The final publication is available at Springer via http://dx.doi.org/10.1007/s10270-015-0469-x

Cyber-Physical Systems Challenges—A Needs Analysis for Collaborating Embedded Software Systems

Pieter J. Mosterman[1], **Justyna Zander**[2]

[1] MathWorks, 3 Apple Hill Drive, Natick, MA 01760-2098, USA
pieter.mosterman@mathworks.com

[2] Worcester Polytechnic Institute, Worcester, MA, USA
dr.justyna.zander@ieee.org

Received: date / Revised version: date

Abstract Embedding computing power in a physical environment has provided the functional flexibility and performance necessary in modern products such as automobiles, aircraft, smartphones, and more. As product features came to increasingly rely on software, a network infrastructure helped factor out common hardware and offered sharing functionality for further innovation. A logical consequence was the need for system integration. Even in the case of a single original end manufacturer who is responsible for the final product, system integration is quite a challenge. More recently, there have been systems coming online that must perform system integration even after deployment—that is, during operation. This has given rise to the cyber-physical systems (CPS) paradigm. In this paper, select key enablers for a new type of system integration are discussed. The needs and challenges for designing and operating CPS are identified along with corresponding technologies to address the challenges and their potential impact. The intent is to contribute to a model-based research agenda in terms of design methods, implementation technologies, and organization challenges necessary to bring the next generation systems online.

1 Motivation

Engineered systems rely on ingenuity and technology to implement a desired functionality, examples of which include aircraft, automobiles, power plants, smartphones, robots, washers and dryers, pacemakers, and more. Embedded systems are engineered systems that implement functionality by employing computational technologies. The embedded nature allows the computational elements to interact directly (i) with a physical computing platform that it executes on and (ii) with its physical surroundings. In other words, computational logic may obtain input from sensors that measure physical quantities, execute physical instructions of a computing platform to compute output from this input, and provide the output to actuators that effect change in physical quantities and affect the physical behavior.

The intent of this paper is to explore the maturation of embedded systems and the evolution of the concept of cyber-physical systems (CPS). A result of this exploration is the identification of challenges specific to systems of a CPS nature. The perspective reflects upon an industry vantage point. Focus is on models for solving industry-relevant challenges when developing next-generation software systems. While the material is intended to be accessible to the broader research community, the scope is not intended to be comprehensive—instead, it focuses on functional aspects of CPS in particular.

S. Shyam Sunder, Director of the Engineering Laboratory at the National Institute of Standards and Technology provided the following definition of CPS on March 13, 2012:

> “Cyber physical systems are hybrid networked cyber and engineered physical elements co-designed to create adaptive and predictive systems for enhanced performance. Performance metrics include safety and security, reliability, agility and stability, efficiency and sustainability, privacy.”

CPS represent engineered systems that are built from a seamless integration of computational algorithms and physical components. The National Science Foundation (NSF) posits that advances in CPS will enable systems with capability, adaptability, scalability, resiliency, safety, security, and usability that will far exceed that of today’s embedded systems [59].

The study in this work aligns with the NSF perspective and highlights concrete elements of CPS that comprise a new type of methodological thinking. In par-

Send offprint requests to:

ticular, designing for new and unknown system functionalities is highlighted. Such functionalities, which become required or active not sooner than during the deployment of the system, challenge the traditional approach to system engineering. Novel system engineering approaches become a necessity, which in turn relies heavily on enabling system architectures and powerful design-time approaches to conceptualize system behavior at various levels of detail in different parts of the system [59]. The presented study documents a number of concrete challenges to successful CPS design, with emphasis on post-deployment integration issues in systems that collaborate to achieve required functionality. Related work presents specific examples for the collaboration challenges in the context of a pick-and-place machine that solves a distributed version of the Towers of Hanoi puzzle [42].

In Section 2 the transformation of the embedded systems notion into CPS is discussed. On the one hand, the transformation establishes a paradigm that is enriching the functionality. On the other hand, it is accompanied by enormous system integration challenges. Section 3 illustrates the challenges that arise from the nature of CPS. Potential solution approaches are provided as starting points for further investigation. Section 4 compares and contrasts the identified challenges with general challenges in software system engineering. Section 5 concludes the discussion with a summary.

2 Computation in Engineered Systems

This section reflects on how embedded systems have matured as increasingly sophisticated networking technology has become available. As nowadays system 'intelligence' allows systems to take the initiative, the design methods that account for the unknown scenarios continue to be a subject of study. A framework for engineering system architecture in a CPS paradigm is presented. The framework supports utilizing functionality of systems that are not yet available at the traditional system integration time, but only post-deployment. It deals with the scenarios when CPS are already in use, and reorganize or even redefine themselves in the field.

2.1 Maturation of Embedded Systems

In the early days of digital control (e.g., [12]), the sensed and actuated values were closely related with the logic implemented on a single digital control unit. For example, the sensed position value of a physical object and the actuated force value acting on it was strictly bounded to a control unit. However, the explosion of available computational power has affected the space of embedded systems. As a result, embedded systems have rapidly increased in complexity both in terms of their computing power as well as interaction with their surroundings. Moreover, computational features are increasingly interacting with one another. For example, in an automobile, a feature to automatically open and close windows may now interact with a feature that centrally locks doors. There may even be further interaction with a feature that detects a collision. Such interaction allows the windows to automatically close when the doors are locked, as well as the doors to automatically unlock and the windows to open when a collision is detected.

In modern systems, computation has become the technology of choice to implement control features of various forms (feedback control, sequence control, and supervisory control). It continues to be the preferred technology for the data analysis around the control functionality (voting schemes, debouncing, calibration, and others) [8]. Moreover, functionality of a systems nature (e.g., redundancy management, diagnosis, maintenance management, and more) is generally implemented with computational technology. Finally, the flexibility of software renders it uniquely suited for integrating the various systems in embedded systems.

