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 on run-time supporting platforms for cyber physical systems[*]

Yuan SUN[†‡], Gang YANG, Xing-she ZHOU

(*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China*)

[†]E-mail: sunyuan@mail.nwpu.edu.cn

Received Sept. 24, 2016; Revision accepted Mar. 20, 2017; Crosschecked Oct. 12, 2017

Abstract: Cyber physical systems (CPSs) incorporate computation, communication, and physical processes. The deep coupling and continuous interaction between such processes lead to a significant increase in complexity in the design and implementation of CPSs. Consequently, whereas developing CPSs from scratch is inefficient, developing them with the aid of CPS run-time supporting platforms can be efficient. In recent years, much research has been actively conducted on CPS run-time supporting platforms. However, few surveys have been conducted on these platforms. In this paper, we analyze and evaluate existing CPS run-time supporting platforms by first classifying them into three categories from the viewpoint of software architecture: component-based platforms, service-based platforms, and agent-based platforms. Then, for each type, we detail its design philosophy, key technical problems, and corresponding solutions with specific use cases. Subsequently, we compare existing platforms from two aspects: construction approaches for CPS tasks and support for non-functional properties. Finally, we outline several important future research issues.

Key words: Cyber physical system (CPS); Run-time supporting platforms; Component; Service; Agent
https://doi.org/10.1631/FITEE.1601579 **CLC number:** TP311

1 Introduction

The cyber physical system (CPS) (Lee, 2008; Sha *et al.*, 2008; Rajkumar *et al.*, 2010; Li RF *et al.*, 2012) is a recently emerging concept describing a broad range of next-generation engineered systems that tightly integrate computation, communication, and physical processes. For the past few years, with the continuous development and integration of such technologies as embedded computing, wireless sensor networks, and networked control, many complex systems have appeared in various application fields, such as smart grid, Internet of vehicles, and intelligent manufacturing systems. These systems deeply intertwine cyber and physical spaces, and have significantly changed the manner in which people interact with the physical world.

At the same time, the inherent complexity of CPSs challenges their design and implementation (Lee, 2008). In a CPS, computation and physical processes are deeply coupled and continuously interact with each other. The events in the physical space are first reflected in the cyber space, where they are used to make control decisions that are returned to the physical space to adjust physical processes. To enable cooperation between computation and physical processes, we need to not only guarantee their correctness, reliability, and safety, but also consider the interplay between them. The highly dynamic nature of both cyber and physical spaces necessitates that reorganization and reconfiguration are essential for a CPS to adapt itself to time-varying contexts. In addition, time-predictability is very important for CPSs. All the above-mentioned actions need to be conducted in real time. As a consequence, developing

[‡] Corresponding author
[*] Project supported by the Integrated Science-Technology Innovation Plan of Shaanxi Province, China (No. 2015KTZDGY06-03)
ORCID: Yuan SUN, http://orcid.org/0000-0002-8386-014X

CPSs from scratch is not efficient.

One efficient way to construct CPSs is to use CPS run-time supporting platforms. CPS run-time supporting platforms facilitate deployment, execution, and monitoring of CPS tasks. Using such platforms as a base, developers can concentrate on the core missions of the CPS tasks, and do not have to spend much time on common fundamental issues, thereby immensely increasing the development efficiency.

In both academic and industry communities, much attention has been paid to CPS run-time supporting platforms. First, many research projects have been funded. In terms of the intelligent manufacturing system, SOCRADES (Karnouskos *et al.*, 2009), IMC-AESOP (Karnouskos *et al.*, 2010), and Artemis projects (Broy, 2013) are consecutively funded by the European Union (EU). In terms of the intelligent transportation system, important research projects include Moblie-C (Chen *et al.*, 2009), aDAPTS (Wang, 2008), and NTuCab (Seow *et al.*, 2010). In terms of smart grid, there have been many projects. One of the largest EU funded projects is the GRID4EU project (GRID4EU, 2012), which costs 54 million euros. Second, several technology standards have been proposed. The AUTomotive Open System Architecture Standard (AUTOSAR) (AUTOSAR, 2014) was proposed to describe automotive software. Several smart grid standardization studies have been conducted, such as US NIST IOP roadmap (Greer *et al.*, 2014), IEC SMB SG 3 (SMB Smart Grid Strategic Group, 2010), and Microsoft SERA (Microsoft, 2015).

Summarizing existing research efforts is foundation for further research on CPS run-time supporting platforms. In this paper, we comprehensively analyze and evaluate existing platforms. First, we classify existing platforms into three categories from the viewpoint of software architecture: component-based platforms, service-based platforms, and agent-based platforms. Next, we summarize existing work based on the above classification. For each type, we detail its design philosophy, key technical problems, and corresponding solutions with specific use cases. Then, we compare existing platforms from two aspects: construction approaches for CPS tasks, and support for non-functional properties. The comparison results show that existing platforms have both advantages and disadvantages as they are aimed at different application fields. To choose an appropriate platform architecture, the particular characteristics of the application field have to be well studied.

2 Related work

Many surveys have been conducted on CPSs to date. In general, these surveys can be roughly divided into two categories: surveys on general CPS problems and surveys on domain-related problems.

In early CPS studies, much attention was paid to general CPS problems. Wang and Xie (2011) summarized such issues as integration under heterogeneous environments, real-time, safety, and verification. Li RF *et al.* (2012) discussed the main challenges associated with CPSs from the viewpoint of the computing system, network system, as well as control system, and subsequently surveyed recent research advances in available theories and technologies that can be used to design a CPS. Shi *et al.* (2011) and Wan *et al.* (2011) summarized recent work from such technological viewpoints as energy control, security control, transmission and management, control technique, system resource allocation, and model-based software design. Khaitan and McCalley (2015) surveyed recent advances in design technologies, security, resilience, reliability, quality of service (QoS), and real-time. In addition, there exist several specialized surveys on specific general CPS technology issues, such as testing (Asadollah *et al.*, 2015), security (Wu *et al.*, 2016), robustness (Hu *et al.*, 2016), language-based approaches to CPS development (Soulier *et al.*, 2015), and self-adaptation in CPSs (Muccini *et al.*, 2016).

With the gradual deepening of CPS research, domain-related problems have begun to receive an increasing amount of attention. Gunes *et al.* (2014) briefly discussed the research efforts in several application fields, such as emergency response, air transportation, critical infrastructure, intelligent transportation, and robotic services. Moreover, more in-depth summaries in such domains as manufacturing (Monostori *et al.*, 2016), healthcare (Haque *et al.*, 2014), vehicular CPS (Jia *et al.*, 2015), and smart grids (Macana *et al.*, 2011) have been conducted.

It is clear that virtually all the technology problems associated with the development of CPSs from scratch are included in the surveys cited above. As discussed above, with the increasing difficulty in developing CPSs, building CPSs with the aid of run-time supporting platforms is more efficient. In recent years, an increasing number of studies have been conducted in this direction. However, few surveys have been conducted on these studies.

