction. In Section 3, we describe the general concepts of model-driven approaches for the discussion in the following sections. In Section 4, we present a comprehensive introduction to the representative techniques and tools in model-driven development, and provide analytic evaluation of whether and how they support CPS construction. Section 5 is a holistic discussion of model-driven approaches and provides suggestions for building practical CPSs. We conclude in Section 6 and propose future research directions.

2 Background and challenges of cyber-physical systems

For the technical discussion in the following sections, we briefly recall the historical development of CPS and discuss the key challenges.

2.1 Evolution of cyber-physical systems

We are actually now in the era of CPS 2.0, which is characterized as the intersection of technologies of computation, communication, control, big data, cloud computing, and artificial intelligence. It is also called CPS 1.5 as CPS 2.0 is also widely referred to as human-cyber-physical systems (HCPS). The preliminary CPS, or the so-called CPS 1.0, is characterized as networked cyber-systems and physical systems (Liu ZM et al., 2019), in which

1. the cyber world is formed by systems for computing, control, and communication networks;

2. the physical world includes mechanical, electrical, and chemical processes, etc.;

3. the cyber systems control the physical side using sensors and actuators;

4. a database server is needed to collect and process events generated by the sensors in the system, for computation of control decisions;

5. the network connects the cyber systems to the sensors and actuators, as well as the cyber systems for their communication. The data about the states of the physical processes, collected by the sensors, are sent through CPS networks to the database server to be stored and processed, and are used by the CPS control units to compute control decisions. Through the network, the control decisions are sent to the corresponding actuators to perform the control actions of the command.

Along the increasing research and scales of their applications, the number and variety of sensors, actuators, and the monitoring software systems used in CPSs have been growing. Therefore, CPSs become IoT-based integration of computing, communication, and control systems; the volume, variety, and velocity of data generated in CPSs have all the features of big data. A traditional database server used in CPS 1.0 described above does not meet the requirements for storing, processing, and using these data; big data analytic technology and cloud computing platforms are the natural solution to this problem. These CPSs supported by big data and cloud computing technologies are called CPS 2.0. Therefore, the features of CPS 2.0 extend those of CPS 1.0 with those enabled by big data and cloud computation as described below (Liu ZM et al., 2019):

1. Big data can be collected and used to develop algorithms for monitoring the behaviors and interactions among the cyber or physical subsystems, to prevent and predict abnormalities so as to handle them when they occur;

2. Big data can be collected about the environments and behaviors of the system to develop self-reconfiguration middleware components for self-adaptation;

3. Big data can be used to develop intelligent control and services based on high-level knowledge generated from the data;

4. Service-based nature of cloud computing platforms allows the development of applications based on cross-domain CPSs, such as those based on smart homes, smart health, and intelligent transportation

(Broy et al., 2012);

5. Cloud computing platforms are naturally required for the control and management of many data services, software services, interface devices (e.g., sensors and actuators), and physical processes, as well as ICT hardware and infrastructures, as on-demand resources.

Thus, big data and could computing in CPS 2.0 are the key to satisfy requirements for services to cross-domain customers, system safety, security, fault-tolerance, stability, adaptivity, and other QoS constraints. However, they are the source of more challenges including safety, security, and reliability due to the increased scale and complexity, and the use of machine learning systems which are not verifiable, controllable, or composable.

2.2 Challenges of cyber-physical systems

The construction of a CPS is in general a hard engineering process, which would go through many big challenges (e.g., Lee EA (2008, 2010), Rajkumar et al. (2010), Broy et al. (2012), Jirkovský et al. (2017), and Liu Y et al. (2017)). For some of these challenges we do not yet have developed systematic solutions. In this subsection, we summarize these challenges for the analysis of these model-driven methods according to what challenges they can or cannot address. For this purpose, we refer a challenge to label C-n as the challenge index n.

1. C-I: challenges in modeling

Models always play a central role in a CPS related engineering process (Derler et al., 2012; Lee EA, 2018). The complex heterogeneity is the major source of the following challenges in CPS modeling (Derler et al., 2012):

(1) interoperable integration (or composition) of the large variety of software systems developed by different people (or organizations), using different technology paradigms (i.e., languages, modeling theories, and tools);

(2) distribution, mobility, and spatial-temporal properties;

(3) interactions, concurrency, and synchronization among a large number of cyber systems and physical systems.

2. C-II: challenges in verification

Applications of CPSs are in general mission-critical and have safety requirements for realtime, concurrency, security, fault-tolerance, and robustness. The sources of challenges in verification of these requirements are due to

(1) large scale of the systems with complex behaviors which are the discrete and continuous state changes;

(2) a large variety of security threats in the operation environment and complex security vulnerabilities in the different cyber systems, their communications protocols, and the network;

(3) complex uncertainties in the environment and in the execution of the system because of the use of machine learning systems.

We need to develop novel theories, methods, and tools for effective verification and validation of safety, security, fault-tolerance, realtime, and robustness of CPSs.

3. C-III: challenges in system evolution

First, a complex CPS cannot be built from scratch. It in general starts from a simple working system and then evolves. Second, the application domain and the requirements are dynamic and changing. Therefore, a CPS is always open and new systems are continuously developed, and plugged to play; or existing systems which are running by themselves are connected to the CPS. Dynamically integrating various kinds of systems into the CPS is difficult.

4. C-IV: challenges in meeting diverse QoS requirements

CPSs of different applications have different requirements for system QoS, including extensibility, reliability, maintainability, availability, and other significant requirements related to a "good" system (sometimes called the "ilities" (Chen LP et al., 2013)). Moreover, reconfigurability, adaptivity, and evolvability are requested to cope with the dynamic uncertainty of the internal and external environments and requirements in CPSs. In addition, interoperability is particularly needed, because many complex CPS applications usually involve multi-domain scenarios (e.g., the smart health CPS described in Broy et al. (2012) across various domains including autopilot system, traffic management, and health care system). Besides, predictability is demanded to guarantee the specified outcome of a CPS's behaviors meeting the system requirements (Gunes et al., 2014), because not all the components of the CPS are predictable, such as data-driven intelligent components. Generally speaking, the

above QoS attributes and constraints should be well captured and modeled for implementation and verification in diverse CPS applications.

3 General concepts in model-driven approaches

The theories, techniques, and supporting tools of model-driven approaches are gradually maturing. There are several model-driven approaches, such as model-driven engineering (MDE), model-driven architecture (MDA), model-based testing (MBT), model-driven development (MDD), and model-based systems engineering (MBSE). These approaches have been adapted to diverse domains and used to solve practical problems.

There are two main concepts in model-driven approaches, i.e., "models and modeling" and "model transformation."

3.1 Models and modeling

Models are abstractions of real subjects (e.g., systems) and are built by people to help analyze the subjects. Focusing on a specific viewpoint, a model reflects the characteristics of the systems. Researchers from different domains define models differently. We list five definitions in Table 1.

Definitions of models are given from two aspects: what a model is and what a model is used for. To be useful, a model should satisfy a set of characteristics as follows:

1. Abstract: focusing on relevant aspects while removing irrelevant ones;

2. Purposeful: built to address a specific set of concerns of stakeholders;

3. Understandable: expressing in a form that is understood by stakeholders;

4. Accurate: faithfully representing the modeled system;

5. Predictive: used to answer questions about the modeled system;

6. Cost-effective: being cheaper and faster to build than the actual system.

Based on different criteria, models can be divided into different categories:

1. Considering the building purposes, there are descriptive models and prescriptive models.

2. Considering the forms, there are mathematical models, graph models, text models, etc.

3. Considering the abstraction levels (MDA (Mellor, 2004)), models are categorized as meta-meta-model, meta-model, model, and system (subject).

4. Considering the precision, models are divided into three levels, namely, conceptual models, specification models, and implementation models (Fowler, 2003). In MDA, a similar classification is computation-independent model (CIM), platform-independent model (PIM), and platform-specific model (PSM).

To build a qualified model, we must carefully choose the modeling language and follow an approved modeling process. Here, modeling means the activities of building models. For different purposes, a substantial number of modeling languages (e.g., UML, AADL, and BPMN) have been proposed. Modeling languages can be simply divided into two groups: general-purpose modeling languages and domain-specific modeling languages (DSMLs).

Compared with general-purpose modeling languages which contain a wide range of modeling abilities, domain-specific modeling languages aim at serving for specific domains (or problems). As a research trend, more and more DSMLs will be proposed and employed to build models. There are

Table 1 Model definitions

No.	Definition
1	"Engineering models aim to reduce risk by helping us better understand both a complex problem and its potential solutions before undertaking the expense and effort of a full implementation." (Selic, 2003)
2	"A model is a set of statements about some system under study (SUS)." (Seidewitz, 2003)
3	"Models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones." (Brown, 2004)
4	"A model is an abstraction of a (real or language-based) system allowing predictions or inferences to be made." (Kühne, 2006)
5	"A model of a system is a description or specification of that system and its environment for some certain purpose." (The Object Management Group, 2006)

two ways to define a DSML: (1) by tailing and adapting general-purpose modeling languages and (2) by defining DSML from scratch.

A modeling language contains three components: abstract syntax (language concepts and rules for combining concepts), concrete syntax (notations and representations of the concepts), and semantics (the meaning of concepts). Fig. 2 shows the components of modeling languages and the relations among them.

Modeling processes also play a key role in building models. By employing one modeling language, different people may build models with large quality differences. Quality differences lie in the experience of building models, and the main part of experience is reflected in the modeling process (efficient process guidance). Considering the diversity of modeling purposes and modeling languages, the modeling processes are also different.

Details on the topic of modeling can be found in Cabot and Gogolla (2012).

3.2 Model transformation

Model transformation plays a key role in achieving automation of model-driven approaches. Model transformation takes in source models and generates target models. Four definitions of model transformation are listed in Table 2.

Based on the source and target models, model transformation methods can be divided into different groups:

1. Considering the forms of source and target models, model transformation methods can be categorized into "text-to-model," "model-to-model," and "model-to-text;"

2. Considering the abstraction level of source and target models, model transformation methods can be categorized into "horizontal transformation" and "vertical transformation."

There are also other classification methods, such as meta-model based model transformation methods, model merging, and model evolution. Two systematic classifications of model transformation methods can be found in Czarnecki and Helsen (2003) and Mens and van Gorp (2006).