A consequence of the prolific use of computational technology in interacting features and integration functionality of embedded systems is that the overall system complexity has grown by leaps and bounds. Software components that are dispersed throughout a modern embedded system interact with one another across well-defined interfaces. At a feature level, they also collaborate via their shared interaction with the physical world. Note that the physical world is comprised of both the surroundings that are sensed and actuated, as well as by the physical electronics execution platform. The challenges resulting from these characteristics are mostly of a 'systems' nature. Here, the power of computation can be exploited along another dimension by helping address the challenges [35, 53] from a design perspective. For example, a significant value proposition is that system integration issues can be tackled by enabling virtual system integration studies based on computational models.

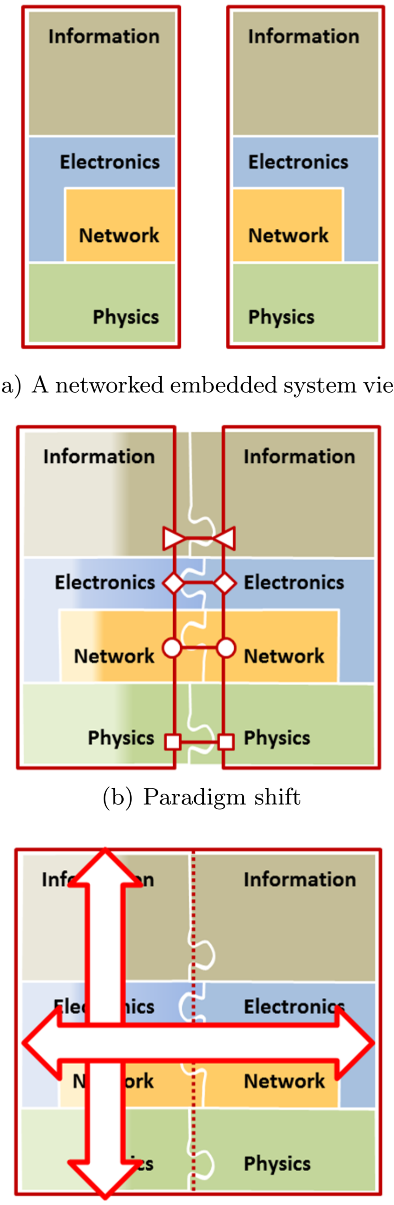
Key to enabling the analysis at the systems level is support for abstractions. This support makes the level of detail manageable. Model-Based Design [40, 43, 47] centers around abstractions by using computational models with executable semantics. A simulation engine enables executing the computational models and performing much of the system integration effort at a high level of abstraction. Moreover, support for model elaboration allows gradually adding detail to the models to the point where actual implementations may be made part of the overall system model. These practices are known as software-in-the-loop, processor-in-the-loop, and hardware-in-the-loop configurations [37]. A valuable additional benefit derived from having executable semantics is that either a hardware description or a software implementation can be generated automatically from the models [11].

2.2 Cyber-Physical Systems

More recently, a new breed of systems has emerged that challenges the classic design paradigm where the various features of a system are integrated by one original equipment manufacturer (OEM) [1]. Instead, as a consequence of the support for sophisticated decision logic, systems are now endowed with the capability to show initiative. In this new paradigm, systems will implement features by relying on collaboration with other systems that provide part of the required functionality. Moreover, this configuration occurs while being deployed in the field, typically by connecting the various collaborating systems across a network. Because of their ability to communicate in the corresponding cyberspace while operating in a physical environment, these systems are classified as CPS [51]. A recent example of CPS advances has been developed as part of the SmartAmerica Challenge, a U.S. government initiative, where a team of academics and practitioners unveiled a concept for the Smart Emergency Response System (SERS) [34,38,66]. SERS builds on the paradigm of *openness* and *dynamic configurability* to provide an extensive system of autonomous vehicles that automatically delivers provisions and rescue tools based on emergency responder requests. The resulting human-in-the-loop perspective further connects the system to survivors by locating them while communicating information and delivering assistance in the aftermath of a disaster.

As CPS are transforming the system integration paradigm, OEMs must adjust their business practices. Whereas in traditional networked embedded systems one OEM would be responsible for system integration prior to deployment, CPS do not integrate until after deployment. Figure 1 illustrates this paradigm transformation in the form of networked embedded systems architectures as designed by an OEM. Figure 1(a) shows two networked embedded systems. Each of them reflects the traditional paradigm where an information part executes as software on a number of computation platforms that are connected by a network fabric and embedded in a physical fabric. In addition to executing on physical media, the computing platforms connect to physical sensor and actuator transducer components. Further physical components that comprise the physical surroundings are presented along with the transducer components embedded in the physical fabric. A similar conceptualization is discussed in related work [26].

The separate networked embedded systems are increasingly being connected to one another, first in order to share data, but recently also to share functionality. From there, the computing platform and the network of connected systems become intensively shared as well. Finally, the physical world becomes shared and operated on in collaboration. According to the CPS paradigm shift, all of these layers are opening up to becoming available for collaborating systems. Now physics, electronics, information, and the network can be shared between multiple systems as illustrated in Fig. 1(b). This sharing mechanism contributes to such a tight integration that the systems become one emerging CPS as a system of systems. The result is illustrated in Fig. 1(c).



(a) A networked embedded system view

(b) Paradigm shift

(c) CPS paradigm analysis

Fig. 1 A networked embedded system view versus a cyber-physical system view

In detail, Fig. 1(b) presents the situation where elements of the various layers of the networked embedded system on the left-hand side are shared with the system on the right-hand side. The part of an architectural layer that is not shared is shown in a lighter color. Note that these shared elements can represent architectural solutions, behavioral functions, features, networks, or data, and they can have physical form factors. Such sharing leads to a tight coupling between architectural layers of the formerly separate systems and has a seemingly contradictory consequence. Whereas the layers become *closely knit* entities, a degree of *openness* between architectural layers is required. Both the tight coupling and the openness extend beyond the traditional paradigm of networked embedded systems.