3 Classification

Software run-time supporting technologies, e.g., Fractal (Bruneton *et al.*, 2006), open services gateway initiative (OSGI) (Dobrev *et al.*, 2002), and Java agent development framework (JADE) (Bellifemine *et al.*, 2008), are more effective in handling the increasing complexity compared to general software. These technologies have gradually matured over many years of development. In recent years, these mature technologies have gradually been receiving the attention of researchers in the field of CPSs. At present, virtually all technologies used in existing CPS run-time supporting platforms are modified versions of these well-developed ones. From the viewpoint of software architecture, we classify existing platforms into three categories: component-based platforms, service-based platforms, and agent-based platforms. In this section, we just give overviews of the three types of platforms, and the concrete analyses of them are conducted in subsequent sections.

3.1 Component-based platforms

The design philosophy of component-based CPS run-time supporting platforms (Obermaisser and Huber, 2009; Dubey *et al.*, 2011; Bures *et al.*, 2013; Martínez *et al.*, 2013; Acosta *et al.*, 2014; Inam *et al.*, 2014; Levendovszky *et al.*, 2014; Ni *et al.*, 2014) is rooted in component-based software engineering. In these types of platforms, CPS components are employed to encapsulate low-level operations in both cyber and physical spaces, and CPS tasks comprise a set of CPS components. The critical technical problems in these types of platforms usually exist in aspects such as CPS component construction, construction and deployment of CPS tasks, and reconfiguration of CPS tasks. Provision of real-time support and decoupling are the main challenges of CPS components. The deployment mechanism is required to provide support for robustness, autonomy, and real-time. Because of the low flexibility of traditional components, integrating reconfiguration mechanisms into these types of platforms is very essential.

Distributed real-time managed systems (DREMS) (Levendovszky *et al.*, 2014) is a typical component-based CPS run-time supporting platform. It is aimed at distributed and mobile scenarios, e.g., clusters of satellites, or swarms of unmanned aerial vehicles (UAVs). As depicted in Fig. 1, the supporting software is distributed across the nodes of the system. Each node in the system manages multiple types of devices and the nodes can interact with each other through communication devices managed by themselves. CPS components are hosted by actors, which are specialized operating system (OS) processes. Actors can run in parallel, and can be migrated from node to node. Actors are configured and managed by the deployment manager, a privileged actor installed on each node of the system. The OS and middleware are responsible for performance isolation between actors of different tasks.

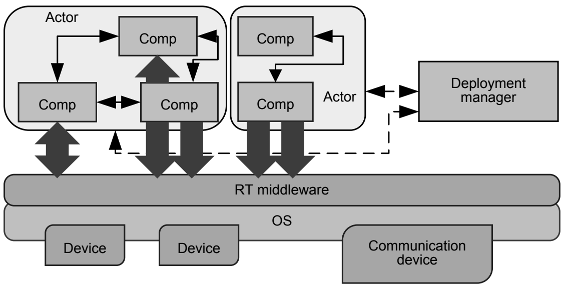

Fig. 1 A component-based CPS run-time supporting platform (Levendovszky *et al.*, 2014)
Comp: component; OS: operating system

3.2 Service-based platforms

The design philosophy of service-based CPS run-time supporting platforms (Cucinotta *et al.*, 2009; Huang *et al.*, 2009b; Mendes *et al.*, 2010; Vicaire *et al.*, 2010; Dillon *et al.*, 2011; Wang *et al.*, 2012) was borrowed from service oriented architecture (SOA) technology (Papazoglou and Heuvel, 2007). As a consequence, it inherits several advantages of SOA technology, such as loose coupling and flexibility. CPS services, which integrate the abilities of physical entities and related software, are treated as basic structural units of CPS tasks. In general, service-based platforms have three layers: service provision layer, service repository layer, and CPS task execution layer. In the service provision layer, the functions of both physical devices and related software are abstracted into CPS services. The main challenge in this layer is how to describe CPS services. CPS services are registered and discovered in the service repository layer. The CPS task execution layer provides an execution environment for CPS tasks. A service selection or composition mechanism is usually required in this layer.

Fig. 2 depicts a typical service-based CPS run-time supporting platform (Mendes *et al.*, 2010), which is designed for an industrial automation system. The industrial equipment in the system is managed by smart embedded devices, which interact with each other by the service bus. An automation entity is a software component inside a smart embedded device. Smart embedded devices serve as brokers between the industrial equipment and the platform. First, they are responsible for controlling and coordinating accesses to the industrial equipment. Second, they expose the functions of the industrial equipment as CPS services. Third, they can compose multiple CPS services into higher ones using their internal orchestration engines.

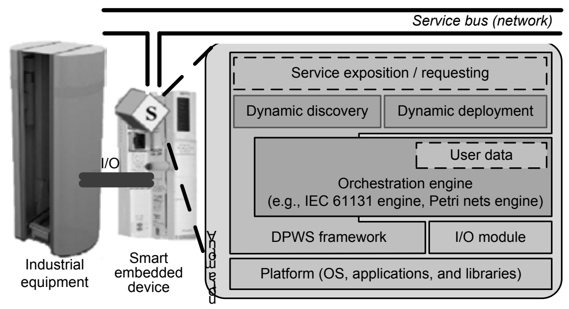

Fig. 2 A service-based CPS run-time supporting platform (Mendes *et al.*, 2010)
DPWS: devices profile for web services

3.3 Agent-based platforms

Agent-based CPS run-time supporting platforms (Lin *et al.*, 2011; Vrba *et al.*, 2011b; Leitão, 2013; Giordano *et al.*, 2016) are currently applied in fields where both local autonomy and global collaboration have to be considered. The underlying design philosophy of these types of platforms is derived from agent-based software engineering (Leitão *et al.*, 2013). CPS agents are used to manage autonomous entities in the system, and CPS tasks need the collaboration of the sets of CPS agents. CPS tasks are usually also designed as CPS agents—the so-called ‘CPS task agents’. CPS task agents can use other CPS agents in negotiations. CPS agent construction and evolution of CPS agents are the main technical points in these types of platforms. Traditional agents have shortcomings in aspects such as real-time, heterogeneity, and dynamics, which have to be considered and handled during CPS agent construction.

A typical agent-based CPS run-time supporting platform (Hsieh, 2010) is depicted in Fig. 3. It is designed for a holonic manufacturing system. The manufacturing resources, production management entities, and order management entities in the system are encapsulated as three different types of CPS agents. These CPS agents interact with each other by the agent communication language (ACL), which was proposed by the Foundation for Intelligent Physical Agents (FIPA). The directory facilitator is used to publish and discover the CPS agents’ abilities. The agent management system is responsible for naming, locating, and controlling the CPS agents. The message transport service is employed to communicate with other platforms.

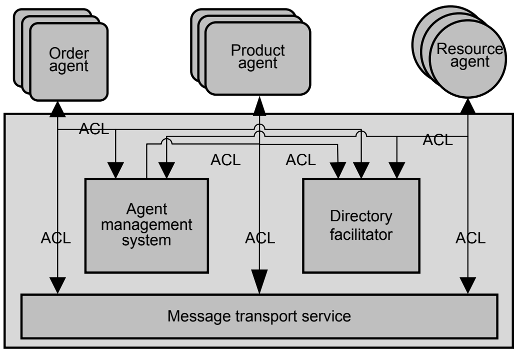

Fig. 3 An agent-based CPS run-time supporting platform (Hsieh, 2010)
ACL: agent communication language

4 Component-based CPS run-time supporting platforms

4.1 CPS component model

Existing studies on CPS components focus on the following two aspects: (1) In a CPS, non-functional properties such as real-time are equally important as functional features; (2) Existing components use interfaces to interact with each other, leading to strong dependence between components.