Model transformation methods are built on certain model transformation techniques, such as XSLT (W3C, 1999), VIATRA (Varró and Balogh, 2007), QVT (The Object Management Group, 2008), ATL (Jouault et al., 2008), Kermeta (Jézéquel et al., 2009), MOFM2T (Oldevik et al., 2005), JTL (Cicchetti et al., 2010), Tefkat (Lawley and Steel, 2005), and MOMENT (Boronat et al., 2006). Some of the techniques are general-purpose (e.g., XSLT and ATL), whereas the others are domain-specific (e.g., VIATRA). Table 3 illustrates briefly these nine model transformation techniques.

Most of the current model transformation techniques and tools are meta-model theory based ones and can work only with a special kind of model. Certain effort is required to use the general-purpose model transformation techniques properly. The scalability and adaptability of the domain-specific model transformation techniques are limited.

Another research aspect of model transformation is model transformation verification, which is

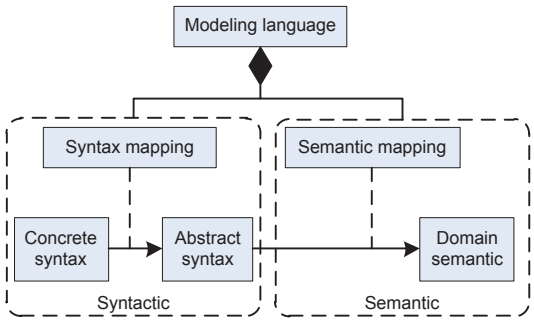

Fig. 2 Illustration of the modeling language components

Table 2 Model transformation definitions

Reference	Definition
Miller and Mukerji (2003)	"The process of converting a model into another model of the same system."
Kleppe et al. (2003)	"Automatic generation of a target model from a source model, according to a transformation description."
Tratt (2005)	"Model transformation is a program that mutates one model into another."
Wang TX et al. (2017)	"Model transformation is a process of generating target models based on source models. The transformation rules shall be built between same or similar concepts that are from the two models, respectively."

crucial in improving the quality and reliability of the generated models. One of the key purposes of model transformation verification is to validate the consistency between source model(s) and target model(s). Three surveys on model transformation verification are Amrani et al. (2012), Calegari and Szasz (2013), and Rahim and Whittle (2015). Focusing on two dimensions, "the properties to be verified" and "the approaches and techniques used to do verification," Table 4 shows the former dimension summary results and Table 5 shows the latter dimension summary results.

3.3 Summary of CPS construction and model-driven approaches

Traditionally, an industrial CPS is usually designed and constructed in a separation-integration manner. That means, initially, independent design is conducted respectively on the constitutions of a CPS, such as control system design, hardware system design, and software system design. A lot of simulations on each design are then performed separately. Afterwards, all the involved systems are implemented and integrated at the final phase.

Table 3 Model transformation languages and techniques

Technique	Characteristic	Relevant standards and techniques
XSLT	Associating patterns with templates	XML, XMI (The Object Management Group, 2000)
VIATRA2	Employing graph transformation techniques	VPM meta-modeling approach (Varró and Pataricza, 2003)
QVT	Conforming to MOF standards	MOF 2.0
ATL	Conforming to KM3 standards (Jouault and Bézivin, 2006)	Meta-model theory based
Kermeta	Rely on E-MOF	LOGO programming language
MOFM2T	Model-to-text transformation	MOF script language
JTL	Bidirectional transformations	EMF-based tool
Tefkat	Special for MOF models	Employing patterns and rules
MOMENT	Special for MOF models	Model merging

Table 4 Properties in the model transformation process

Category		Remarks
Language-related properties (computational nature)	Termination property	Target model existence
	Determinism	Unique target model
	Preservation of execution semantics	Transformation rules consistence
	Typing	Focusing on the transformation language
Transformation-related properties (modeling nature)	Conformance and model typing	Generated model conforms to the target meta-model; automatically checked
	N-ary transformation properties	A set of source models (model composition)
	Model syntax relations	Structural correspondence
	Model semantic relations	Dynamic consistency
	Functional behavior	Mathematical functions (e.g., injective, surjective)

Table 5 Techniques and approaches for model transformation verification

Category	Main techniques employed	Remarks
Testing-based	Meta-model coverage: equivalence partitioning; rule-based constraints: OCL, ECL, etc.	Challenging in defining test cases and oracles; partial correctness
Theorem proving	Direct verification: theorem provers (e.g., Coq), OCL, etc.; indirect verification: theorem provers, proof checker, etc.	Used particularly in key subsystem or component verification of a CPS, not usually for the whole CPS
Graph theory	Manually constructing mathematical proofs using graph theories	Both the transformation process and the verification of transformation results can be supported by graph theories
Model checking	Model checker is used; techniques for solving state space explosion, such as partial order reduction	Cover only certification of model checking

Obviously, this manner of CPS construction could get into troubles especially at the integration stage, and surely integration of various systems and components is a very costly work. This is because these systems are usually designed and analyzed using a variety of modeling formalisms and tools, and each representation of models commonly highlights certain features and disregards others. As suggested by the CPS realm pioneers, constructing CPS demands a new scientific and engineering foundation, especially the theories and methods to build CPS models (Lee EA, 2008; Rajkumar et al., 2010).

In consideration of the features of model-driven approaches, we naturally expect that model-driven approaches could benefit CPS construction as follows:

1. Model-driven approaches are always supported with mature technologies and tools, which can be employed to model various cyber or physical components, systems, processes, etc. Meanwhile, these generated models are used to do requirement analysis, system verification, program synthesis, and simulation.

2. Model-driven approaches provide potential solutions to some of the CPS challenges to some extent, such as modeling heterogeneous systems, time and concurrency, system integration, and diversity requirements.

In the next section, we analyze the techniques and tools used to verify whether the above expectations are met.

4 Techniques and tools

In this section, we introduce 10 techniques/tools in MBSE and analyze their applications in CPS. Though some of them do not directly support CPS modeling, they provide the foundation for CPS modeling so that extension tools can be developed based on them to provide support.

Some of these techniques/tools are modeling languages or standards that are equipped with several tools to automate them, whereas others are tools themselves. In fact, modeling languages, standards, and their companion tools are strongly connected to each other and they are always studied together as a whole object. Therefore, for simplicity, in the following we do not distinguish modeling languages, standards, or tools, and uniformly call them "tools" instead. One should notice that a tool here denotes a concept that integrates a modeling language or a standard, the methodology or theory behind it, and its relevant tools.

We first introduce 10 typical model-based tools for CPS construction in Section 4.1, and then make a comparative analysis of these tools in Section 4.2. In Section 4.3, we specifically analyze how the CPS construction challenges are addressed with the tools.

4.1 Introduction of model-based tools for CPS

In this subsection, we introduce 10 typical model-based tools which have been used mainly in the industry nowadays. They provide functionalities and features that support CPS modeling or act as foundations upon which some extensions are developed especially for CPSs. For each tool, we will give a brief introduction to its history, characteristics, and the role it plays in CPS MBSE. For most of the tools we introduce below, some of their extension tools will also be introduced. They extend the tools with some additional features and functionalities to provide particular support for different CPS aspects, such as modeling, simulation, analysis, and verification.

4.1.1 AADL

Architecture Analysis and Design Language (AADL) is an architecture description language defined as the Society of Automotive Engineers (SAE) standard (Feiler et al., 2006). It is developed for modeling and analyzing the architecture of real-time embedded systems. The core language of AADL consists of abstractions of components that can represent different components, interfaces, and concepts in system architectures and runtime semantics for different mechanisms for communication and control of systems, which can provide good support for dealing with CPS heterogeneity. AADL can be used to model and analyze both systems already built and systems under design and integration. It can be used in the analysis of architecture patterns that are partially defined and in full-scale analysis of a complete system model extracted from the source code. AADL can support predicting and analyzing critical system qualities, such as performance, schedulability, and reliability at early stages of

system development. AADL is supported by a wide range of tools, among which the most popular is OSATE (https://osate.org/).

However, AADL cannot specify the dynamic physical behavior in CPS. Banerjee et al. (2012) proposed BAND-AiDe to support the modeling of physical processes (like differential equations) in CPSs, which extends AADL by developing a new annex called BAN-CPS. Nevertheless, the BAN-CPS annex is limited to supporting only physical properties that characterize physical behaviors, environmental factors, and physical entities (such as the blood glucose level or blood pressure) and does not support time-related properties.

Liu J et al. (2019) proposed an annex of AADL, called AADL+, to facilitate the modeling of not only the discrete and continuous behavior of CPS, but also the interaction between cyber components and physical components. They also developed a plug-in for OSATE for CPS modeling. The plug-in supports syntax checking and simulation of the system model through linking with Modelica.

AADL itself does not provide formal support for CPS specification and verification. Larson et al. (2013) proposed an extension of AADL called behavioral language for embedded systems with software (BLESS) to fill in the gap. As an annex of AADL, BLESS provides formal semantics for AADL behavioral descriptions. Additionally, BLESS can formally specify the contracts on AADL components by capturing both functional and timing properties. The corresponding tools were also developed for automatically generating and proving verification conditions from these contracts.

4.1.2 UML/SysML/MARTE

Unified Modeling Language (UML) is a standard, general-purpose software modeling language, developed by Booch et al. (1999) and has been adopted as a standard and managed by the Object Management Group (OMG) since 1997. It provides visual components to help specify, visualize, construct, and document the artifacts of a software system. UML captures the static structure and dynamic behavior of a system at a high level in an object-oriented way using a collection of objects and provides multiple views to model different aspects of a system. UML can also help arrange models into packages, control dependencies among the packages, and manage the versioning of model units with the organizational constructs it provides. However, UML is a discrete modeling language and does not support modeling continuous systems like those found in engineering and physics. There are many tools for UML, and a list of common tools can be found online (https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools).

Many variations and/or extensions of UML have been proposed to meet particular CPS requirements. Here we give two examples.