At a system level, the tight coupling within architectural layers—with delineation across these layers—results in a general openness and interaction in a vertical and horizontal sense, as illustrated in Fig. 1(c). Vertically, the entire system may, for example, run on a different platform than initially planned. This situation occurs if new and better platforms evolve on other systems of which the original system is composed. Horizontally, some of the functionalities across any layer (or a combination thereof) originally owned by a native system can now be reused by any other system under design. Thus, Fig. 1(c) reflects a CPS paradigm where the actual implementation of some of the system platforms, components, and transducers is not yet available or may change at typical system integration time. Moreover, some of the physical and network fabric may be shared with other systems—the extent, behavior, and performance of which is still unknown.

2.3 Reflection

On the one hand, the CPS paradigm shift creates new and exciting opportunities for technology to be utilized in novel and unexpected manners and applications. On the other hand, it complicates matters in terms of shared resources, as illustrated in Fig. 1(c). The emerging complexity involves solving challenges such as shared energy usage, shared network connection, and dynamically changing availability of certain components, functionalities, and systems. To now account for this complexity in the design of CPS that is scalable and collaborative, time and resources management is required.

However, the opportunities still appear to far outweigh the technological challenges. Since CPS are designed to be open, they allow for device collaboration and functionality reuse. The open accessibility of reusable functionality enables a system developer to utilize a much broader range of computational benefits than in a closed situation. Finally, from a user perspective, the CPS-based products provide the ability to be part of an extensive ecosystem of features.

3 Challenges When Embedded Software Systems Collaborate

Based on the conceptual model in Fig. 1(c), challenges that arise in the CPS paradigm can be analyzed along two dimensions. Along the horizontal dimension, collaborating systems are established and dissolved at runtime—that is, while fielded. Challenges arise because of dynamically creating a configuration and because of the intent to achieve a concerted (integrated) function. Along the vertical dimension, the technology stack enables collaborating systems, which introduces challenges to provide the necessary infrastructure. In the proceedings, the analysis is scoped to functional aspects only so as to maintain focus and remain manageable. Therefore, though essential along the broad spectrum of CPS needs, matters such as maintainability, dependability, security, resilience, certification, policy, and others (e.g., [57,59]) are not addressed in this examination. Furthermore, the analysis is confined to specific challenges that are particular to collaborating embedded software systems. Issues that are found in networked embedded systems (e.g., load balancing, power/performance tradeoff, execution time analysis, heterogeneous targets, and so forth) are not included. Control specific challenges are discussed in related and complementary work [4].

3.1 Dynamically Creating a Configuration

Table 1 provides an overview of specific needs when dynamically configuring a system of collaborating systems, challenges that arise, technologies to address these challenges, and the impact of addressing each of the needs. Each entry is briefly described in this section.

Virtual System Integration From a design perspective, *virtual system integration* is a critical need as the physical system in all its potential variants does not become available until runtime. Central to addressing this need are challenges in terms of the detail that models capture, which may vary a lot depending on the task at hand (e.g., control synthesis vs. safety specification). Of particular importance is the need for a consistent, comprehensive, and accurate simulation-based approach to quality assurance methods. Pertinent advances in technology include property-based automated selection of model detail [13,32,60], counterexample guided refinement [9], and requirements guided selection of abstractions [21]. Further virtual system integration challenges stem from the heterogeneity of system multimodels, which falls under the purview of computer automated multiparadigm modeling [39] with specific value in combining dynamic semantics [33] and execution semantics [36]. Finally, selective design refinement calls for connectivity with implementation models (e.g., software-in-the-loop) or systems (e.g., hardware-in-the-loop), which relates to developments in real-time simulation [50].

Runtime System Adaptation A key implementation need is *runtime system adaptation*, which closely reflects the CPS paradigm. To adapt the system, it is necessary but

Table 1 Needs analysis for dynamic configuration of collaborating systems.

Need	Challenge	Technology	Impact
Virtual system integration	Proper models in design	Generation of models with necessary detail based on property selection	Confidently design systems as part of a reliable system ensemble.
	System-level design and analysis by using models	Connecting, combining, and integrating models represented in different formalisms	
		Efficient simulation models to be used across dynamic and execution semantics	
	Connectivity among models, software, and hardware	Open tool platforms with trusted interfaces for communication across synchronized and coordinated models, software, and hardware devices	
Runtime system adaptation	Reasoning and planning adaptation of an ensemble of systems	Introspection of the system state, configuration, and services it makes available	Exploit exogeneous functionality for efficient, economical, and resilient operation.
		Maintenance of consistent information and management of inconsistencies regarding the ensemble of systems with sufficient fidelity at runtime	
		Online model calibration	
	Testing with functionality on deployed systems	Environment models to enable surrogate interactions	

challenging to maintain the state of the current ensemble of subsystems, to be able to introspect and reason about potential changes, and to find an optimal configuration in the face of available resources. The real-time constraints make these challenges particularly difficult. Work on the use of models@run.time [7] provides runtime supervision and reconfiguration technologies that serve as valuable starting points. Models that capture the current and planned configurations must be kept consistent in the face of distributed architectures, and inconsistencies must be managed if necessary. The validity of models where the dynamic system environment requires online calibration is essential. This scenario occurs when adaptations at runtime are being planned (e.g., [20]). Finally, before a new configuration can be operated, it is essential to test the functionality. In an online setting, testing requires that surrogate interface behavior serve as a substitute for live interaction.

3.2 Achieving a Concerted Function

Table 2 provides an overview of specific needs for effective and efficient concerted collaboration, as well as the challenges, candidate technologies, and intended impact. Each entry is briefly described in this section.

Emerging Behavior Design An existential need is the ability to *design emerging behavior*. An example of this is a CPS version of the Towers of Hanoi [41, 44] where the individual discs rely on a local plan to determine their next move, and a sorted stack emerges. The behavior exhibited by the system ensemble is a result of concerted plans and actions of the collaborating systems. Emerging behavior is not explicitly designed into any one of the collaborating systems specifically. As such, a key challenge that relates to this type of design is to plan and control the collaboration so that the desired behavior emerges. This is particularly difficult in the face of loosely coupled and concurrently operating collaborators. A starting point for developing formal methods that address this challenge lies in elevating technologies developed based on the globally asynchronous/locally synchronous (GALS) paradigm (e.g., [31]) to the system of systems level. Further, advances in multi-agent systems [64] have developed an important and valuable body of work, while formal methods for design refinement have been illustrated in the context of Conway's Game of Life [54].