4.1.1 Real-time component

Currently, only a few CPS components support hard real-time operation. The ARINC-653 component model (ACM) (ARINC-653 is a software specification for space and time partitioning in safety-critical avionics real-time operating systems) (Dubey *et al.*, 2011) and real-time container component model (RT-CCM) (Martínez *et al.*, 2013) are two such components. Both components are based on the Lightweight CORBA (common object request broker architecture) component model (LwCCM) and are very similar. The differences between them lie in the number of ports and the component configuration. An RT-CCM component has only two kinds of ports, facetport and recepport, while an ACM component has two additional kinds of ports besides RT-CCM. In respect of component configuration, RT-CCM provides a series of configurable properties, which are assigned to concrete values during component instantiation. In addition to configurable properties, ACM provides another kind of property: state variables. The value of a state variable cannot be modified outside the component; therefore, it can be used to monitor the internal state of a component from outside.

To provide real-time support, an RT-CCM component is designed as an active component. Active components can not only be invoked through facetport, but also respond to external or timed events. These responses are implemented by the components themselves and must be declared beforehand. Moreover, these responses are triggered inside the components, and have nothing to do with invocations from other components; however, they can invoke other components that are connected to the recepport.

4.1.2 Independent component

The dependable emergent ensembles of components (DEECo) (Bures *et al.*, 2013) is aimed at dealing with the inherent difficulties in large-scale distributed CPSs, such as dynamics, openness, and autonomy. Compared to traditional components, a DEECo component system has strong independence. There are two key concepts in a DEECo component system: component and ensemble. Components are independent of each other regardless of whether the procedure is development, deployment, or execution. Components comprise knowledge and expose their functionalities as interfaces and processes. Ensembles are employed to link a group of components and manage their interactions. To reduce coupling and improve independence, DEECo components communicate with each other in the knowledge exchange process carried out by ensembles. Moreover, Bures *et al.* (2014) integrated gossip-based communication into DEECo components to further improve their independence.

4.2 Construction and deployment of component-based CPS tasks

In general, CPS tasks are first constructed and then deployed to run-time supporting platforms. In the construction procedure, the developer must decide on the composition of CPS tasks, connections between components, values of configurable properties, resources that have to be reserved, etc. Then, these pieces of information are used to generate a deployment plan, which guides the deployment procedure. The challenges in the construction and deployment procedures lie in aspects such as real-time (Martínez *et al.*, 2010; Inam *et al.*, 2014), robustness (Pradhan *et al.*, 2014), and autonomy (Pradhan *et al.*, 2014).

Inam *et al.* (2014) proposed a construction and deployment specification for real-time CPS tasks. The proposed specification is modified from the real-time CORBA specification. On one hand, metadata about the temporal behaviors and resource requirements of components are added to the real-time CORBA specification, and the component based software engineering-modelling and analysis suite for real-time applications (CBSE-MAST) (Lopez *et al.*, 2006) is used to analyze the temporal behaviors of both components and tasks. On the other hand, metadata about the resources in the platform are also added. On the basis of the metadata, the schedulability aanalysis of real-time CPS tasks can be performed. Thus, the feasibility of current CPS task construction procedures can be verified.

Martínez *et al.* (2010) presented a runnable virtual node (RVN) based two-stage component integration approach. In the proposed approach, at the first stage, RVNs are first constructed by mapping task sets to CPU-time partitions and assigning priorities to the task sets. Then, the time constraints of the RVNs are verified through simulation, test, or local schedulability analysis. At the second stage, RVNs are connected together, and the priorities of CPU-time partitions are assigned. Subsequently, a global dispatcher is employed to conduct global schedulability analysis. Finally, if the priority assignments are proved to be feasible, all codes generated in the two stages are compiled and linked with OS and middleware binaries. In this way, real-time CPS tasks are generated.

In distributed and mobile CPSs, e.g., clusters of satellites or swarms of UAVs, the deployment of CPS tasks is required to be conducted autonomously. This is necessary because the dynamic nature of these systems is so high that even the task deployment procedure could be influenced. Therefore, human intervention to address the difficulty is not very easy. Pradhan *et al.* (2014) proposed a resilient and autonomous deployment infrastructure in which all the nodes of systems are installed with three modules: deployment manager, component server, and group member monitor. The deployment infrastructure is composed of all the deployment managers. Group member monitors maintain group membership information and detect the failures of its members. The multi-staged deployment process employed is depicted in Fig. 4. All the nodes in the group have the opportunity to receive the deployment plan. The node that first receives the deployment plan becomes the deployment leader automatically. This leader is responsible for initiating the deployment process, analyzing the plan, and allocating deployment actions.

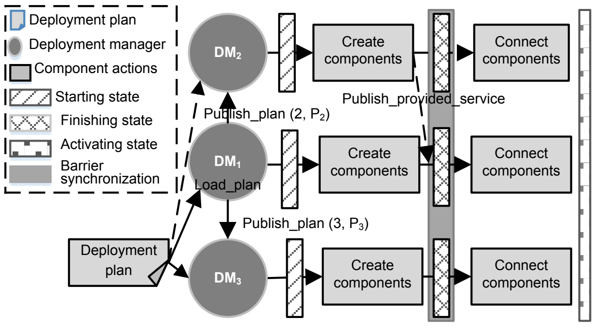

Fig. 4 A multi-staged deployment and configuration procedure (Pradhan *et al.*, 2014)
DM: deployment manager

4.3 Reconfiguration of component-based CPS tasks

As is well-known, CPSs operate in dynamic and uncontrollable environments. The topologies of CPSs are dynamic as failures of nodes or communication links can cause changes. Further, the available resources and missions of CPS tasks can change at times. This high dynamic nature makes reconfiguration of CPS tasks absolutely essential, especially for component-based CPS tasks.

A runtime@model (Morin *et al.*, 2009) based reconfiguration is used in u-kevoree (Acosta *et al.*, 2014). The concept of runtime@model is based on the idea of reflection. It allows changing the composition of tasks and connections between components. Specifically, u-kevoree provides four kinds of adaptation abilities: parametric adaptation, architectural adaptation, dynamic provision of component types, and adaptation for remote management. u-kevoree uses an adaptation engine as a core module to perform adaptation. During the reconfiguration process, a delta-model is first obtained through model comparison, and then a script for safe system reconfiguration is generated according to the delta-model. Finally, the script is executed to complete the adaptation.

Axelsson and Kobetski (2014) proposed a version updating based reconfiguration mechanism. In the proposed mechanism, all components are stored in a publicly accessible server. Further, components are downloaded to nodes according to the requirements of tasks. Inside nodes, components are executed in a special environment. The server is also responsible for maintaining components, including component recovery when failures occur, and updating components when new versions appear. Note that a task can be reconfigured only to a certain extent in this manner. Afanasov *et al.* (2014) presented a context-oriented reconfiguration mechanism. In their mechanism, the environment is abstracted as a set of contexts, and every context is associated with a group of behaviors. Layer functions, a core concept in this mechanism, are built on the basis of contexts and context-related behaviors. By embedding layer functions into existing components, a reconfiguration of CPS tasks is achieved.