SysML is an extension of a subset of UML to support MBSE of complex and large-scale systems such as CPS (Friedenthal et al., 2015). Compared with UML, SysML better addresses system requirements and parameters. It is lightweight in expressing system engineering semantics and its model management constructs can support the specification of models, views, and viewpoints. SysML can help specify and build CPSs and specify components that can then be designed using different domain-specific languages. SysML provides system engineers with a coherent, standardized way of representing system components and their interactions in a way that is easily transformed, manipulated, and maintained (Huang et al., 2018). There are many tools for UML that provide support for SysML, e.g., MagicDraw & SysML (https://www.nomagic.com/product-addons/magicdraw-addons/sysml-plugin) and Papyrus & SysML (https://www.eclipse.org/papyrus/components/sysml/).

MARTE (Selić and Gérard, 2014) is a domain-specific modeling language for real-time and embedded systems. It is defined via a UML2 profile, and extends the UML with concepts related to the domain of real-time and embedded systems. MARTE supersedes the UML by providing a precise, comprehensive, and flexible model of time, the ability to specify quantitative and qualitative measures, the ability to accurately model hardware resources and software resources specific to real-time and embedded software, and the ability to accurately capture the relationships between software applications and the computing platforms that support them. These features make MARTE especially suitable for CPS modeling (Mallet, 2015). SysML and MARTE can be combined to model a CPS in a multi-view way (Huang et al., 2018). There are many tools for UML that provide support for MARTE,

e.g., Papyrus & MARTE (https://www.eclipse.org/papyrus/components/marte/).

4.1.3 Ptolemy

Ptolemy (https://ptolemy.berkeley.edu/) (Ptolemaeus, 2014) is an open-source software framework for modeling and simulating complex systems, especially hierarchical and heterogeneous systems like the CPS. Ptolemy is based on actor models, which are components that execute concurrently and share data with each other by sending messages via ports. The actor-based model is natural for expressing concurrency between system modules in a dataflow style. Different domains can be associated with actors to support different models of computation (MoCs), such as the dataflow model, discrete-event model, and continuous-time model. Ptolemy is convenient for supporting a modeling hierarchy by defining a composite actor, which is itself a composite of other actors.

Lee EA et al. (2015) proposed an open-source simulator for cyber-physical systems called CyPhySim, which is based on Ptolemy II. It supports different simulation techniques for different MoCs, including ordinary differential equations, hybrid systems, discrete-event models, and the functional mockup interface (FMI). CyPhySim features an innovation called "smooth tokens" that can dramatically reduce the computation for simulation.

4.1.4 SCADE

SCADE (https://www.ansys.com/products/embedded-software/ansys-scade-suite) is a model-based development environment for critical embedded software developed by the ESTEREL company in France. With native integration of the formally defined SCADE language, SCADE is the integrated design environment for critical applications including requirements management, model-based design, simulation, verification, qualifiable/certified code generation, and interoperability with other development tools and platforms. SCADE is used mainly for designing critical embedded software, such as flight control and engine control systems, landing gear systems, automatic pilots, and power and fuel management systems. SCADE supports a complete software design process, from architecture to detailed design of components, with a set of supporting tools. It provides dataflow-based and state-machine-based modeling techniques and powerful debugging and simulation techniques in a highly integrated environment. SCADE also supports different types of model analysis, like static analysis, rule checker, timing, and stack size optimization. It also supports code generation: it includes a tool called KCG, a code generator for C and Ada languages from SCADE models.

4.1.5 Simulink/Stateflow

Simulink (https://www.mathworks.com/products/simulink.html) is a graphical programming environment developed by MathWorks for modeling, simulating, and analyzing multi-domain dynamical systems. It is integrated with the MATLAB tool and can either drive MATLAB or be scripted from it. Simulink offers different types (including self-defined type) of basic block diagrams, from which users can design the model of their systems at the system level (in graphical or text mode). Its language supports describing both discrete- and continuous-time behaviors of the system. Simulink supports simulation, validation, and verification of system models, and code generation for programming languages such as Java, C/C++, and Ada from the script of Simulink. Stateflow extends Simulink with a design environment for developing state machines and flow charts. It improves the capability of Simulink for modeling and simulating complex embedded systems. Because of the features above, Simulink/Stateflow can provide good support for modeling and simulation of CPS. It provides convenience for concurrency modeling of both the cyber and physical parts of a system with a strong ability to stress timing and synchronization issues.

PVS-Simulink is an integrated environment for CPS model-based analysis (Bernardeschi et al., 2018). It uses PVS, a popular theorem prover, for the specification of discrete components, and uses Simulink to model continuous processes. PVS-Simulink then integrates the simulation of the PVS specifications and simulation of Simulink models to form a co-simulation environment for CPS. With PVS-Simulink, one can conduct a co-simulation of a CPS with its discrete components with the ability to verify components by PVS.

4.1.6 Modelica

Modelica (https://www.modelica.org/) is an open-source, object-oriented, declarative language for modeling complex and heterogeneous systems. It is suited for multi-domain modeling (Fritzson, 2014). It supports modeling both continuous and discrete behavior with mathematical equations (i.e., differential, algebraic, and discrete equations). These equations can be largely solved in an efficient and automatic way. Modelica is especially suited to embedded control systems modeling and simulations. Recently, new developments have occurred in Modelica to facilitate integrated CPS modeling and timing simulation (Henriksson and Elmqvist, 2011).

The Modelica language is optimally suited for the modeling task, but Modelica had not been focused on the modeling and simulation of moderately sized models until 2015. The platform OpenModelica has been proposed to deal with large-scale CPSs (Braun et al., 2017). It implements sparse solver techniques to efficiently compile and simulate large-scale Modelica models.

4.1.7 BPMN

Business Process Model and Notation (BPMN) (http://www.bpmn.org/) (The Object Management Group, 2013) is an OMG standard for specifying business processes in a business model. It provides a graphical notation called the "business process diagram" for specifying business processes. BPMN supports business process management for both technical users and business users, by providing a notation that is intuitive for business users, but can represent complex process semantics. BPMN also defines the semantics of the graphics notation under formal languages like the Business Process Execution Language (BPEL) (Juric et al., 2004). With notations that are understandable for all business users, from business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the businesspeople who will manage and monitor those processes, BPMN creates a standardized bridge between business process design and process implementation.

BPMN lacks structure, expressiveness, and flexibility for expressing the requirements brought by the integration of the CPS into the business process. Bocciarelli et al. (2017) proposed an extension of BPMN for specification and management of the business processes supporting cyber-physical production systems (CPPSs). It is composed of PyBPMN, a BPMN extension that annotates performability-oriented properties in BPMN models, and eBPMN, a platform for business process simulation. The BPMN for CPS (BPMN4CPS) provides efficient support for modeling complex resources in CPPSs, and also supports the simulation-based analysis of CPPSs in terms of reconfigurability, adaptability, and reliability.

4.1.8 KeY/KeYmaera X

KeY (https://www.key-project.org/) (Leino, 2007) is a specification and verification tool for Java programs developed by the Karlsruhe Institute of Technology. It accepts specifications written in the Java Modeling Language or Java, and transforms them into a dynamic logic formula, which can then be analyzed and proved through a set of sound deductive rules of the logic. KeY is powerful in that it supports both interactive and fully automated correctness proofs.

Based on KeY, Platzer (2018) proposed a deductive verification tool for hybrid systems, called "KeYmaera X" (http://www.ls.cs.cmu.edu/KeYmaeraX/). KeYmaera X is based on a variation of dynamic logics for specification and verification of hybrid systems, called "differential dynamic logic" (d$\mathcal{L}$). KeYmaera X extends KeY with computer algebra systems like Mathematica and corresponding algorithms and proof strategies so that d$\mathcal{L}$ formulae can be proved in practical verification. d$\mathcal{L}$ captures the basic mathematical model for the CPS. It can specify functional and temporal properties of hybrid systems, which proved to be crucial for CPSs in practice (Platzer, 2018). Based on d$\mathcal{L}$, Müller et al. (2016) and Lunel et al. (2019) proposed a component-based approach for CPS verification by introducing extra structures in d$\mathcal{L}$ to express different CPS components, such as a physical plant and controller. Their theory has been fully implemented in KeYmaera X.

4.1.9 Event-B/Hybrid Event-B

Event-B (http://www.event-b.org/) is a formal method for system-level modeling and analysis that is influenced by the B Method (Abrial, 2005) and Action Systems. It features use of set theory as a modeling notation, refinement as the means to represent systems at different abstraction levels, and mathematical proof as the approach for verifying consistency between refinement levels. A system behavior in Event-B is modeled by a collection of state variables and a collection of guarded actions on the state variables. Accordingly, it allows for modeling of highly concurrent systems (Abrial et al., 2010). Also, Event-B enables a stepwise refinement to model and verify complex systems. Thus, it is naturally used to develop many complex safety-critical CPS applications (e.g., railway systems). Generally, Event-B is used to describe a discrete event system, and proof obligations therein permit verification of properties of the event system. Accordingly, it requires extensions to better model and verify a CPS, which typically integrates both discrete and continuous processes. Hybrid Event-B was proposed in recent years by introducing a single Hybrid Event-B machine or multiple Hybrid Event-B machines to particularly address problems of representing continuous CPS behaviors (Banach et al., 2015, 2017). This extended version of Event-B allows a stepwise refinement within the CPS construction process, even as the continuous physical processes have not yet been digitalized.

The Rodin Platform is an open-source Eclipse-based integrated development environment (IDE) for Event-B that provides effective support for modeling and tool-assisted automated proof. It has a graphical front end for Event-B specification, and features design-time feedback, which is almost immediate feedback as the model is being constructed. Generation of the proof obligations is built into Rodin, which frees the user from the tedious work of writing them explicitly. Different from the general-purpose theorem provers that provide a strong logical foundation, Rodin chooses a trade-off between logical soundness and practicality from an industrial point of view (Abrial et al., 2010). Development of Rodin was supported by the European Union ICT Project ADVANCE (2011 to 2014) and is now the widely accepted tool for Event-B-based modeling and verification.

4.1.10 Hybrid system related tools

A CPS is generally developed as a hybrid system. Accordingly, many hybrid system related tools can be extended to support CPS modeling and verification. Because the concept of a hybrid system was first proposed in the 1990s (Maler et al., 1992), hybrid automata were typically used in building hybrid system models (Henzinger, 1996). A rich body of research on hybrid automata has been conducted in the past three decades. For example, hybrid input/output (I/O) automata were proposed to support interaction and composition of multiple hybrid systems (Lynch et al., 2003). Operational semantics on a hybrid system was defined for simulation to enable large-scale applications (Lee EA and Zheng, 2005). Machine-learning-enhanced model verification on hybrid systems was also studied.