Data Sharing From an implementation perspective, a key need is *data sharing* with a focus on streaming data (e.g., from dislocated sensors). The challenge is that data may have incongruous sources that operate concurrently and at multiple rates. While traditionally synchronization of data streams is the responsibility of system integration, along with manifold further challenges [45], it now must apply at runtime and in real time. To address the challenges, an increasing use of models and monitoring capabilities may be imperative [35]. To share data

Table 2 Needs analysis for concerted function of collaborating systems.

Need	Challenge	Technology	Impact
Emerging behavior design	Collaborative planning, guidance, and control	Analysis methods across loosely coupled architectures	Systematically design systems that are part of a system ensemble to (optimally) realize desired ensemble behavior.
		Planning and synthesis of distributed control functionality on concurrent resources	
		Accessible formal methods that apply to collaborative problems	
Data sharing	Multirate architectures	Synchronization of data from incongruent sources	Effectively exploit distributed information resources for exclusive system features.
	Extracting and deriving specific value from general information	Information represented as high-level models with well-defined metamodels and ontologies	
Functionality sharing	Multi-use functionality post deployment	Generation of models for a desired task perspective by property identification and model behavior selection	Create novel system features post deployment.
		Performance characterization via performance models and measures	
		Online calibration based on objective and performance criteria	
	Feature interaction	Assumption formalization and dependency effect analysis	
Collaborative functionality testing	Systematic test suite generation and automated test evaluation	Model-based test generation from requirements while preserving the context of dynamic configurations	Assure the quality of collaborative operations on shared resources. Identify root causes of failure in a distributed environment. Automatically mitigate these root causes.
	Reproducible test results under minimum uncertainty	Setting of initial conditions and injecting fault data	
		Temporal and spatial partitioning to isolate functionality for a specific system architecture under investigation	

that contains information (e.g., models) with sophisticated data structures, a model of the structure that takes the form of a metamodel must be shared. Knowledge about the shared concepts is supported by exchange of corresponding ontologies. A step in this direction is the RoboEarth network where robots generate, share, and reuse data across Internet connections [65].

Functionality Sharing Beyond data, there is a need for *functionality sharing* among collaborating systems. Functional models may be dynamically processed to extract behavior from a given perspective and determine whether it can be functionally utilized. Moreover, the performance demanded by CPS puts rigorous constraints on assembling functionality in terms of latency, timing, error recovery, testability, and so forth. The shared functionality should characterize performance constraints (e.g., [29]) but also be amenable to a different use, which calls for parametrized implementation (e.g., low latency vs. high throughput), but also necessitates online calibration [28] of combined and interacting functionality (as well as models [20]). Finally, to reason about whether certain functionality may be applicable to serve a system purpose, the assumptions must be shared and the dependencies must be analyzable.

Collaborative Functionality Testing The emerging behavior from collaboration critically calls for *collaborative functionality testing*. The collaboration makes systematic test suite generation a challenge. Automated test data generation and systematic fault data injection requires formalized requirements for reliable testing. Because CPS operate in a physical world, safety mechanisms and fault mitigation strategies (also called fallback strategies) must be designed into the system. However, functionality that becomes dynamically available makes current testing approaches (e.g., to manage system variability [2]) challenging to apply. In addition, test behavior including the test evaluation must be reproducible and, if possible, automated. This demand is a particular challenge in a distributed system configuration as effective testing requires the initial state and test input to be set deterministically. Hence, whether to rely on stateless or stateful architectures presents a choice with important test implications [27]. Moreover, peripheral behavior that may influence the outcome of a test must be minimized if not eliminated so that the source for failure can be isolated. Separating functionality under test in temporal and spatial partitions is a potential solution, but it requires coordinating such partitioning among the collaborating systems.

3.3 Providing the Technological Infrastructure

Table 3 provides an overview of technological infrastructural needs and related challenges, technologies, and impact. Each entry is briefly described in this section.

Design Artifact Sharing In order to support virtual system integration, resources available from various disparate organizational sources must be accessible. This makes *design artifact sharing* an essential need. In the more general context of collaborating along the system lifecycle, the industrial Framework for Embedded Systems Tools (iFest) builds a services platform for tool coupling that is based on the Open Services for Lifecycle Collaboration (OSLC) infrastructure [56]. Beyond the services infrastructure, technologies for model abstraction selection, obfuscation and encryption, and so forth, are key. Projections in orthographic software modeling [3] efficiently support the various resulting views and their corresponding tools. Finally, as different tools rely on different implementations of execution semantics, consistent exchange of models necessitates models of the semantics of the execution engines (especially their numerical algorithms) [40].

Wireless Communication To support the dynamic configuring, there is a need for *wireless communication* that calls for a wireless network infrastructure. A key challenge is developing a standardized protocol stack that is sufficiently rich, yet configurable, to satisfy rigorous resource constraints (memory footprint, latencies, throughput, and so forth). Furthermore, integrating cyberspace calls for technology that is backward compatible with the Internet protocol (e.g, IEEE 802.15.4e [48]). Additionally, precise timing and synchronization between collaborating systems must be tackled to enable the real-time and embedded nature. Technologies such as the precision time protocol IEEE 1588 [10] and support for combined periodic/nonperiodic events [67] may be adapted to integrate into the network fabric.