5 Service-based CPS run-time supporting platforms

5.1 Description of CPS services

CPS services integrate operations in both cyber and physical spaces. Describing CPS services involves defining functionalities and non-functional properties of CPS services.

Traditional service models, such as web services description language (WSDL) (Curbera *et al.*, 2002) and web ontology language for services (OWL-S) (Martin *et al.*, 2005), cannot be directly employed to describe CPS services, because these models are designed for general software services and cannot describe the particular characteristics of CPS services. First, operations in the physical space are integrated into CPS services, and these operations are sensitive to the surrounding environment. Therefore, the usability of CPS services is closely associated with the environment. Second, some physical devices could provide more than one CPS service, but these CPS services cannot always execute at the same time due to the intrinsic features of the physical devices. Third, physical devices always have their own working areas. Beyond these areas, CPS services provided by the physical devices have no meaning.

Huang *et al.* (2009a) first extended OWL-S (Martin *et al.*, 2005) to develop a context-sensitive resource-explicit CPS service model. In the model, providers of CPS services are modeled as physical resources, which can be specified by three properties: profile, context, and services. A CPS service description consists of four parts: process, profile, grounding, and context. Further, both physical resources and services must declare their constraints. The relationships between the CPS services provided by the same physical resource, such as ‘concurrence’ and ‘exclusivity’, are important constraints of physical resources. CPS services not only inherit the constraints of their providers, but also can define their specific constraints.

Huang *et al.* (2009b) subsequently presented a context-sensitive service model based on physical entity ontology. In the service model, physical entities are hierarchically organized according to their relationships. In terms of context, two new constraints, context precondition and context effect, are incorporated and treated as a complement to traditional service provision constraints, namely precondition and effect. In general, service provision constraints in the model are divided into two categories: context-free constraints and context-sensitive constraints. The former are described by precondition and effect, whereas the latter are described by context precondition and context effect. The service models presented by Huang *et al.* (2010) and Wang *et al.* (2012) are similar to this service model.

Jin *et al.* (2014) proposed a service model that can define time- and space-related characteristics. Their model has three main concepts: device, resource, and service. All the main concepts are associated with the time or space property. The concept of available time is used to define the time slots when a CPS service is running because a CPS service is not always available. Because physical devices can provide CPS services only in certain areas, the concept of ‘working range’ is used to define it. Zhu *et al.* (2015) studied the effect of CPS services and proposed a concept called ‘AppliedTo’ to express the fact that after the execution of a CPS service, the context of some physical entities may change.

5.2 Discovery of CPS services

A service discovery mechanism is employed to efficiently find a set of published services that meet the given requirements. Because the scale of CPSs is continuously increasing (Stojmenovic, 2014), the main challenge of service discovery is how to keep its scalability. That is, as the system size increases, the discovery procedure should be as efficient as possible, and the computation and communication overhead should be as small as possible.

Universal plug and play (UPNP) (Miller *et al.*, 2001) is a device-level SOA technology. In UPNP, devices are abstracted as services, and service advertisement and discovery are performed using multicast messages. These multicast messages are transmitted periodically, resulting in rapid energy consumption and much unnecessary network traffic. Park *et al.* (2013) introduced the concept of control device manager (CDM) to handle UPNP. In each region of the system there is at least one CDM, which is responsible for managing all the devices in the region. A device needs only to register its services in one of the CDMs located in the region. In this way, these periodic multicast messages are transmitted merely over the network consisting of all the CDMs in the system. Thus, the number of messages and the energy consumption speed of the device's battery can be significantly reduced.

From the view of service registry, CDMs behave as decentralized service repositories because one CDM maintains only a part of the overall service information of the system. This kind of decentralized service registry was also used by Vicaire *et al.* (2010), Dillon *et al.* (2011), and Li *et al.* (2011). Sometimes a CDM is also called a 'discovery proxy' (Jammes *et al.*, 2005). A discovery proxy is usually hosted by the node with strong computation and storage capacities (Park *et al.*, 2013). If gateway nodes exist in the system, it can also be hosted by a gateway (Vicaire *et al.*, 2010; Dillon *et al.*, 2011; Li *et al.*, 2011).

Wang *et al.* (2012) considered centralized service repositories. As only one service repository contains the information of a huge number of services, searching for the required services in this kind of large dataset is very time-consuming. Thus, they proposed to build a service aggregation graph to speed up the service search process. In this method, services with similar service functionality are grouped into a service spanning tree (SST), and all the generated SSTs compose the service aggregation graph. Common clustering algorithms, such as *k*-means, could be employed to obtain the two-layer service aggregation graph, but Wang *et al.* (2012) did not provide details on how to use the service aggregation graph to achieve efficient service search.

Hellbruck *et al.* (2013) presented a named service bus to eliminate the need for explicit service registration or a global service repository. The basic idea underlying this method is discovery of a service by its name. In this method, CCN-daemons, which act as a bridge between services and service consumers, are key components. A service needs only to locally register itself at a CCN-daemon by its name prefixes. CCN-daemons store the mapping relations between name prefixes and services in forward information bases (FIBs). Service requests, encoded as CCN interests, are first submitted to the local CCN-daemon. If there is no locally available service, the local CCN-daemon looks for matching entries in its FIB and forwards the CCN interests to another CCN-daemon. Once obtaining the former CCN interests, the CCN-daemon performs the same operation as above until the required services are found.

5.3 Composition of CPS services

CPS tasks are composed of a number of CPS services. In general, the composition of CPS services falls into three categories: static composition, dynamic composition, and composition at design time.

In static composition, the operations of the CPS task and the functionality requirements in each operating step are clear, but the CPS service to be invoked in each step is unknown. At runtime, static composition is reduced to service selection. Because a CPS is a highly dynamic system, context information is very important for selecting the most appropriate CPS service.

Wang *et al.* (2012) proposed a static service composition mechanism that exploits the workflow business logic model to build an abstract process graph (APG) of the task. An APG is composed of a set of functionality services and their relations. The functionality services are simply placeholders that are subsequently replaced with appropriate concrete services. This service composition mechanism is composed of two phases: service filtering and service selection optimization. In the first phase, to eliminate improper services as early as possible, all the SSTs need to perform service filtering based on an elaborately constructed context filtering rule. Services matching the filtering rule are filtered out, and the remaining services are treated as candidate services. In the second phase, the selection procedure is reduced to a combinatorial optimization problem with QoS. Because the number of candidate services is very low, the combinatorial optimization problem can be solved quickly.

In dynamic service composition, nothing is known except the goal of the CPS task. In this situation, artificial intelligence (AI) planning can be used to perform service composition. Given the initial state of a task, its goal, and all possible operations, AI planning can be used to find an operation sequence fulfilling the goal of the task.

Huang *et al.* (2009b) proposed an iterative two-stage AI planning based dynamic service composition approach. In the proposed approach, the AI planning step consists of two stages: abstract composition and physical composition. In the abstract composition stage, CPS services are selected according to their functionalities, while context related constraints are omitted. An abstract planning framework is obtained at the end of abstract composition. In the physical composition stage, concrete CPS services are selected by considering context information. Note that it is an iterative approach. Whenever the goal of a CPS task or the precondition of any CPS service is not satisfied, an AI planning step is initiated.