The hybrid communicating sequential process (HCSP) has commonly been used in hybrid system modeling and verification in recent years. Generally, by using MATLAB/Simulink, a hybrid system is modeled as a graphic representation. By defining semantics for Simulink, the graphic models can be automatically transformed into HCSP, thereby supporting both formal verification and simulation. Regarding HCSP, significant contributions have been made by Zhan et al. (2016), who defined the axiomatic semantics for HCSP, namely, the hybrid Hoare logic (HHL) (Liu J et al., 2010). Also, they developed an improved HHL theorem prover for interactive proof by shallowly embedding an HHL proof system into Isabelle/HOL (higher-order logic), which is capable of releasing proof burden (Wang SL et al., 2015). In addition, He and Li (2017) proposed a hybrid system specification language, which is similar to the Dijkstra Guarded Command Language. The specification language combines the program, physical process, and communication abstractions, thereby building the hybrid relational calculus based on unifying theories of programming (UTP).

In summary, by extending the tools on hybrid systems, CPS construction challenges, especially the unification of a continuous physical dynamic model and discrete computing state transition, can be well addressed. However, as emerging CPSs become increasingly large-scale, heterogeneous, autonomous, and intelligent, systematic and specific

tools for CPS construction and verification are still demanded to address new challenges.

4.2 Comparative analysis of tools for CPS construction

In this subsection, we give a comparative analysis of the tools introduced in Section 4.1. We make the comparison based mainly on two dimensions: modeling languages and functionalities that are significant for CPS construction.

4.2.1 Modeling languages

As shown in Table 6, we focus on four aspects in comparison of the tool modeling languages in Section 4.1: (1) "modeling language" indicates the modeling languages used by the respective tool; (2) "executable semantics" indicates if the semantics of the languages is executable; (3) "graphical notation" indicates whether the tool supports graphical notations; (4) "target system" indicates the types of systems that the tool is suitable for modeling. We also list extensions of the tools introduced above. The extensions contain the listed features in the modeling languages.

Most tools provide self-defined modeling languages. For example, AADL provides a language consisting of a set of different types of component abstractions and properties (Feiler et al., 2006), Event-B provides the Event-B language which is based on the typed set theory, and HCSP itself is a modeling language.

Ptolemy supports multiple computation models in its semantics, such as a dataflow model, a process network, a discrete event model, and a continuous-time model. The underlying language of SCADE is the synchronous language LUSTRE (Halbwachs et al., 1991). KeY was developed for the verification of a variation of dynamic logic—a logic for specifying and reasoning about program behaviors.

The semantics of some modeling language tools (e.g., Ptolemy and SCADE) is executable, so they are mainly for low-level and specific-domain modeling or specification, while other tools are mainly for high-level modeling or specification.

Most of the tools support graphical notations, except for KeY, whose model is formal program written in text forms.

The above tools support modeling a wide range of systems according to their different features and developing visions. For example, AADL is most

Table 6 Comparison of modeling languages

Tool	Extension(s)	Modeling language	Executable semantics	Graphical notation	Target system
AADL	BAND-AiDe, AADL+	–	No	Yes	Real-time embedded systems & architecture of systems/BAND-AiDe: body area networks and devices; AADL+: cyber-physical systems
UML	SysML, MARTE	–	No	Yes	General systems & architecture of systems/ SysML: parametric systems; MARTE: real-time embedded systems
Ptolemy	CyPhySim	Dataflow & process network & discrete event model & ...	Yes	Yes	Hierarchical and heterogeneous systems & real-time systems & real-time embedded systems
SCADE		LUSTRE	Yes	Yes	Real-time embedded systems
Simulink/ Stateflow	PVS-Simulink	–	Yes	Yes	Hybrid & real-time systems & hierarchical systems
Modelica	OpenModelica	–	Yes	Yes	Complex and heterogeneous systems & hybrid & real-time systems
BPMN	BPMN4CPS	–	No	Yes	Business process of general systems/ BPMN4CPS: business process of CPS
KeY	KeYmaera X	Dynamic logic	Yes	No	Java programs/KeYmaera X: hybrid systems
Event-B	Hybrid Event-B	–	Yes	Yes	Highly concurrent systems & discrete event systems/hybrid systems & CPS
HCSP/HHL		–	Yes	Yes	Hierarchical and real-time systems & hybrid systems & CPS

– indicates that these tools provide self-defined modeling languages. The term after "/" shows the more specific types of systems that the modeling languages of the tool extensions are more suitable for modeling

suitable for modeling real-time embedded systems, especially the architecture of systems at a high level. As indicated in Section 4.1, the modeling languages of some tool extensions are more suitable for modeling more specific types of systems. For example, BAND-AiDe is especially suitable for modeling body area networks and devices (Banerjee et al., 2012), while AADL+ is especially suitable for CPS modeling. Classical Event-B is suitable for modeling discrete event systems, while the extended Hybrid Event-B is especially suitable for hybrid systems and CPS modeling.

4.2.2 Functionalities

We consider six basic functionalities in MBSE that are important for CPS modeling. As shown in Table 7, the functionalities include: (1) model transformation, indicating support for model transformation; (2) physical modeling, indicating support for modeling physical processes; (3) verification, indicating support for verification; (4) program synthesis, indicating support for synthesizing programs; (5) simulation, indicating support for simulation; (6) system scale, indicating support for system modeling scale. We list both tools and their extensions because they may differ in these functionalities.

As can be seen, only some of the tools support model transformation, e.g., Ptolemy, Simulink, BPMN, and HCSP. Specifically, HCSP supports bidirectional model transformation with Simulink (Zhan et al., 2016). More commonly, the introduced tools in the survey act as the model transformation targets. For example, Event-B was the transformation target from UML (Younes et al., 2014).

Modeling physical components and their behaviors is indispensable to MBSE of CPSs. The tools UML, BPMN, and KeY do not provide such functionality. Although some tools, like AADL and SCADE, provide support for modeling real-time systems, they do not support modeling physical processes. Event-B does not support modeling of continuous behaviors, but many non-software parts, such as mechanical components, can be modeled with it. Tools like Ptolemy, Simulink/Stateflow, Modelica, Hybrid Event-B, and HCSP support modeling both real-time and continuous physical processes.

Regarding formal verification, KeY and its extension, KeYmaera X, support theorem proving for dynamic logics. Similarly, Event-B/Hybrid Event-B and HCSP also support a theorem proving style verification. By contrast, SCADE and Simulink/Stateflow support model checking for their

Table 7 Comparison of functionalities

Tool/Extension	Model transformation	Physical modeling	Verification	Program synthesis	Simulation	System scale
AADL	No	No	No	C, Ada	No	Large-scale
AADL/BAND-AiDe	No	Yes	No	C, Ada	No	Large-scale
AADL/AADL+	No	Yes	No	C, Ada	Yes	Large-scale
AADL/BLESS	No	Yes	Yes	C, Ada	No	Large-scale
UML	No	No	No	*	No	Large-scale
UML/SysML	No	Yes	No	*	No	Large-scale
UML/MARTE	No	No	No	*	No	Large-scale
Ptolemy	Yes	Yes	No	C, Java, VHDL	Yes	Moderate
Ptolemy/CyPhySim	Yes	Yes	No	C, Java, VHDL	Yes	Moderate
SCADE	Yes	No	Model checking	C, Ada	Yes	Large-scale
Simulink/Stateflow	Yes	Yes	Model checking	C/C++	Yes	Moderate
Simulink/Stateflow/ PVS-Simulink	Yes	Yes	Model checking & theorem proving	C/C++	Yes	Moderate
Modelica	No	Yes	No	No	Yes	Moderate
Modelica/OpenModelica	No	Yes	No	No	Yes	Large-scale
BPMN	Yes	No	No	No	No	Large-scale
BPMN/BPMN4CPS	Yes	Yes	No	No	Yes	Large-scale
KeY	No	No	Theorem proving	No	No	Small
KeY/KeYmaera X	No	Yes	Theorem proving	No	No	Small
Event-B/Hybrid Event-B	No	Yes	Theorem proving	No	Yes	Large-scale
HCSP	Yes	Yes	Theorem proving	System C	Yes	Large-scale

* A wide range of languages

models. The PVS-Simulink, an extension tool of Simulink/Stateflow, supports both model checking and theorem proving.

For program synthesis, the most popular target languages generated by the tools are C and Ada. HCSP can also support generation of System C programs. The UML tool and its extensions can generate a wide range of languages supported by different code generators. Additionally, although some tools like Rodin for Event-B do not support code generation, they provide extension points for three-part plug-ins to extend the demanded functionalities.

Tools with executable semantics support simulation, e.g., HCSP and Event-B. For some tools that do not support simulation, their extension tools might support it because they are associated with other tools that support simulation. For example, AADL+ supports simulation of the system model by linking with Modelica, which supports simulation with its executable semantics (Platzer, 2018).

CPSs are usually large-scale systems. Some tools regard logical soundness as the only, or the most preferred, consideration. That may create great difficulty in scaling up the system. For example, KeY and its extension tool KeYmaera X are best suited for relatively small CPSs because it is impractical for theorem proving of large programs in CPSs. By contrast, some tools treat logical soundness as one of the design considerations, and come to a practical tradeoff of multiple design considerations. For example, Event-B is demonstrated to be fruitful in large-scale industrial projects.

4.3 Special analysis regarding CPS challenges

In this subsection, we conduct a special analysis of the aforementioned tools to consider whether and how they cope with new CPS challenges. We summarize the analysis results in Table 8. Specifically, we analyze the potential solutions to each challenge introduced in Section 2.2.

4.3.1 Regarding C-I: challenges in modeling

Complex heterogeneity is the major source of modeling challenges. Most of the aforementioned tools can provide support for dealing with CPS heterogeneity to some extent, including AADL, UML, Ptolemy, SCADE, Modelica, Event-B, HCSP, and their extensions.

Among them, AADL has strong capabilities for describing the architecture of a heterogeneous system because of its pragmatic and practice-inspired effectiveness in combining software and hardware component models (Feiler et al., 2006).