Hardware Resource Sharing To support dynamic configurability and enable efficient implementation of functionality (as well as design resilience), *hardware resource sharing* among collaborating systems must be effective. One of the challenges is providing an infrastructure that enables the dispatch of functionality. A standardized technology stack, including dynamic task scheduling and machine instructions, holds promise (e.g., a real-time virtual machine [17]). In more flexible scenarios, the functionality is specified separately from the dispatch target architecture, which is the premise of platform-based design [5,46]. For testing, the implementation on a particular target may create challenging requirements. The reason is that certain distinctive behavior (e.g., overflow handling, instruction fetch, and so forth) that is idiosyncratic to a target may lead to complicated corner cases. Safety in the face of resource sharing is challenged by the heterogeneous nature of an ensemble of systems. This bears out in safety monitoring components (e.g., watchdogs and mitigators [23]) that must be aware of the particular semantics of time in the face of rigorous restrictions for using such safety assuring components [22]. Finally, as some elements of the system ensemble may require higher safety integrity level than others, interchangeability of a safety component and its dynamically changing allocation to hardware requires a thorough understanding of system operations and component interactions.

Service Utilization The need to leverage services of collaborating systems requires proper *service utilization*. Interesting developments in real-time service oriented architectures (SOA) [16,63] include real-time OSGi using Java [6]. Developments in real-time middleware (for a survey see [49]) include a real-time version of the High Level Architecture (HLA) [30], a real-time Common Object Request Broker Architecture (CORBA) (e.g., [25]), and the Data Distribution Service (DDS) [55]. Also, the robot operating system (ROS) [52] is of interest but currently lacking real-time capabilities. A challenging element here is discovering the functionality provided by services along with constraints such as real-time, memory, availability, requested guarantees, cost, and others [18] under which their utilization is possible. Note that the discovery must oftentimes be real-time in its own right. Further, a semantic layer is desired to enable mapping devices to semantic services. In related work, a service is associated with at least one semantic description in an Ontology Web Language (OWL) [19] used for describing semantic web services [58]. Services that provide different functionality, but that can still be applied or that can be orchestrated to provide the necessary functionality, must be discoverable. A taxonomy of services enables such discovery. With a related ontology, orchestration of services in a central coordinating architecture may then achieve emerging behavior, whereas in a decentralized control structure a choreography of the services may do so [24].

4 Software System Modeling Challenges in CPS

The new paradigm of collaborating systems and post-deployment (dynamic or runtime) system integration contributes to leading edge and intelligent applications. This

Table 3 Needs analysis for an infrastructure to support collaborating systems.

Need	**Challenge**	**Technology**	**Impact**
Design artifact sharing	Tool coupling among disparate organizations	Traceability across semantic and technology adaptation, and intellectual property protection	Conveniently, efficiently, and consistently collaborate between stakeholders throughout the system life cycle.
		Information extraction from obfuscated intellectual property	
	Support manifold views and tools in design	Configurable view projections that are tool specific	
		Consistent semantics across tools by modeling the execution engines	
Wireless communication	Physically aware protocol stack that is configurable and compatible with the Internet protocol (IP)	Real-time services of graded quality with a low footprint and a configurable protocol stack that includes time and location information	Reliably configure flexible system configurations for features with varying quality of service.
	Precise timing and synchronization in a distributed environment	Physical layer based timing and synchronization architectures	
		Scheduling of periodic and aperiodic events with reliable execution times	
Hardware resource sharing	Flexible and transferrable embedded functionality dispatch	Standardized and configurable real-time execution stack	Optimize runtime cost by contracting out system resources and balancing use of external resources. Enhance resiliency by allocating lost functionality to distributed resources.
	Performance characterization from abstract functional representations	Platform-based modeling of execution behavior functionality	
	Determination of key test cases for different implementations	Characterization of computational architectures	
	Safety of heterogeneous system ensembles	Modeling the semantics of time	
		Dynamically mixing safety integrity levels	
Service utilization	Real-time embedded services operating in a physical environment	Real-time middleware and service oriented architectures with physical capabilities	Dynamically assemble systems post deployment and purpose available functionality to serve singular needs.
	Smart services discovery	Service ontologies with taxonomies for similarity and transformability matching	
	Information sharing in a heterogeneous system ensemble	Language and ontology infrastructure to support translation and transformation	

section studies how software system modeling challenges identified in other work [14,15] bear out in CPS.

Generally, research in model-based approaches can be classified along the following axes:

- **Design** Model-Based Design requires good models, physics models of the plant, as well as that of the computing and communication platform and information models of the application. Yet, instead of the traditional approach to obtaining models by means of controlled system identification, the necessity to share models between distinctly incongruent entities and for interface-based model analysis becomes prominently important.
- **Implementation** Support for an open paradigm requires a well-defined and strict layering in the technology stack so as to enable including distinctly separate elements. Moreover, standards and interfaces are critical to addressing the specific implementation effects of the computing and communication platform. These implementation effects include, for example, finite precision quantization effects, saturation effects induced by fixed-point integers, scheduling effects, and delays caused by the scheduling of tasks.
- **Organization** Certification of safety-critical systems currently relies on a comprehensive evaluation of the system design. Systems that are open require reconsideration and adaptation of certification approaches. Allowing incremental and compositional evidence are essential and key elements to this end.

Not suprisingly, challenges in the general area of software system modeling [14,15] can be well aligned along the three axes. For example, *combining different modeling paradigms* [15] relates to the key aspect of CPS design. The models must represent the physical and computational world. Also, *domain-specific modeling* approaches stem from various design fields. CPS indeed relate mechanical engineering, electrical engineering, and electronics engineering while supporting means to analyze thermal effects, radio-frequency interference, and others. Moreover, *virtual system integration* plays an essential role in the CPS design because system parts may not be available until after deployment. In addition, such a system integration process requires developing a new understanding of *integration testing through simulation*. In particular, CPS call for more adaptive methods and on-the-fly mechanisms that can test the system at design, execution, and deployment levels.

Along the implementation axis, the open nature of CPS requires communicating information about its functionality and performance. This is best done by sharing specific models. Therefore, modeling support must extend beyond the traditional design stage into *later stages of the system lifecycle* as noted in related work [14]. Furthermore, CPS models are generally employed for *runtime system adaptation*. Additional challenges along the implementation axis extend *beyond the multicore computational platforms* to expand into a space of heterogeneous computing resources. For example, computational functionality that is shared by collaborating embedded systems may migrate from local to remote locations, possibly provided by *cloud-based services*.