Composition in design time (Puttonen *et al.*, 2008; Mendes *et al.*, 2010) is useful in scenarios where a CPS task needs to be analyzed before it is executed. In this composition approach, both the CPS services that will be invoked and the working process of the CPS task are known. Mendes *et al.* (2010) used this composition approach in a proposed industrial automation system. In their system, a CPS service is modeled as a Petri net, and thus the composition of the CPS services becomes synchronizations between two or among more Petri nets. Specifically, synchronization is realized by adding a connection logic that connects the transitions of two Petri nets.

6 Agent-based CPS run-time supporting platforms

6.1 CPS agents

General software agents have been investigated for decades, and many technical specifications for them have been proposed by FIPA. However, they are still unfit for CPSs because they do not support the particular characteristics of CPSs. In recent years, agents specialized for CPSs have received much attention from researchers.

First, a great majority of CPS tasks have very strict time constraints, and some low-level control tasks can be executed only on dedicated hardware. Provision of real-time support is an important problem. Several solutions have been proposed (Lepuschitz *et al.*, 2009; Vrba *et al.*, 2011b; Ferreira *et al.*, 2013). Vrba *et al.* (2011b) presented a real-time guaranteed CPS agent, called a ‘holonic agent’ (Fig. 5). A holonic agent contains three core modules: high-level control module (HLC), low-level control module (LLC), and control interface. The HLC makes high-level control decisions for the overall agent. The LLC directly interacts with actuators, and its control tasks directly run on programmable logic controllers (PLC). Therefore, real-time operation can be guaranteed. The HLC communicates with the LLC through a control interface.

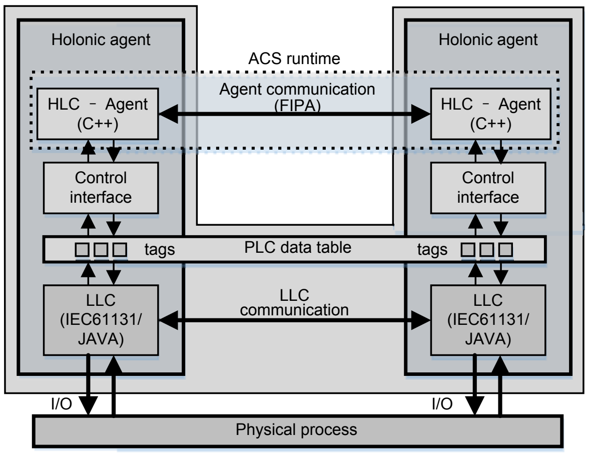

Fig. 5 The structure of a holonic agent (Vrba *et al.*, 2011b)
ACS: autonomous cooperative system; HLC: high-level control module; PLC: programmable logic controllers; LLC: low-level control module

Second, it is envisioned that with the emergence of more general CPSs, the demand to integrate a CPS with existing heterogeneous engineered systems, such as enterprise systems and wireless sensor networks (WSNs), will be very strong (Stojmenovic, 2014). Nevertheless, existing agents have more defects in interoperability compared to services. In recent years, enhancing the interoperability of CPS agents by incorporating them with services has raised concerns (Mendes *et al.*, 2009; Leitão, 2013). A service-oriented CPS agent for an intelligent manufacturing system was presented by Leitão (2013). Unlike traditional agents, the abilities of CPS agents are exposed as a number of CPS services. Such CPS agents not only can keep their autonomy, but also inherit strong interoperability from services.

Third, some researchers argue that existing agent communication technologies (e.g., ACL) can guarantee only syntactic interoperability. To enable such interoperability, several implicit semantics have to be embedded in implementations of existing agents (Vrba *et al.*, 2011a). Therefore, existing agents have difficulty in understanding new knowledge. Because a CPS is a highly dynamic system, when using existing agents in a CPS, improvement of their adaptation abilities is very important. Integrating CPS agents with explicit semantics is an effective approach (Al-Safi and Vyatkin, 2007; Lin *et al.*, 2011; Vrba *et al.*, 2011a). Such agents are sometimes called 'semantic CPS agents'. Semantic CPS agents can not only understand knowledge directly described by ontology, but also reason out new facts from ontology. If the ontology is extended with new knowledge, semantic CPS agents can understand the new knowledge immediately without any modification to their current implementations.

6.2 Evolution of CPS agents

With the scale of CPSs increasing and the autonomy of CPSs being enhanced, CPSs are faced with an increasing number of uncertainties, such as disturbances and failures. It requires not only that CPS agents be able to adjust their own behaviors, but also that a set of CPS agents can evolve through cooperation. In this aspect, many different approaches are available.

Hsieh (2010) investigated the evolution problem when resources fail. To avoid system disorders, they proposed an evolution mechanism based on four types of CPS agents: detector agent, initiator agent, standby agent, and optimizer agent. The CPS agent that first detects a resource failure becomes a detector agent. Because failures will inevitably block some agents, the CPS agent that is the first to be blocked becomes an initiator agent. An initiator agent has two duties. First, it must send requests to other CPS agents for discharge of contract; the CPS agents that agree with the discharge become standby agents. Second, it must send requests to the primary optimizer. The primary optimizer then appoints a CPS agent as an optimizer agent that performs optimal reconfiguration on the basis of available resources.

Leitão and Restivo (2006) proposed a reorganization mechanism combining hierarchical and heterarchical control approaches. In this mechanism, reorganization is defined as a transformation of the system between the static state and transitory state. In the static state, the system is hierarchically organized, and high-level agents autonomously manage its low-level agents. When disturbances or failures occur, agents first try to recover themselves. If a failure, they increase their autonomy factors and propagate their reorganization requirements to other agents in the form of pheromone. The agents that sense the pheromone continue to propagate the pheromone until some agent decides to reorganize the system. At this point, the system is transformed to the transitory state. In the transitory state, the system is heterarchically organized, and top-level agents directly manage the lowest-level agents. Then, the recovery procedure is initiated. When the pheromone disappears, the system is transformed to the static state again.

Self-organization and chaos theory have been employed to conduct evolution of CPS agents by Barbosa *et al.* (2015). Self-organization consists of behavioral self-organization and structural self-organization. Behavioral self-organization is smooth evolution. CPS agents can select appropriate behavior at any time according to their states and the environment. Structural self-organization is dramatic evolution. Its evolution goal is to maintain the stability of the overall system. The self-organization mechanism inside CPS agents is composed of three main parts as depicted in Fig. 6: monitoring module, discovery module, and reasoning module. The monitoring module is used to collect the system state. The discovery module is responsible for predicting new events according to the system state. Both events and the system state are employed to perform behavioral and structural reasoning. The reasoning results are entered into a nervousness stabilizer to avoid system disorders.