Table 8 Support for coping with CPS challenges

Tool	C-I	C-II		C-III	C-IV
	Modeling (heterogeneity)	Time	Concurrency	System evolution	Meeting diverse QoS
AADL	Yes	Yes	Yes	Partly supported	Well supported
AADL/BAND-AiDe	Yes	Yes	Yes	Partly supported	Well supported
AADL/AADL+	Yes	Yes	Yes	Partly supported	Well supported
UML	Yes	No	No	Partly supported	Partly supported
UML/SysML	Yes	No	Yes	Partly supported	Partly supported
UML/MARTE	Yes	Yes	Yes	Partly supported	Partly supported
Ptolemy	Yes	Yes	Yes	Well supported	Well supported
Ptolemy/CyPhySim	Yes	Yes	Yes	Well supported	Well supported
SCADE	Yes	Yes	Yes	Partly supported	Well supported
Simulink/Stateflow	Yes	Yes	Yes	Partly supported	Well supported
Simulink/Stateflow/ PVS-Simulink	Yes	Yes	Yes	Partly supported	Well supported
Modelica	Yes	Yes	Yes	Partly supported	Well supported
Modelica/OpenModelica	Yes	Yes	Yes	Partly supported	Well supported
BPMN	No	No	No	Not well addressed	Partly supported
BPMN/BPMN4CPS	No	Yes	No	Not well addressed	Partly supported
KeY	No	No	No	Not well addressed	Partly supported
KeY/KeYmaera X	No	Yes	No	Not well addressed	Partly supported
Event-B	Yes	No	Yes	Partly supported	Well supported
Event-B/Hybrid Event-B	Yes	Yes	Yes	Partly supported	Well supported
HCSP	Yes	Yes	Yes	Partly supported	Well supported

UML does not focus on a specific domain, but it provides a flexible extension mechanism to widen its application field (Mallet et al., 2017). Both SysML and MARTE are extensions of UML and provide certain support for CPS heterogeneity. SysML can represent system components and their interactions (Selić and Gérard, 2014; Huang et al., 2018), and MARTE can provide the ability to model hardware and software resources and capture their relationships.

In Ptolemy, a complex model is partitioned into a hierarchical tree of nested submodels, and each level of the hierarchy is constrained to be locally homogeneous by using a common MoC (Lasnier et al., 2013). Subsequently, the submodels at the same level are joined together to form homogeneous networks, which are hierarchically combined to form a larger heterogeneous model (Ptolemaeus, 2014). Through such hierarchical multi-modeling, Ptolemy supports heterogeneous CPS modeling with strong semantics.

Similarly, SCADE and Simulink/Stateflow adopt the hierarchical multi-modeling approach to model complex CPSs. Unlike the approach adopted in Ptolemy, SCADE hierarchically combines finite-state machines with a synchronous/reactive formalism, and Simulink/Stateflow combines continuous- and discrete-time models with finite-state machines (Ptolemaeus, 2014).

Modelica is specifically designed for modeling large, complex, and heterogeneous systems. Models in Modelica are described by schematics (a.k.a. object diagrams) which consist of connected components (Fritzson, 2014; Nazari et al., 2015). There are connectors (a.k.a. ports) that are based on differential-algebraic equations (DAEs) to describe the possible interactions between components. Consequently, large models with more than 100 000 equations can be efficiently dealt with when using Modelica.

Event-B, especially its extension Hybrid Event-B, was developed for modeling not only software components but also non-software parts of the system and its environment. Specifically, the continuously varying behaviors of physical processes are addressed by designing multiple machines. Mediated by INTERFACE and PROJECT constructs, these machines are capable of cooperating.

HCSP is typically used to unify discrete state changes and continuous evolution of physical differential equations. However, more work is needed for the issue of its composability, which supports networked construction of CPSs, as well as the issue of extendibility for scaling up systems.

There are several co-modeling approaches for better CPS modeling by combining different tools. Selić and Gérard (2014) described a combination of SysML and MARTE to model a complex CPS. Therein, SysML was used to represent the mechanical and physical components that interact directly with the software, and MARTE was applied to model the software components and the information related to real-time and embedded aspects. Moreover, SysML was adopted in conjunction with Modelica to construct a heterogeneous simulation environment for CPS design and simulation (Wang BB and Baras, 2013). Specifically, Modelica was used to model and simulate subsystems, and Modelica models are then transformed into SysML blocks such that all domain-specific details are hidden for system engineers.

With increasing CPS component heterogeneity, the aforementioned tools gradually become incompetent. More effort is required to study the definition, refinement, and composition of the contracts of unifying interfaces of CPS components.

4.3.2 Regarding C-II: challenges in verification

CPS applications are usually developed for safety-critical domains, which demand verification on the safety requirements for realtime, concurrency, security, fault-tolerance, and robustness.

Generally, mathematical-model-based formal verification is known to play a fundamentally important role in verifying system safety. A CPS is typically a hybrid system with continuous behaviors and discrete controls that interact with each other. Formal verification for hybrid systems has basically two branches: (1) theorem proving, an inductive method that regards the assertion of "the system meets its specification" as a logical proposition, and then proves it by deductive reasoning through a set of reasoning rules; (2) model checking, an algorithmic method which automatically detects "whether the system behavior meets the given properties" by exhaustively searching the state space of the system model to be verified. Because it is much easier to check "whether a structure satisfies a formula" than to prove that "a formula is satisfied for all structures," model checking is widely accepted in system

or software verification (Clarke, 2008). Model checking can be further divided into explicit and implicit approaches (Wang J et al., 2019). The former traverses the state space through state computation, whereas the latter traverses the state space through fixed point generation. However, both approaches are essentially an exhaustive search problem on finite state space (Wang J et al., 2019) and can be further reduced to a reachability analysis problem (Xue et al., 2017). The CPS is obviously a system of infinite states. The scale of its state space is far beyond the computational capability of current computing facility. We call it the state explosion problem, which makes it difficult to compute the reachable set and perform formal verification in such complex and large-scale CPSs. To tackle this problem, a fundamental thought is to build an approximation model, such that the approximated reachable set is computable and the abstracted CPS model is verifiable in scale. By computing the over-approximation of the reachable set, we can say that a CPS is safe if the over-approximation set is disjoint from the unsafe region. In turn, we can say that a CPS is unsafe if the under-approximation set intersects with the unsafe region. Moreover, through backward under-approximation computation of the reachable set, we can determine the initial safe set of the CPS that ensures its reliable operation (Xue et al., 2016).

Specifically, we take two representative requirements, realtime and concurrency, as the criteria, to evaluate the aforementioned tools for their support on analysis or verification of such requirements.

AADL is especially effective for modeling complex real-time embedded systems because of inclusion of the thread. The thread is a component that can be concurrently scheduled by run-time execution (Feiler et al., 2006). Hence, AADL provides support for CPS time and concurrency properties.

Regarding the time and concurrency properties, UML provides elementary support, which is not enough for CPS modeling (Selić and Gérard, 2014; Mallet et al., 2017). SysML has the same ability as UML to support these two properties because of its dependence on the basic UML time model. However, MARTE can be regarded as a real-time extension of UML because it directly supports several concepts that are specific to the real-time domain, such as semaphores, concurrent tasks, or schedulers and scheduling policies (Selić and Gérard, 2014).

Ptolemy aims at modeling and simulating real-time embedded systems (Eidson et al., 2012). In Ptolemy, domains are defined to provide the rules for governing concurrent execution of components and the communication between components (Ptolemaeus, 2014). To support the interoperability of domains, Ptolemy provides a mechanism that represents a coherent notion of time. With this mechanism, domains share a common representation of time and a multi-form model of time.

SCADE builds a model-based development for critical embedded systems with a dataflow synchronous language called LUSTRE. The synchronous interpretation of LUSTRE makes it well suited for handling time in programs (Halbwachs et al., 1991). Unlike other languages that have parallel and concurrent execution schemes, parallelism in LUSTRE is fine-grained, while its concurrent execution is rather inefficient. Therefore, SCADE provides limited support for modeling CPS concurrency (Halbwachs et al., 1991; ESTEREL, 2015).

Simulink provides support for describing both continuous- and discrete-time CPS behaviors using differential equations and discrete-time difference equations (Broman et al., 2012). Moreover, Simulink supports concurrency using model partitioning. Partitioning (https://www.mathworks.com/help/xpc/ug/concurrent-execution-on-simulink-real-time.html) allows engineers to create tasks independent of the details of the target system on which the application is deployed. Hence, partitioning in Simulink is used to designate regions of the system for concurrent execution.

Modelica is based on the synchronous dataflow principle and the single assignment rule (Fritzson, 2014). Therein, all variable values can be accessed at any time during continuous integration and at event instants. Simultaneously, the relations between variables in active equations have to be fulfilled concurrently at every time instant. Hence, Modelica supports modeling CPS time and concurrency properties.

There is little evidence that BPMN and BPMN4CPS support modeling of the time aspect of systems. However, the term "fork" is used in BPMN to refer to dividing a path into multiple parallel paths (The Object Management Group, 2013). It is a place in the process where activities can be performed concurrently. Therefore, BPMN and BPMN4CPS

support modeling the concurrency property of systems.

KeY does not provide support for modeling the CPS time property because of the underlying modeling language and dynamic logic (Platzer, 2008; Fulton et al., 2015). KeYmaera X, an extension of KeY, can support modeling of continuous time because of its differential dynamic logic (d$\mathcal{L}$). The KeY and KeYmaera X execution schemes are sequential; they do not have the CPS concurrency property (Platzer, 2010).

Event-B follows the idea of the Action System approach, which models system behavior by a collection of state variables and a collection of guarded actions that act on the state variables. This structure enables Event-B to model highly concurrent systems. The time property was not supported in Event-B, but modeled in Hybrid Event-B, because time is especially required in modeling continuously varying behaviors.

HCSP was well structured on the basis of process algebraic specification, which is especially suitable for modeling complex concurrent, parallel, and distributed systems. Moreover, the time property is modeled and verified in HCSP.

4.3.3 Regarding C-III: challenges in system evolution

A CPS evolves typically by dynamically integrating and collaborating various kinds of systems. System integration and coordination require systematic modeling languages and corresponding architecture methodology. However, in general, a systematic architecture modeling language and the corresponding modeling and analysis methodologies are still missing.