Challenges along the organization axis reflect the confidence with which safety-critical systems are designed. Design approaches that enable *correct by construction* mechanisms are important research topics much like the challenge of *making formal methods accessible* to aid in establishing a next-generation certification process. Also, the complexity of CPS and the modern ways of modeling and organizing the design process call for more collaborative and incremental methods for quality assurance, including testing automation, verification, and self-testing abilities of the emerging CPS arrangements. Such methods can be capitalized on if proper *tooling for advanced artifact management* is in place.

More detail and further challenges, as well as opportunities, have been identified in related work [59, 62].

5 Conclusions

With the advent of inexpensive computing power, computation has increasingly become the technology of choice to implement features in products that range from consumer electronics to safety-critical power plants. In these products, computation is embedded in and tightly integrated with both a physical surrounding as well as the physical computing platform. As embedded systems matured, it became essential to factor out common hardware in a network infrastructure, which further provided an opportunity for innovation by sharing feature functionality.

For all the success that networked embedded systems can lay claim to, the corresponding computational complexity has introduced system integration issues that have been a continuous source of complications and even missed delivery deadlines (e.g., [61]). While in a traditional engineering paradigm, the OEM is responsible for the system integration, a cyber-physical systems (CPS) paradigm is transforming this convention. In CPS, the system integration occurs post deployment during operation of the system. Given the difficulty in system integration during the system development phases, it is no surprise that post deployment system integration is an engineering problem fraught with extreme complications such as feature interaction, multirate distributed architectures, and collaborative control.

The CPS paradigm as an ensemble of systems, dynamically establishing collaborations, has been developed. Needs and challenges to capitalize on this CPS paradigm have been identified with a focus on the functional characteristics. Technology directions to address the challenges have been presented along with potential impact of success. General software system modeling challenges have been related to specific CPS challenges. Related work presents specific examples of the collaboration challenges in the context of a pick-and-place machine that solves a distributed version of the Towers of Hanoi puzzle [42].

Beyond the functional characteristics, there are many challenges related to maintainability, dependability, security, resilience, certification, and policy. For example, dynamically adjustable quality assurance in terms of system safety, validation, verification, and certification constitutes a tremendous challenge for CPS. Because of the dynamic and flexible nature of CPS, current quality assurance methods are insufficient to satisfy and fulfill system reliability demands. Concepts such as self-diagnosing, self-evaluation, self-testing, self-correcting, safe self-reconfiguring, self-transforming, and self-approving are fields of study with great promise. Further out, automation of these quality assurance methods is an area with much discovery ahead and much progress to be made.

As the systems of tomorrow are becoming a reality, technologies with the most merit to overcome the challenges will prove their worth and be increasingly adopted. This work has intended to contribute to a model-based research agenda that helps bring next-generation systems online. Promising avenues to explore are identified to expedite the dawn of this new age.

References

1. acatech Position Paper. Cyber-physical systems. driving force for innovation in mobility, health, energy and production. acatech—National Academy of Science and Engineering, Munich, Germany, December 2011.
2. Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. A model-based testing methodology for the systematic

validation of highly configurable cyber-physical systems. In *The Sixth International Conference on Advances in System Testing and Validation Lifecycle*, pages 66–72, 2014.