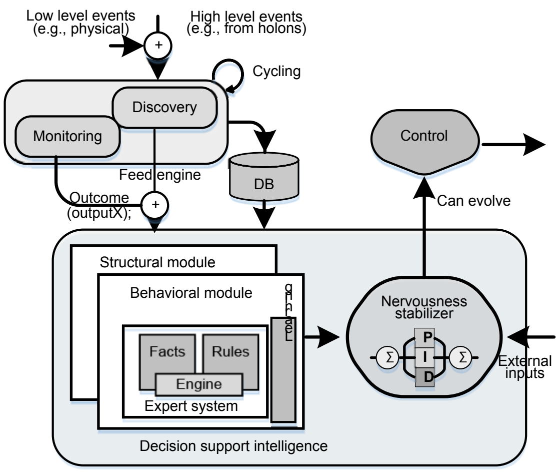

Fig. 6 A self-organization mechanism in a CPS agent (Barbosa *et al.*, 2015)
DB: database

7 Comparison

Existing CPS run-time supporting platforms are realized mainly for specific application fields. Their design and implementation measures differ significantly, leading to much difficulty for researchers to choose an appropriate reference. Thus, in the following subsections, we first compare existing CPS run-time supporting platforms, and then discuss the best platform architectures in different application scenarios.

7.1 Comparison of construction approaches for CPS tasks

In general, during CPS task construction procedures, three things need to be decided: (1) How to realize structural units of CPS tasks? (2) How to make structural units interact with each other? (3) How to compose a number of structural units into CPS tasks?

7.1.1 Realization of structural units

There are three kinds of measures to realize structural units of CPS tasks: CPS component, CPS service, and CPS agent. The main differences among the three types of structural units lie in aspects such as applicable scenario and realization difficulty. From the aspect of applicable scenario, CPS components and services are fit to encapsulate not only simple operations (Vicaire *et al.*, 2010; Fouquet *et al.*, 2012) but also complex operations comprising a number of simple operations (Cucinotta *et al.*, 2009; Martínez *et al.*, 2013; Levendovszky *et al.*, 2014), whereas CPS agents are fit to encapsulate more complex operations that are required to be controlled and managed by themselves (Lin *et al.*, 2011; Giordano *et al.*, 2016). From the aspect of realization difficulty, CPS agents are obviously more complicated than CPS components and CPS services, thus needing more time to realize.

7.1.2 Interaction between structural units

The interaction between structural units can be conducted in three ways: interface, standardized protocol, and knowledge. Interfaces, which define the data types of all input and output parameters, are primarily used to invoke CPS components. CPS services are invoked through a standardized protocol (e.g., WSDL). In essence, a standardized protocol can be treated as a kind of complex interface. Using this standardized method, we can clearly describe the structural units with more complicated functionalities, and can also perform various kinds of interactions. The communication between CPS agents can also be conducted through a standardized protocol (e.g., FIPA-ACL). Note that by using such standardized methods, we can decouple the interdependences between structural units to a substantial degree. To further lower the coupling, knowledge-based interaction is used (Kim *et al.*, 2013; Bures *et al.*, 2014), and in several situations, ontology is even employed to express the knowledge (Lin *et al.*, 2011).

7.1.3 Composition of structural units

The composition of structural units can be roughly divided into three categories: fully static composition (Cucinotta *et al.*, 2009; Vicaire *et al.*, 2010; Martínez *et al.*, 2013; Acosta *et al.*, 2014), fully dynamic composition (Huang *et al.*, 2009b), and dynamic-static hybrid composition (Wang *et al.*, 2012; Leitão, 2013). Fully static composition can be used for all kinds of structural units. Its characteristic is that structural units are statically connected with each other before running. At present, CPS tasks that need early analysis are usually constructed in this manner. For example, we can perform schedulability analysis during the fully static composition procedure.

Fully dynamic composition and dynamic-static hybrid composition are usually used for CPS services and CPS agents. CPS services and agents have not only static properties, but also dynamic properties, which are related to the context or environment. Because fully static composition is performed before the execution of a CPS task, it cannot use the dynamic properties. Nevertheless, fully dynamic composition and dynamic-static hybrid composition can employ both static and dynamic properties. In dynamic-static hybrid composition, the operational steps of the CPS task can be known before execution, but the structural units that will be invoked are not known before runtime. In fully dynamic composition, only the goal of the CPS task is known. Thus, AI planning is usually used to obtain the operating steps of the CPS task at runtime.

Moreover, not all CPS tasks are constructed by composition of structural units. In several platforms, CPS tasks can be built by programming frameworks using provided system-level application programming interfaces (APIs) (Srbljic *et al.*, 2012; Vicaire *et al.*, 2012; Kim *et al.*, 2013), and executed in corresponding run-time environments.

7.1.4 Summary

From the aspect of autonomy, CPS agents are better than CPS components and CPS services because autonomy is an essential attribute of CPS agents. In general, we consider that CPS services have stronger interoperability than CPS agents (Leitão, 2013), and that CPS agents have stronger interoperability than CPS components. If semantics is integrated, both CPS services and CPS agents will have the strongest interoperability.

Fully static composition can be used for all kinds of structural units, while fully dynamic composition and dynamic-static hybrid composition are usually used for CPS services and CPS agents. Because fully static composition cannot leverage dynamic properties, and fully dynamic composition and dynamic-static hybrid composition can employ dynamic properties, CPS services and CPS agents have stronger adaptability than CPS components. These comparision results are listed in Table 1.

Table 1 Comparison of three types of structural units

Structural unit	Autonomy	Interoperability	Adaptability
CPS component	●	●	●
CPS service	●●	●●●	●●
CPS agent	●●●	●●	●●

The number of black spots is used to indicate the degree of such aspects as autonomy, interoperability, and adaptability

7.2 Comparison of support for non-functional properties

The following performance indicators are used in the comparison: real-time, reconfigurability, scalability, context-awareness, resilience, and security.

7.2.1 Real-time

In general, there are two kinds of real-time activities: hard real-time activity and soft real-time activity. A hard real-time activity must always be completed before its deadline (Buttazzo, 2011); otherwise, the correctness of the entire system may be influenced. A soft real-time activity has fewer critical requirements; also, it should be completed before its deadline. If the deadline is missed, there will be no catastrophic outcome, excepting that the QoS of the system may be influenced.

Providing real-time support in run-time supporting platforms is onerous. Only a few platforms have such ability (Cucinotta *et al.*, 2009; Dubey *et al.*, 2011; Vrba *et al.*, 2011b; Martínez *et al.*, 2013; Wu *et al.*, 2016). An active component-based approach was presented by Martínez *et al.* (2013). To build real-time CPS tasks on the basis of their approach, a corresponding task construction and deployment mechanism is needed. Note that the real-time CPS task construction procedure is fully static and cannot be applied in highly dynamic scenarios. Cucinotta *et al.* (2009) proposed a QoS negotiation mechanism. With the web services agreement framework (Aiello *et al.*, 2005), service providers and service requestors can negotiate the QoS level. Because allocations of CPU time are based on a reservation-based scheduling framework, the key issue during the negotiations is the choice of appropriate scheduling parameters for service providers. If the current QoS level cannot be met, the service provider will decrease the QoS level in the next negotiation. Note that such a QoS negotiation mechanism is used only for soft real-time activities.