As the analysis in Section 4.3.1 showed, many tools support modeling system heterogeneity. That enables these tools to partially support integration and collaboration of heterogeneous systems. There are two main ideas for integrating heterogeneous systems. First, an approach of multi-paradigm modeling describes a way of combining diverse computing models. Typically, Ptolemy II is combined with both actor-oriented modeling and multi-paradigm modeling. Other tools that support heterogeneous systems integration include Simulink/Stateflow, SCADE, Modelica, AADL, UML, Event-B, and HCSP. The second idea is tool integration, which combines different modeling tools through transformation languages or by using collaborative simulation. This approach is a great challenge because of its weak chain of tools. Fortunately, many efforts have been made on this approach, and prototype tools have already been proposed (we introduce them in detail in the discussion section).

The current CPS is usually built on a cloud platform. The cloud is strongly featured by service-oriented architecture (SOA). SOA-based architecture modeling was reported by Tariq et al. (2015).

In consideration of the ever-evolving and increasingly independent, autonomous, and intelligent CPS component systems, Chen X and Liu (2017) proposed a component-based CPS architecture model based on interface-driven rCOS. Based on the architecture model, they designed and implemented a networked system of energy meters by jointly using coordination language Reo and Eclipse extensible coordination tools (ECTs). However, the architecture model has provided only the formal schema without semantics defined for cyber-physical components. In addition, Tan et al. (2009) proposed a spatio-temporal event model for CPSs. They provided a mathematical model for events, and described different types and spatio-temporal properties of events, along with proposing operations on event composition.

Addressing the current challenge, we still need real progress on a CPS architecture model, systematic modeling language, and the corresponding methodology. Moreover, a CPS is naturally cloud-native and simultaneously combined with IoT, big data, and ever-evolving cyber or physical components. Accordingly, the core issues related to the CPS architecture model are to successfully abstract and virtualize by layers and to rapidly adapt to continuous changes. Recently, the new software-defined techniques show potential advantages in building and contraction of CPSs (Qin et al., 2014).

4.3.4 Regarding C-IV: challenges in meeting diverse QoS requirements

Various QoS properties need to be considered when constructing CPS models. The tools mentioned in Section 4.1 can partly address different functional and non-functional CPS requirements, such as the following.

AADL provides support for various kinds of

requirements concerning critical system qualities along with conventional modeling, including performance, real-time schedulability, safety, and security (Feiler et al., 2006; Brau et al., 2017). Zhang and Feng (2014) presented an aspect-oriented QoS modeling method based on AADL to include new features for separation of concerns (e.g., timeliness and fault-tolerance).

Functional and performance requirements are used when modeling a CPS using SysML to specify the capabilities or the conditions that must be performed or satisfied. MARTE can model two basic requirements of CPS QoS: timeliness and resources (Selić and Gérard, 2014). Timeliness is the ability to respond to events in the environment in a timely manner, which can vary greatly in different applications. Resources refer to the fact that program execution requires physical resources, such as processors and memory. Selić and Gérard (2014) presented a flexible co-modeling approach for CPS by combining SysML, MARTE, and pCCSL. With this, they modeled the CPS characteristics, including hardware/software coordination, continuous and stochastic behaviors, and non-functional properties (e.g., time and energy consumption).

Bocciarelli et al. (2017) presented a BPMN extension that is used to annotate several properties in BPMN models. These properties are related to performance (e.g., service time, response time, and throughput) and reliability (e.g., mean time to failure and mean time to repair). Such an extension addresses the performance- and reliability-oriented characterization of both CPS resources and their components.

Event-B/Hybrid Event-B and HCSP have also demonstrated their versatility in many industrial and academic projects (Banach et al., 2015, 2017; Zhan et al., 2016). In addition, many other tools, although not included in the introduction of commonly used tools in Section 4.1, can be used to meet some CPS requirements. For example, π-calculus can be used to deal with mobile computing; to achieve information security, π-calculus can be further improved into *spi*-calculus. Moreover, a labeled transition system can be extended to model some of the non-functional requirements, such as timed automata, hybrid automata, probabilistic automata, and stochastic hybrid automata.

5 Discussion

Although CPS is still in its infancy, the realms of embedded systems and hybrid systems are strongly CPS-related. Through analysis in this survey, we found that many model-driven techniques and tools that have been working well in the above realms, still work in CPS construction. Specifically, the proposed challenges are addressed by diverse tools. We can surely say that the model-driven approaches are a natural choice in building a CPS and of great potential in better coping with the challenges.

CPSs are typically multidisciplinary and highly complex. To perform a model-driven CPS development, we have to carefully choose a set of mature techniques and tools. However, these tools are heterogeneous themselves. Accordingly, integration of these tools becomes part of the C-III in CPS development and operations. Furthermore, because new devices and protocols may be involved in this CPS, we may have to define and implement special model-driven techniques (e.g., modeling languages) in certain scenarios. An inherent problem is how to have those in-use model-driven techniques (or tools) collaborate with the newly developed ones. Currently, a large number of model transformation instances have been used in practice, but require expertise to define and verify their correctness.

A better potential solution is to build an integrated highly scalable platform. This platform provides a unified modeling language (or data description language), with strict syntax and rich semantics, as a bridge linking other model-driven techniques and tools via models. If a mature or new model-driven tool (technique) needs to be integrated with this platform, only a bidirectional adaptor is required. An adaptor is in charge of mutual transformation between two kinds of models; one kind is the input (or output) model of the specific tool, and the other kind is the platform-specific model. With the help of the adaptor, this newly integrated tool (technique) can communicate with all the other model-driven tools (techniques) on this platform.

XML is used as the foundational data exchange standard on the Internet. However, it focuses only on data formats (syntactic) and does not consider the data semantics. Ontology, as a knowledge representation form, owns modeling languages such as RDF and OWL. Because ontology modeling

languages concern both syntactics and semantics, they can be adopted and used as the unified modeling language of the potential integrated platform.

Currently, two ongoing projects, OpenCAESAR (https://opencaesar.github.io/) and Karma (http://www.micromesh.com.cn/metagraph), are working on proposing platforms as described above. The challenges of constructing a complex CPS can be better addressed with this kind of platform, because a complex problem can be decomposed, layer by layer, into small and independent problems that can be specified by more specific-domain modeling languages. All domain modeling languages (models) can be integrated in the above platform by defining a suitable adaptor.

6 Conclusions

In this paper we first illustrated the CPS evolution course and challenges. A systematic review of background related to model-driven techniques and tools was also conducted. Then we discussed mainly the state-of-the-art model-driven approaches in building CPSs. Specifically, we carefully introduced the most representative model-driven techniques and tools, along with analysis of how they are applied to CPS construction. We also conducted comparative analyses on these techniques and tools from various dimensions including modeling languages, functionalities, and the supports addressing CPS challenges.

The limitations of this survey include: (1) CPSs are still in their infancy. As a result, CPS-specific engineering techniques and tools are still uncommon. We could only choose to study CPS-related techniques and tools (the majority of them are more commonly used in modeling embedded systems or hybrid systems). Due to the limitation of space, 10 representative techniques and tools were analyzed instead of introducing more related ones. (2) The paper may provide more benefits for CPS developers because we have paid more attention to analyses of the features the techniques and tools have provided from a technical perspective, instead of thinking where they will go from an academic viewpoint.

For future research, we foresee that unified modeling platforms such as OpenCAESAR and Karma could lead the potential academic and industrial research direction in CPS engineering, to which we will pay sustained attention. Moreover, as CPSs keep evolving (e.g., humans will be a kind of resource and capability), the related characteristics of the CPS will undoubtedly expand (more complex as humans join in). This will result in more challenges, and accordingly, more comparative dimensions could be used to analyze the existing model-driven approaches.

Contributors

Tie-xin WANG and Bin GU jointly designed the research. Bo LIU, Yuan-rui ZHANG, Xue-lian CAO, and Yu LIU processed the data. Bo LIU, Tie-xin WANG, and Bin GU drafted the manuscript. Yuan-rui ZHANG, Xue-lian CAO, and Yu LIU helped organize the manuscript. Bo LIU, Tie-xin WANG, and Xue-lian CAO revised and finalized the paper.

Acknowledgements

We wish to thank Prof. Nai-jun ZHAN for his technical support on this work, and Prof. Zhiming LIU for his great help on paper design and writing. In addition, Dr. Heng-jun ZHAO provided helpful information on HCSP and Event-B, and Dr. Xia ZENG shared her idea for organizing this paper. We also thank the anonymous referees for helping us improve the paper.

Compliance with ethics guidelines

Bo LIU, Yuan-rui ZHANG, Xue-lian CAO, Yu LIU, Bin GU, and Tie-xin WANG declare that they have no conflict of interest.

References

Abrial JR, 2005. The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge, UK.

Abrial JR, Butler M, Hallerstede S, et al., 2010. Rodin: an open toolset for modelling and reasoning in Event-B. *Int J Softw Tools Technol Transfer*, 12(6):447-466. https://doi.org/10.1007/s10009-010-0145-y

Amrani M, Lucio L, Selim G, et al., 2012. A tridimensional approach for studying the formal verification of model transformations. Proc IEEE 5th Int Conf on Software Testing, Verification and Validation, p.921-928. https://doi.org/10.1109/ICST.2012.197

Banach R, Butler M, Qin SC, et al., 2015. Core Hybrid Event-B I: single Hybrid Event-B machines. *Sci Comput Program*, 105:92-123. https://doi.org/10.1016/j.scico.2015.02.003

Banach R, Butler M, Qin SC, et al., 2017. Core Hybrid Event-B II: multiple cooperating Hybrid Event-B machines. *Sci Comput Program*, 139:1-35. https://doi.org/10.1016/j.scico.2016.12.003

Banerjee A, Kandula S, Mukherjee T, et al., 2012. BANDAiDe: a tool for cyber-physical oriented analysis and design of body area networks and devices. *ACM Trans Embed Comput Syst*, 11(S2):49. https://doi.org/10.1145/2331147.2331159

Bernardeschi C, Domenici A, Masci P, 2018. A PVS-Simulink integrated environment for model-based analysis of cyber-physical systems. *IEEE Trans Softw Eng*, 44(6):512-533. https://doi.org/10.1109/TSE.2017.2694423

Bocciarelli P, D'Ambrogio A, Giglio A, et al., 2017. A BPMN extension for modeling cyber-physical-production-systems in the context of Industry 4.0. IEEE 14th Int Conf on Networking, Sensing and Control, p.599-604. https://doi.org/10.1109/ICNSC.2017.8000159

Booch G, Rumbaugh J, Jacobson I, 1999. The Unified Modeling Language User Guide. Addison Wesley Longman Publishing Co., Inc., USA.