3. Colin Atkinson, Dietmar Stoll, and Philipp Bostan. Orthographic software modeling: A practical approach to view-based development. In Leszek A. Maciaszek, César González-Pérez, and Stefan Jablonski, editors, *Evaluation of Novel Approaches to Software Engineering*, volume 69 of *Communications in Computer and Information Science*, pages 206–219. Springer Berlin Heidelberg, 2010.
4. John Baillieul and Panos J. Antsaklis. Control and communication challenges in networked Real-Time systems. *Proceedings of the IEEE*, 95(1):9–28, January 2007.
5. Felice Balarin, Massimiliano DAngelo, Abhijit Davare, Douglas Densmore, Trevor Meyerowitz, Roberto Passerone, Alberto Sangiovanni-Vincentelli Alessandro Pinto, Alena Simalatsar, Yosinori Watanabe, Guang Yang, and Qi Zhu. Platform-based design and frameworks: Metropolis and metro ii. Computational Analysis, Synthesis, and Design of Dynamic Systems, chapter 10, pages 259–322. CRC Press, Boca Raton, FL, 2009. ISBN: 9781420067842.
6. Pablo Basanta-Val, Marisol García-Valls, and Iria Estévez-Ayres. Enhancing osgi with real-time java support. *Softw., Pract. Exper.*, 43(1):33–65, 2013.
7. Nelly Bencomo, Robert B. France, Betty H. Cheng, and Uwe Aßmann, editors. *Models@run.time*, volume 8378 of *Lecture Notes in Computer Science (LNCS)*. Springer, Berlin, Germany, 2014.
8. Stuart Bennett and Derek A. Linkens. *Real-time Computer Control.* Peter Peregrinus, London, UK, 1984. ISBN 0-3090-8908-5.
9. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 154–169, 2000.
10. Todor Cooklev, John C. Eidson, and Afshaneh Pakdaman. An implementation of ieee 1588 over ieee 802.11b for synchronization of wireless local area network nodes. *Instrumentation and Measurement, IEEE Transactions on*, 56(5):1632–1639, Oct 2007.
11. Zhen Ru Dai. An approach to model-driven testing with uml 2.0, u2tp and ttcn-3. *PhD thesis, Technical University Berlin, ISBN: 978-3-8167-7237-8. Fraunhofer IRB Verlag*, 2006.
12. Richard C. Dorf. *Modern Control Systems.* Addison Wesley Publishing Co., Reading, MA, 1987.
13. John B. Ferris and Jeffrey L. Stein. Development of proper models of hybrid systems: A bond graph formulation. In François E. Cellier and José J. Granda, editors, *1995 International Conference on Bond Graph Modeling and Simulation (ICBGM '95)*, number 1 in Simulation, pages 43–48, Las Vegas, January 1995. Society for Computer Simulation, Simulation Councils, Inc. Volume 27.
14. Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.
15. Robert B. France and Bernhard Rumpe. The evolution of modeling research challenges. *Software and System Modeling*, 12(2):223–225, 2013.
16. Marisol García-Valls, Iago Rodríguez Lopez, and Laura Fernández-Villar. iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Trans. Industrial Informatics*, 9(1):228–236, 2013.
17. Zonghua Gu and Qingling Zhao. A state-of-the-art survey on real-time issues in embedded systems virtualization. *Journal of Software Engineering and Applications*, 5(4):277–290, 2012.
18. Joe Hoffert, Shanshan Jiang, and Douglas C. Schmidt. A taxonomy of discovery services and gap analysis for ultra-large scale systems. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, pages 355–361, New York, NY, USA, 2007. ACM.
19. Ian Horrocks and Peter F. Patel-Schneider. Knowledge representation and reasoning on the semantic web: Owl. Handbook of Semantic Web Technologies, chapter 9, pages 365–398. Springer, Berlin, Germany, 2011.
20. Yilin Huang, Mamadou D. Seck, and Alexander Verbraeck. Towards automated model calibration and validation in rail transit simulation. In Peter M. A. Sloot, G. Dick van Albada, and Jack Dongarra, editors, *Proceedings of the International Conference on Computational Science, ICCS*, pages 1253–1259, 2010.
21. Zhihao Jiang, Pieter J. Mosterman, and Rahul Mangharam. Requirement-guided model refinement. Technical Report MLAB-70, University of Pennsylvania, December 2014.
22. Bernhard Kaiser, Vanessa Klaas, Stefan Schulz, Christian Herbst, and Peter Lascych. Integrating system modelling with safety activities. In *Computer Safety, Reliability, and Security, 29th International Conference, SAFECOMP 2010, Vienna, Austria, September 14-17, 2010. Proceedings*, pages 452–465, 2010.
23. Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software - Volume 33*, SCS '03, pages 37–46, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
24. Ravi Khadka, Brahmananda Sapkota, Luís Ferreira Pires, Marten van Sinderen, and Slinger Jansen. Model-driven development of service compositions for enterprise interoperability. In *Enterprise Interoperability - Third International IFIP Working Conference, IWEI 2011, Stockholm, Sweden, March 23-24, 2011. Proceedings*, pages 177–190, 2011.
25. Yamuna Krishnamurthy, Chris Gill, Douglas C. Schmidt, Irfan Pyarali, Louis Mgeta, Yuanfang Zhang, and Stephen Torri. The design and implementation of real-time CORBA 2.0: dynamic scheduling in TAO. In *Proceedings of the 10th IEEE Real-time Technology and Application Symposium (RTAS '04)*, pages 121–129, 2004.
26. Edward A. Lee. Time for high-confidence cyber-physical systems, March 2012. *Invited Plenary Talk*, Perfomance Metrics for Intelligent Systems (PerMIS-12) Workshop, University of Maryland.
27. Philipp Leitner, Stefan Schulte, Schahram Dustdar, Ingo Pill, Marco Schulz, and Franz Wotawa. The dark side of

SOA testing: Towards testing contemporary soas based on criticality metrics. In *5th International ICSE Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2013, May 26, 2013, San Francisco, CA, USA*, pages 45–53, 2013.

28. Jesse Levinson and Sebastian Thrun. Automatic online calibration of cameras and lasers. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.
29. Jie Liu and Feng Zhao. Towards service-oriented networked embedded computing. Technical Report MSR-TR-2005-28, Microsoft Research, February 2005.
30. Thom McLean, Richard M. Fujimoto, and J. Brad Fitzgibbons. Middleware for real-time distributed simulations. *Concurrency - Practice and Experience*, 16(15):1483–1501, 2004.
31. Steven P. Miller, Mike W. Whalen, Dan OâĂŹBrien, Mats P. Heimdahl, and Anjali Joshi. A methodology for the design and verification of globally asynchronous/locally synchronous architectures. Technical Report NASA/CR-2005-213912, National Aeronautics and Space Administration (NASA), Langley Research Center, September 2005.
32. Pieter J. Mosterman and Gautam Biswas. Towards procedures for systematically deriving hybrid models of complex systems. In *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, pages 324–337, 2000.
33. Pieter J. Mosterman and Gautam Biswas. A hybrid modeling and simulation methodology for dynamic physical systems. *SIMULATION*, 78(1):5–17, 2002.
34. Pieter J. Mosterman, David Escobar Sanabria, Enes Bilgin, Kun Zhang, and Justyna Zander. A heterogeneous fleet of vehicles for automated humanitarian missions. *Computing in Science & Engineering*, 12:90–95, August 2014.
35. Pieter J. Mosterman, Jason Ghidella, and Jon Friedman. Model-based design for system integration. In *Proceedings of The Second CDEN International Conference on Design Education, Innovation, and Practice*, pages CD–ROM: TB–3–1 through TB–3–10, Kananaskis, Alberta, Canada, July 2005.
36. Pieter J. Mosterman, Jason R. Ghidella, and Elisabeth M. O'Brien. Model-coverage as a quality measure and teaching tool for embedded control system design. In *Proceedings of the 2007 Frontiers in Education Conference (FIE 2007)*, pages T3J–1–6, Milwaukee, WI, October 2007.
37. Pieter J. Mosterman, Sameer Prabhu, and Tom Erkkinen. An industrial embedded control system design process. In *Proceedings of The Inaugural CDEN Design Conference (CDEN'04)*, Montreal, Canada, July 2004. CD-ROM: 02B6.
38. Pieter J. Mosterman, David Escobar Sanabria, Enes Bilgin, Kun Zhang, and Justyna Zander. Automating humanitarian missions with a heterogeneous fleet of vehicles. *Annual Reviews in Control*, 38(2):259–270, 2014.
39. Pieter J. Mosterman and Hans Vangheluwe. Computer automated multi-paradigm modeling in control system design. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, pages 65–70, Anchorage, Alaska, September 2000.
40. Pieter J. Mosterman and Justyna Zander. Advancing model-based design by modeling approximations of computational semantics. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 3–7, Zürich, Switzerland, September 2011. keynote paper.
41. Pieter J. Mosterman and Justyna Zander. GitHub Repository: Towers of Hanoi in MATLAB/Simulink for Industry 4.0. Jan 2015. doi: 10.5281/zenodo.13977.
42. Pieter J. Mosterman and Justyna Zander. Industry 4.0 as a cyber-physical system study. *Software and System Modeling*, x(x):x–x, 2015.
43. Pieter J. Mosterman, Justyna Zander, Gregoire Hamon, and Ben Denckla. A computational model of time for stiff hybrid systems applied to control synthesis. *Control Engineering Practice*, 20(1):2–13, 2012.
44. Pieter J. Mosterman, Justyna Zander, and Zhi Han. The towers of hanoi as a cyber-physical system education case study. In *Proceedings of the First Workshop on Cyber-Physical Systems Education*, Philadelphia, PA, April 2013.
45. Gerrit Muller. Coping with system integration challenges in large complex environments. In *The Seventeenth International Symposium of the International Council on Systems Engineering INCOSE*, page paper ID: 7.1.4, Jun 2007.
46. Marco Di Natale. Specification and simulation of automotive functionality using AUTOSAR. Computational Analysis, Synthesis, and Design of Dynamic Systems, chapter 21, pages 523–548. CRC Press, Boca Raton, FL, 2012. ISBN: 9781439846650.
47. Gabriela Nicolescu and Pieter J. Mosterman, editors. *Model-Based Design for Embedded Systems.* Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, Boca Raton, FL, 2009. ISBN: 9781420067842.
48. Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys & Tutorials*, 15(3):1389–1406, July 2013.
49. Héctor Pérez and J. Javier Gutiérrez. A survey on standards for real-time distribution middleware. *ACM Computing Surveys*, 46(4):49:1–49:39, March 2014.
50. Katalin Popovici and Pieter J. Mosterman, editors. *Realtime Simulation Technologies: Principles, Methodologies, and Applications.* Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, Boca Raton, FL, 2012.
51. President's Council of Advisors on Science and Technology. Leadership under challenge: Information technology R&D in a competitive world. Technical report, Executive Office of the President of the United States, August 2007.
52. Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
53. Ahmed H. Sameh, George V. Cybenko, Malvin H. Kalos, Kenneth W. Neves, John Rischard Rice, Danny C. Sorensen, and Francis Patrick Sullivan. Computational