7.2.2 Reconfigurability

Existing research on adaptability is focused on two aspects: reconfiguration of CPS tasks and scalability of run-time supporting platforms. For CPS task reconfiguration, many different approaches exist. Acosta *et al.* (2014) employed a model comparison approach. In their approach, the difference between a new model and an old model, the so-called 'delta-model', is first compiled to a reconfiguration script. Then, the script is executed to reconfigure the CPS task. Vicaire *et al.* (2010) and Bures *et al.* (2013) conducted reconfiguration by adjusting the members of the group. Barbosa *et al.* (2015) investigated an evolution-based reconfiguration mechanism. Real-time operation is still important in the CPS task reconfiguration. Valls *et al.* (2013) proposed an approach for reconfiguration of service-based soft real-time systems. In their approach, reconfiguration is viewed as a transformation of a CPS task from an execution graph to another graph. In the approach, a reconfiguration manager first expands an application graph, and then performs an admission control test on the expanded graph to obtain a schedulable expanded graph. Finally, a service composition module chooses an appropriate path from the schedulable expanded graph as the new execution graph of the CPS task. All the operations are performed in a time-bounded manner.

7.2.3 Scalability

Many CPS scalability studies have been conducted (Vicaire *et al.*, 2010; Li *et al.*, 2011; Bures *et al.*, 2013; Hellbruck *et al.*, 2013; Park *et al.*, 2013; Bures *et al.*, 2014). The common method used to enhance scalability is replacing centralized control with decentralized control. Vicaire *et al.* (2010) and Li *et al.* (2011) used a gateway to locally manage some of the physical devices in the system. Park *et al.* (2013) chose devices with strong computation and storage capacity as discovery proxies for local regions. By adding more local managers, increases in the scale of the system can be easily accommodated. However, such a decentralized solution is not fit for highly dynamic scenarios. Consequently, knowledge-based approaches have been proposed. Bures *et al.* (2014) proposed integration of gossip-based communication into DEECo components. In the proposed approach, membership evaluations for DEECo components are conducted by propagating the knowledge of DEECo components. Thus, local managers are not needed. The named service bus (Hellbruck *et al.*, 2013) is also a knowledge-based approach. In this approach, the FIBs are constructed by sharing knowledge between all the nodes of the system.

7.2.4 Context-awareness

Context in CPSs refers to physical resources and environment. As we have discussed, physical resources and environment are more important in CPSs than in general software systems. First, CPS structural units are integrations of many low-level operations in cyber and physical spaces, such as sensing, computing, communicating, and actuating. Therefore, the execution of CPS structural units requires the participation of one or more physical resources. Second, the execution of CPS structural units has an influence on their surrounding environment, further influencing other CPS structural units.

The focus of existing physical resource studies is on management and modeling of physical resources. Nikam and Ingle (2014) argued that while composing a new CPS service, providing appropriate physical resources was very challenging because of the heterogeneity of physical resources and related CPS services. Therefore, they proposed a physical resource provision algorithm to deal with it. Huang *et al.* (2009b) proposed to incorporate a physical resource ontology into CPS service models. Huang *et al.* (2009a) proposed the concept of ‘service provision constraint’ to define the relationship among the CPS services provided by a physical resource. Wan *et al.* (2014) presented a resource-centric service model. In this model, a resource description template is employed to model physical resources.

The emphasis of existing environment research efforts is sensing and processing of environmental information. The majority of existing platforms have the ability to sense the environment. In the DEECo component system (Bures *et al.*, 2013), environmental information sensed by components is represented as knowledge and stored in the knowledge repository layer. Because the execution of CPS services has an influence on the environment, many CPS service models are incorporated with environmental information, such as the environment related precondition and effect. In the architecture of Rainbow (Giordano *et al.*, 2016), sensors and actuators are abstracted as virtual objects (VOs), and a computational node manages several VOs. Environmental information sensed by VOs feeds into CPS agents in computational nodes.

Under certain circumstances, raw environmental information is less meaningful, and must be further processed to obtain more meaningful results. The most common type of processing used is complex event processing (CEP), which generates complex events from raw environmental information in accordance with the predefined rules. Many studies (Tan *et al.*, 2009; Ahmadi *et al.*, 2010) have been conducted on CPS CEP. These studies have been adequately surveyed by Li F *et al.* (2012). Among existing platforms, only a few platforms, e.g., Rainbow (Giordano *et al.*, 2016), are explicitly integrated with a CEP mechanism. However, it is worth pointing out that the proposed CEP technologies can be easily incorporated into any existing platform when necessary.

7.2.5 Resilience

CPSs are expected to provide correct behavior under failures. Many different approaches have been proposed for this resilience aspect.

Several researchers have proposed algorithms and techniques for enhancing the resilience of CPSs toward different kinds of failures. Parvin *et al.* (2013) proposed to use multi-computational units to handle the failures of computing subsystems in CPSs. Woo *et al.* (2008) presented a software engineering methodology to design resilient CPSs. Specifically, Woo *et al.* used feedback control laws to define a protocol that makes CPSs resilient to software failures. Andersson *et al.* (2008) presented a distributed algorithm that transmits and receives physical information in the presence of sensor faults.

Some researchers have proposed mechanisms that make CPSs resilient to multiple kinds of failures. In wireless sensor actuator systems, the topology, links, and nodes are all unreliable, and multiple kinds of failures may occur. To make this kind of CPSs resilient, Pajic *et al.* (2012) introduced the embedded virtual machine (EVM), a programming abstraction in which controller tasks are maintained across physical node boundaries and functionality is capable of migrating to the most competent set of physical controllers. The corresponding EVM-based algorithms allow network control algorithms to operate seamlessly over less reliable wireless networks with topological changes. Xiao *et al.* (2008) proposed an external coordination layer that is used to separate fault-tolerance mechanisms from business logics. The coordination layer is also used to manage the fault-tolerant mechanism. Therefore, it is possible to use different kinds of fault-tolerant mechanisms to handle different kinds of failures.

7.2.6 Security

Security refers to the ability to ensure the privacy of data, control access, and resist attacks in CPSs. To the best of our knowledge, only a few existing platforms support security. DREMS (Otte *et al.*, 2014) employs a secure transport (ST) mechanism to ensure secure information exchange. Specifically, ST is a network transport layer that enforces information flow partitions based on security classifications. Vegh and Miclea (2016) proposed a secure CEP mechanism that uses a public-key algorithm. To ensure that each user has limited access to the data, the access level is granted by the private key possessed by each user.

7.2.7 Summary

Most of existing platforms have varying levels of reconfigurability. In general, agent-based platforms have the strongest reconfigurability, whereas component-based platforms have the weakest reconfigurability. Naturally, there are several exceptions. For example, as u-kevoree (Acosta *et al.*, 2014) uses a runtime@model-based reconfiguration mechanism, it can achieve the same reconfigurability as common service-based platforms.

As discussed above, several studies have been conducted on scalability, but the majority of them are not integrated into existing platforms. In general, knowledge-based approaches can obtain better scalability than decentralized control-based approaches.

Almost all of the platforms are able to leverage context information and handle failures. In general, the platforms using centralized management (the majority of component- and service-based platforms) have lower resilience than the platforms using decentralized management (the majority of agent-based platforms).

Real-time is supported only by several platforms. As far as we know, only a few existing platforms support security. However, it is worth pointing out that existing security solutions can be easily integrated with any existing platform when necessary.

Table 2 compares several typical CPS run-time supporting platforms in terms of non-functional properties support.

7.3 Best platform architectures in different application scenarios

In this section, we consider such representative CPS application scenarios as vehicle electronic systems, intelligent manufacturing systems, smart grids, swarms of UAVs, and intelligent transportation systems.