Boronat A, Carsí JA, Ramos I, 2006. Exogenous model merging by means of model management operators. Proc 3rd Workshop on Software Evolution Through Transformations: Embracing the Change, p.1-19. https://doi.org/10.14279/tuj.eceasst.3.8

Brau G, Navet N, Hugues J, 2017. Heterogeneous models and analyses in the design of real-time embedded systems—an avionic case-study. Proc 25th Int Conf on Real-Time Networks and Systems, p.168-177. https://doi.org/10.1145/3139258.3139281

Braun W, Casella F, Bachmann B, 2017. Solving large-scale Modelica models: new approaches and experimental results using OpenModelica. Proc 12th Int Modelica Conf, p.557-563. https://doi.org/10.3384/ecp17132557

Broman D, Lee EA, Tripakis S, et al., 2012. Viewpoints, formalisms, languages, and tools for cyber-physical systems. Proc 6th Int Workshop on Multi-paradigm Modeling, p.49-54. https://doi.org/10.1145/2508443.2508452

Brown AW, 2004. Model driven architecture: principles and practice. *Softw Syst Model*, 3(4):314-327. https://doi.org/10.1007/s10270-004-0061-2

Broy M, Cengarle MV, Geisberger E, 2012. Cyber-physical systems: imminent challenges. In: Calinescu R, Garlan D (Eds.), Large-Scale Complex IT Systems. Development, Operation and Management. Springer, Berlin, p.1-28. https://doi.org/10.1007/978-3-642-34059-8_1

Cabot J, Gogolla M, 2012. Object Constraint Language (OCL): a definitive guide. Proc 12th Int Conf on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-Driven Engineering, p.58-90. https://doi.org/10.1007/978-3-642-30982-3_3

Calegari D, Szasz N, 2013. Verification of model transformations: a survey of the state-of-the-art. *Electron Notes Theor Comput Sci*, 292:5-25. https://doi.org/10.1016/j.entcs.2013.02.002

Chen LP, Babar MA, Nuseibeh B, 2013. Characterizing architecturally significant requirements. *IEEE Softw*, 30(2):38-45. https://doi.org/10.1109/MS.2012.174

Chen X, Liu ZM, 2017. Towards interface-driven design of evolving component-based architectures. In: Hinchey M, Bowen JP, Olderog ER (Eds.), Provably Correct Systems. Springer, Cham, p.121-148. https://doi.org/10.1007/978-3-319-48628-4_6

Cicchetti A, di Ruscio D, Eramo R, et al., 2010. JTL: a bidirectional and change propagating transformation language. Proc 3rd Int Conf on Software Language Engineering, p.183-202. https://doi.org/10.1007/978-3-642-19440-5_11

Clarke EM, 2008. The birth of model checking. In: Grumberg O, Veith H (Eds.), 25 Years of Model Checking: History, Achievements, Perspectives. Springer, Berlin, p.1-26. https://doi.org/10.1007/978-3-540-69850-0_1

Czarnecki K, Helsen S, 2003. Classification of model transformation approaches. Proc 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, p.1-17.

Derler P, Lee EA, Vincentelli AS, 2012. Modeling cyber-physical systems. *Proc IEEE*, 100(1):13-28. https://doi.org/10.1109/JPROC.2011.2160929

Eidson JC, Lee EA, Matic S, et al., 2012. Distributed real-time software for cyber-physical systems. *Proc IEEE*, 100(1):45-59. https://doi.org/10.1109/JPROC.2011.2161237

ESTEREL, 2015. Efficient development of safe avionics software with DO-178C objectives using SCADE Suite®. ESTEREL Technologies. http://www.peraglobal.com/upload/contents/2015/11/20151113142739_85462.pdf [Accessed on Aug. 30, 2020].

Feiler PH, Gluch DP, Hudak JJ, 2006. The Architecture Analysis & Design Language (AADL): an introduction. Software Engineering Institute, p.145. https://doi.org/10.1184/R1/6584909

Fowler M, 2003. UML Distilled: a Brief Guide to the Standard Object Modeling Language (3rd Ed.). Addison-Wesley, Boston, USA.

Friedenthal S, Moore A, Steiner R, 2015. A Practical Guide to SysML: the Systems Modeling Language. Elsevier, Waltham, USA.

Fritzson P, 2014. Principles of Object Oriented Modeling and Simulation with Modelica 3.3: a Cyber-Physical Approach. John Wiley & Sons, Inc. https://doi.org/10.1002/9781118989166

Fulton N, Mitsch S, Quesel JD, et al., 2015. KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. Proc 25th Int Conf on Automated Deduction, p.527-538. https://doi.org/10.1007/978-3-319-21401-6_36

Gill H, 2008. From Vision to Reality: Cyber-Physical Systems. http://labs.ece.uw.edu/nsl/aar-cps/Gill_HCSS_Transportation_Cyber-Physical_Systems_2008.pdf [Accessed on Aug. 30, 2020].

Gunes V, Peter S, Givargis T, et al., 2014. A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Trans Int Inform Syst*, 8(12):4242-4268. https://doi.org/10.3837/tiis.2014.12.001

Halbwachs N, Caspi P, Raymond P, et al., 1991. The synchronous data flow programming language LUSTRE. *Proc IEEE*, 79(9):1305-1320. https://doi.org/10.1109/5.97300

He JF, Li Q, 2017. A hybrid relational modelling language. In: Gibson-Robinson T, Hopcroft P, Lazić R (Eds.), Concurrency, Security, and Puzzles: Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday. Springer, Cham, p.124-143. https://doi.org/10.1007/978-3-319-51046-0_7

Hehenberger P, Vogel-Heuser B, Bradley D, et al., 2016. Design, modelling, simulation and integration of cyber physical systems: methods and applications. *Comput Ind*, 82:273-289. https://doi.org/10.1016/j.compind.2016.05.006

Henriksson D, Elmqvist H, 2011. Cyber-physical systems modeling and simulation with Modelica. Proc 8th Int Modelica Conf, p.502-509. https://doi.org/10.3384/ecp11063502

Henzinger TA, 1996. The theory of hybrid automata. Proc 11th Annual IEEE Symp on Logic in Computer Science, p.278-292. https://doi.org/10.1109/LICS.1996.561342

Huang P, Jiang KQ, Guan CL, et al., 2018. Towards modeling cyber-physical systems with SysML/MARTE/pCCSL. IEEE 42nd Annual Computer Software and Applications Conf, p.264-269. https://doi.org/10.1109/COMPSAC.2018.00042

Jézéquel JM, Barais O, Fleurey F, 2009. Model driven language engineering with Kermeta. Proc 3rd Int Summer School Conf on Generative and Transformational Techniques in Software Engineering III, p.201-221. https://doi.org/10.1007/978-3-642-18023-1_5

Jirkovský V, Obitko M, Mařík V, 2017. Understanding data heterogeneity in the context of cyber-physical systems integration. *IEEE Trans Ind Inform*, 13(2):660-667. https://doi.org/10.1109/TII.2016.2596101

Jouault F, Bézivin J, 2006. KM3: a DSL for metamodel specification. Proc 8th IFIP WG 6.1 Int Conf on Formal Methods for Open Object-Based Distributed Systems, p.171-185. https://doi.org/10.1007/11768869_14

Jouault F, Allilaire F, Bézivin J, et al., 2008. ATL: a model transformation tool. *Sci Comput Program*, 72(1-2):31-39. https://doi.org/10.1016/j.scico.2007.08.002

Juric MB, Mathew B, Sarang P, 2004. Business Process Execution Language for Web Services: BPEL and BPEL4WS. Packt Publishing.

Kleppe A, Warmer J, Bast W, 2003. MDA Explained: the Model Driven Architecture: Practice and Promise. Addison-Wesley, Boston.

Kühne T, 2006. Matters of (meta-) modeling. *Softw Syst Model*, 5(4):369-385.

Larson BR, Chalin P, Hatcliff J, 2013. BLESS: formal specification and verification of behaviors for embedded systems with software. Proc 5th Int Symp on NASA Formal Methods, p.276-290. https://doi.org/10.1007/978-3-642-38088-4_19

Lasnier G, Cardoso J, Siron P, et al., 2013. Distributed simulation of heterogeneous and real-time systems. Proc IEEE/ACM 17th Int Symp on Distributed Simulation and Real Time Applications, p.55-62. https://doi.org/10.1109/DS-RT.2013.14

Lawley M, Steel J, 2005. Practical declarative model transformation with Tefkat. MoDELS Int Workshops Doctoral Symp, p.139-150. https://doi.org/10.1007/11663430_15

Lee EA, 2008. Cyber physical systems: design challenges. Int Symp on Object/Component/Service-Oriented Real-Time Distributed Computing. http://chess.eecs.berkeley.edu/pubs/427.html

Lee EA, 2010. CPS foundations. Proc 47th Design Automation Conf, p.737-742. http://chess.eecs.berkeley.edu/pubs/804.html

Lee EA, 2018. Modeling in engineering and science. *Commun ACM*, 62(1):35-36. https://doi.org/10.1145/3231590

Lee EA, Zheng HY, 2005. Operational semantics of hybrid systems. Proc 8th Int Workshop on Hybrid Systems: Computation and Control, p.25-53. https://doi.org/10.1007/978-3-540-31954-2_2

Lee EA, Niknami M, Nouidui TS, et al., 2015. Modeling and simulating cyber-physical systems using CyPhySim. Proc 20th Int Conf on Embedded Software, p.115-124.

Lee J, Bagheri B, Kao HA, 2015. A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manuf Lett*, 3:18-23. https://doi.org/10.1016/j.mfglet.2014.12.001

Leino KRM, 2007. Verification of Object-Oriented Software. The KeY Approach. Springer, Berlin.