science and engineering. *ACM Computing Surveys*, 28(4):810–817, December 1996.

54. Jeff W. Sanders and Graeme Smith. Emergence and refinement. *Formal Aspects of Computing*, 24(1):45–65, 2012.
55. Douglas C. Schmidt, Angelo Corsaro, and Hans Hag. Addressing the challenges of tactical information management in net-centric systems with DDS. CrossTalk special issue on Distributed Software Development, May 2008.
56. Tiberiu Seceleanu and Gaetana Sapienza. A tool integration framework for sustainable embedded systems development. *IEEE Computer*, 46(11):68–71, 2013.
57. Bryant Walker Smith. Automated vehicles are probably legal in the united states. Technical report, The Center for Internet and Society, Stanford, CA, November 2012.
58. Zhexuan Song, Alvaro A. Cárdenas, and Ryusuke Masuoka. Semantic middleware for the internet of things. In Florian Michahelles and Jin Mitsugi, editors, *IOT*. IEEE, 2010.
59. Steering Committee for Foundations in Innovation for Cyber-Physical Systems. Foundations for Innovation: Strategic Opportunities for the 21st Century Cyber-Physical Systems—Connecting computer and information systems with the physical world. Technical report, National Institute of Standards and Technology (NIST), March 2013.
60. Jeffrey L. Stein and Loucas S. Louca. A component-based modeling approach for system design: Theory and implementation. In François E. Cellier and José J. Granda, editors, *1995 International Conference on Bond Graph Modeling and Simula tion (ICBGM '95)*, number 1 in Simulation, pages 109–115, Las Vegas, January 1995. Society for Computer Simulation, Simulation Councils, Inc. Volume 27.
61. Michael Sullivan. JOINT STRIKE FIGHTER–progress made and challenges remain. Technical Report GAO-07-360, United States Government Accountability Office, March 2007.
62. Simon J. E. Taylor, Azam Khan, Katherine L. Morse, Andreas Tolk, Levent Yilmaz, and Justyna Zander. Grand challenges on the theory of modeling and simulation. In Gabriel A. Wainer, Pieter J. Mosterman, Fernando J. Barros, and Gregory Zacharewicz, editors, *SpringSim (TMS-DEVS)*, page 34. ACM, 2013.
63. Wei-Tek Tsai, Yann-Hang Lee, Zhibin Cao, Yinong Chen, and Bingnan Xiao. Rtsoa: Real-time service-oriented architecture. In Elisabetta Di Nitto, Robert J. Hall, Jun Han, Yanbo Han, Andrea Polini, Kurt Sandkuhl, and Andrea Zisman, editors, *2nd IEEE International Symposium on Service-oriented System Engineering*, pages 49–56. IEEE Computer Society, October 2006.
64. Adelinde M. Uhrmacher and Danny Weyns. *Multi-Agent Systems: Simulation and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.
65. Markus Waibel, Michael Beetz, Raffaello D'Andrea, Rob Janssen, Moritz Tenorth, Javier Civera, Jos Elfring, Dorian Gálvez-López, Kai Häussermann, J.M.M. Montiel, Alexander Perzylo, Björn Schießle, Oliver Zweigle, and René van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics & Automation Magazine*, 18(2):69–82, 2011.
66. Justyna Zander and Wayne C. Haase. Hardware, software, and their computational interoperability for smart and effective firefighting, book chapter in Smart Firefighting Roadmap. National Fire Protection Agency, 2015.
67. Yuanfang Zhang, Christopher D. Gill, and Chenyang Lu. Configurable middleware for distributed real-time systems with aperiodic and periodic tasks. *IEEE Transactions on Parallel Distributed Systems*, 21(3):393–404, 2010.