Liu J, Lv JD, Quan Z, et al., 2010. A calculus for hybrid CSP. Proc 8th Asian Symp on Programming Languages and Systems, p.1-15. https://doi.org/10.1007/978-3-642-17164-2_1

Liu J, Li TF, Ding ZH, et al., 2019. AADL+: a simulation-based methodology for cyber-physical systems. *Front Comput Sci*, 13(3):516-538. https://doi.org/10.1007/s11704-018-7039-7

Liu Y, Peng Y, Wang BL, et al., 2017. Review on cyber-physical systems. *IEEE/CAA J Autom Sin*, 4(1):27-40. https://doi.org/10.1109/JAS.2017.7510349

Liu ZM, Chen X, 2016. Model-driven design of object and component systems. Engineering Trustworthy Software Systems, Cham, p.152-255.

Liu ZM, Bowen JP, Liu B, et al., 2019. Software abstractions and human-cyber-physical systems architecture modelling. 5th Int School on Engineering Trustworthy Software Systems, p.159-219. https://doi.org/10.1007/978-3-030-55089-9_5

Lunel S, Mitsch S, Boyer B, et al., 2019. Parallel composition and modular verification of computer controlled systems in differential dynamic logic. In: Ter Beek M, McIver A, Oliveira J (Eds.), Formal Methods—The Next 30 Years. Springer, Cham, p.354-370. https://doi.org/10.1007/978-3-030-30942-8_22

Lynch N, Segala R, Vaandrager F, 2003. Hybrid I/O automata. *Inform Comput*, 185(1):105-157. https://doi.org/10.1016/S0890-5401(03)00067-1

Maler O, Manna Z, Pnueli A, 1992. From timed to hybrid systems. In: de Bakker JW, Huizing C, de Roever WP, et al. (Eds.), Real-Time: Theory in Practice. Springer, Berlin, p.447-484. https://doi.org/10.1007/BFb0032003

Mallet F, 2015. MARTE/CCSL for modeling cyber-physical systems. In: Drechsler R, Kühne U (Eds.), Formal Modeling and Verification of Cyber-Physical Systems. Springer Vieweg, Wiesbaden, p.26-49. https://doi.org/10.1007/978-3-658-09994-7_2

Mallet F, Villar E, Herrera F, 2017. MARTE for CPS and CPSoS: present and future, methodology and tools. In: Nakajima S, Talpin JP, Toyoshima M, et al. (Eds.), Cyber-Physical System Design from an Architecture Analysis Viewpoint. Springer, Singapore, p.81-108. https://doi.org/10.1007/978-981-10-4436-6_4

Mellor SJ, 2004. MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston. http://cds.cern.ch/record/1505924 [Accessed on Aug. 30, 2020].

Mens T, van Gorp P, 2006. A taxonomy of model transformation. *Electron Notes Theor Comput Sci*, 152:125-142. https://doi.org/10.1016/j.entcs.2005.10.021

Miller J, Mukerji J, 2003. MDA Guide Version 1.0.1. Object Management Group.

Mohamed MA, Challenger M, Kardas G, 2020. Applications of model-driven engineering in cyber-physical systems: a systematic mapping study. *J Comput Lang*, 59:100972. https://doi.org/10.1016/j.cola.2020.100972

Müller A, Mitsch S, Retschitzegger W, et al., 2016. A component-based approach to hybrid systems safety verification. In: Ábrahám E, Huisman M (Eds.), Integrated Formal Methods. Springer, Cham, p.441-456. https://doi.org/10.1007/978-3-319-33693-0_28

Nazari S, Sonntag C, Engell S, 2015. A Modelica-based modeling and simulation framework for large-scale cyber-physical systems of systems. *IFAC-PapersOnLine*, 48(1):920-921. https://doi.org/10.1016/j.ifacol.2015.05.190

Oldevik J, Neple T, Grønmo R, et al., 2005. Toward standardised model to text transformations. Proc 1st European Conf on Model Driven Architecture: Foundations and Applications, p.239-253. https://doi.org/10.1007/11581741_18

Platzer A, 2008. Differential dynamic logic for hybrid systems. *J Autom Reason*, 41(2):143-189. https://doi.org/10.1007/s10817-008-9103-8

Platzer A, 2010. Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Berlin.

Platzer A, 2018. Logical Foundations of Cyber-Physical Systems. Springer, Cham.

Ptolemaeus C, 2014. System Design, Modeling, and Simulation Using Ptolemy II. http://ptolemy.org/books/Systems [Accessed on Aug. 30, 2020].

Qin ZJ, Do N, Denker G, et al., 2014. Software-defined cyber-physical multinetworks. Int Conf on Computing, Networking and Communications, p.322-326. https://doi.org/10.1109/ICCNC.2014.6785354

Rahim LA, Whittle J, 2015. A survey of approaches for verifying model transformations. *Softw Syst Model*, 14(2):1003-1028. https://doi.org/10.1007/s10270-013-0358-0

Rajkumar R, Lee I, Sha L, et al., 2010. Cyber-physical systems: the next computing revolution. Design Automation Conf, p.731-736. https://doi.org/10.1145/1837274.1837461

Saeedloei N, Gupta G, 2016. A methodology for modeling and verification of cyber-physical systems based on logic programming. *ACM SIGBED Rev*, 13(2):34-42. https://doi.org/10.1145/2930957.2930963

Sangiovanni-Vincentelli A, Damm W, Passerone R, 2012. Taming Dr. Frankenstein: contract-based design for cyber-physical systems. *Eur J Contr*, 18(3):217-238. https://doi.org/10.3166/ejc.18.217-238

Seidewitz E, 2003. What models mean. *IEEE Softw*, 20(5): 26-32. https://doi.org/10.1109/MS.2003.1231147

Selic B, 2003. The pragmatics of model-driven development. *IEEE Softw*, 20(5):19-25. https://doi.org/10.1109/MS.2003.1231146

Selić B, Gérard S, 2014. Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE. Elsevier, Amsterdam. https://doi.org/10.1016/C2012-0-13536-5

Sha L, Gopalakrishnan S, Liu X, et al., 2008. Cyber-physical systems: a new frontier. Int Conf on Sensor Networks, Ubiquitous, and Trustworthy Computing, p.1-9. https://doi.org/10.1109/SUTC.2008.85

Simko G, Levendovszky T, Maroti M, et al., 2014. Towards a theory for cyber-physical systems modeling. Proc 4th ACM SIGBED Int Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems, p.56-61. https://doi.org/10.1145/2593458.2593463

Tan Y, Vuran MC, Goddard S, 2009. Spatio-temporal event model for cyber-physical systems. Proc 29th IEEE Int Conf on Distributed Computing Systems Workshops, p.44-50. https://doi.org/10.1109/ICDCSW.2009.82

Tariq MU, Grijalva S, Wolf M, 2015. A service-oriented, cyber-physical reference model for smart grid. In: Khaitan SK, McCalley JD, Liu CC (Eds.), Cyber Physical Systems Approach to Smart Electric Power Grid. Springer, Berlin, p.25-42.

The Object Management Group, 2000. Meta-data Interchange (XMI) Specification. https://www.omg.org/spec/QVT/1.0/PDF [Accessed on Aug. 30, 2020].

The Object Management Group, 2006. Model Driven Architecture. http://www.omg.org/mda/ [Accessed on Aug. 30, 2020].

The Object Management Group, 2008. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. https://www.omg.org/spec/QVT/1.0/PDF [Accessed on Aug. 30, 2020].

The Object Management Group, 2013. Business Process Model and Notation (BPMN) Version 2.0.2.

Tratt L, 2005. Model transformations and tool integration. *Softw Syst Model*, 4(2):112-122. https://doi.org/10.1007/s10270-004-0070-1

Varró D, Balogh A, 2007. The model transformation language of the VIATRA2 framework. *Sci Comput Program*, 68(3):214-234. https://doi.org/10.1016/j.scico.2007.05.004

Varró D, Pataricza A, 2003. VPM: a visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics). *Softw Syst Model*, 2(3):187-210. https://doi.org/10.1007/s10270-003-0028-8

Wang BB, Baras JS, 2013. HybridSim: a modeling and co-simulation toolchain for cyber-physical systems. 17th IEEE/ACM Int Symp on Distributed Simulation and Real Time Applications, p.33-40. https://doi.org/10.1109/DS-RT.2013.12

Wang J, Zhan NJ, Feng XY, et al., 2019. Overview of formal methods. *J Softw*, 30(1):33-61 (in Chinese). https://doi.org/10.13328/j.cnki.jos.005652

Wang SL, Zhan NJ, Zou L, 2015. An improved HHL prover: an interactive theorem prover for hybrid systems. Proc 17th Int Conf on Formal Engineering Methods on Formal Methods and Software Engineering, p.382-399. https://doi.org/10.1007/978-3-319-25423-4_25

Wang TX, Truptil S, Benaben F, 2017. An automatic model-to-model mapping and transformation methodology to serve model-based systems engineering. *Inform Syst E-Bus Manag*, 15(2):323-376. https://doi.org/10.1007/s10257-016-0321-z

W3C, 1999. W3C Recommendation. https://www.w3.org/TR/1999/REC-xslt-19991116

Xue B, She ZK, Easwaran A, 2016. Under-approximating backward reachable sets by polytopes. Proc 28th Int Conf on Computer Aided Verification, p.457-476. https://doi.org/10.1007/978-3-319-41528-4_25

Xue B, Mosaad PN, Fränzle M, et al., 2017. Safe over- and under-approximation of reachable sets for delay differential equations. 15th Int Conf on Formal Modeling and Analysis of Timed Systems, p.97-115. https://doi.org/10.1007/978-3-319-65765-3_16

Younes AB, Hlaoui YB, Ayed LJB, 2014. A meta-model transformation from UML activity diagrams to Event-B models. Proc IEEE 38th Int Computer Software and Applications Conf Workshops, p.740-745. https://doi.org/10.1109/COMPSACW.2014.119

Zhan NJ, Wang SL, Zhao HJ, 2016. Formal Verification of Simulink/Stateflow Diagrams: a Deductive Approach. Springer, Cham, p.1-258.

Zhang LC, Feng SG, 2014. Aspect-oriented QoS modeling of cyber-physical systems by the extension of architecture analysis and design language. Proc Int Conf on Computer Engineering and Network, p.1125-1131. https://doi.org/10.1007/978-3-319-01766-2_128