A vehicle electronic system consists of many electronic control units, such as engine control, fuel control, antiskid control, and brake control. Among these control units, engine control has the highest real-time deadline, as the engine itself is a very fast and complex part of a vehicle. Many engine parameters, such as pressure, temperature, flow, engine speed,

Table 2 Comparison of several typical CPS run-time supporting platforms in terms of support for non-functional properties

Typical platform	Structural unit	Real-time	Reconfigurability	Scalability	Context-awareness	Resilience	Security
DREMS (Levendovszky *et al.*, 2014)	CPS component		●		●	●●	●
RT-CCM (Martínez *et al.*, 2013)	CPS component	●			●	●	
DEECo (Bures *et al.*, 2013)	CPS component		●	●●	●	●●	
Physicalnet (Vicaire *et al.*, 2010)	CPS service		●	●	●●●	●	
Huang *et al.* (2009b)	CPS service		●●		●●●	●	
Cucinotta *et al.* (2009)	CPS service	●	●●		●	●	
Rainbow (Giordano *et al.*, 2016)	CPS agent		●●●		●●	●●●	
Vrba *et al.* (2011b)	CPS agent	●	●●●		●	●●●	

The performance indicators are measured by the number of black spots

and oxygen level, must be actively monitored and controlled in real time. Thus, it is clear that real-time speed is the most typical feature of vehicle electronic systems. Because component-based platforms can support hard real-time CPS tasks, they best fit vehicle electronic systems.

As a new-generation manufacturing system, the intelligent manufacturing system (IMS) is faced with strong demand to swiftly adapt to dynamics because mass customization is becoming an appealing production mode. The main characteristics of mass customization are dynamic requirements regarding lot sizes, product variants, lead time, and cost. One of the difficulties in building an IMS is that much legacy equipment exists in manufacturing enterprises. Although legacy equipment might have lower production capacity and might be less flexible than new intelligent equipment, replacing all legacy equipment in a short time is still uneconomical. Therefore, interoperability (compatibility with legacy equipment) is also very important in IMSs. Although both CPS services and CPS agents can deal with the dynamic nature of IMSs, we consider that service-based platforms are more suitable for IMSs because CPS services have stronger interoperability than CPS agents.

Smart grid is the next-generation electrical grid in which information and communication technologies are used to make production, distribution, and use of electricity more efficient and more cost-effective (Fang *et al.*, 2012). Compared with a conventional electrical grid, a smart grid is integrated with more renewable energy sources (mainly solar and wind) and new energy consumers, such as electric vehicles and smart home appliances. Because of the volatile and stochastic nature of these new energy sources and consumers, a smart grid is highly dynamic. These result in current centralized automation software control power grids (primarily supervisory control and data acquisition systems) reaching their limits in terms of scalability, computational complexity, and communication. Inevitably, in future smart grids, to provide more flexibility and scalability, decentralized software architectures with stronger local autonomy will be more appealing (Vrba *et al.*, 2014). From this viewpoint, agent-based platforms are the best fit for smart grids.

Swarms of UAVs are representative highly dynamic and distributed CPSs. A single UAV has very limited resources and can complete only very simple tasks. Thus, several UAVs often form a cluster to complete complex tasks. Swarms of UAVs often operate in risky and unknown environments; hence, it is impossible to make plans in advance as not enough information is available. To handle unknown environments, a single UAV must have strong local autonomy. Hence, we consider that agent-based platforms are the most suitable for swarms of UAVs.

The intelligent transportation system (ITS) has been proposed to leverage information and communication technologies to reduce traffic congestion and improve traffic safety. ITSs have three specific characteristics (Chen and Cheng, 2010): (1) ITSs are geographically distributed; (2) Vehicles exist in dynamic environments; (3) Vehicles need to interact with each other flexibly. Because of such characteristics, vehicles in ITSs have to increase their autonomies. Therefore, using an agent-based platform is a natural choice.

In general, in situations where the demand for real-time support is strong, such as vehicle electronic systems, using a component-based platform is the best choice. In dynamic environments, using either a service-based platform or an agent-based platform is feasible. However, if interoperability is also strongly needed, such as in IMSs, service-based platforms are the best. If local autonomy is obvious, such as in smart grids, swarms of UAVs, ITSs, agent-based platforms are the best. The results are also listed in Table 3.

8 Conclusions and future research issues

In this paper, we comprehensively analyzed and evaluated existing CPS run-time supporting platforms. The following conclusions can be drawn. First, existing platforms have both advantages and disadvantages because they aim at different application fields. Second, to choose an appropriate platform architecture, the particular characteristics of the application field have to be well known. Specifically: (1) if real-time support is very important, the best choice is a component-based platform; (2) in general, a service-based platform or an agent-based platform is better for dynamic scenarios; (3) if interoperability is also very essential in dynamic scenarios, a service-based platform is better than an agent-based platform; (4) if the system exists in a dynamic environment and has strong local autonomy, an agent-based platform is the best choice.

In further study, it will be very necessary to consider the following issues:

1. Verification of CPS tasks

In model-driven development (Zhou *et al.*, 2014), a model of the system is first constructed. Then, with the model constructed, the system is analyzed and verified before implementation. However, in platform-based design, CPS tasks are composed of a number of prebuilt structural units. The largest difference between these two approaches is that in platform-based design, the correctness of all prebuilt structural units cannot be guaranteed because the formulation models of the prebuilt structural units are unknown. Further, even if all structural units are known to be correct, the correctness of the CPS tasks composed of them cannot be guaranteed, because in a CPS, functional properties and non-functional properties are equally important. Therefore, how to conduct comprehensive verification of such CPS tasks before their executions becomes a problem to be solved.

2. Support for human-in-the-loop applications

In human-in-the-loop applications, humans are brought into the feedback loops. There are two common types of human-in-the-loop applications. In the first type, humans are treated as sensors. Humans' intents are first inferred by processing the electrophysiological signals of humans. Then, embedded computing systems are employed to transform the intents into robot control signals. Thus, robots are able to take the place of humans to interact with the physical environment (Schirner *et al.*, 2013). In the second type, humans are treated as actuators. By monitoring the system operating state, operators can immediately receive alerts when anomalies occur. On receiving an alert, operators can use their experience and knowledge to adjust the system (Wu and Kaiser,

Table 3 The best platform architectures in different application scenarios

Application scenario	Real-time	Autonomy	Dynamics	Interoperability	The best platform architectures
Vehicle electronic systems	●●●	●	●	●	Component-based platforms
Intelligent manufacturing systems	●	●	●●	●●●	Service-based platforms
Smart grids	●	●●	●●	●●	Agent-based platforms
Swarm of UAVs	●●	●●	●●	●	Agent-based platforms
Intelligent transportation systems	●	●●	●●	●●	Agent-based platforms

The number of black spots indicates the degree of the key features

2012). Note that intent inferences and system monitoring are data- and computation-intensive tasks that need to be handled on time. In addition, in a CPS, the inputs of such tasks are susceptible to disturbance. Therefore, to support human-in-the-loop applications, provision of a timely processing mechanism for data- and computation-intensive tasks is essential.

3. Runtime monitoring of CPS tasks

To enhance resilience and safety of existing platforms, we consider that one possible solution is to add a runtime monitoring mechanism to existing platforms. By online tracking execution traces of CPS tasks, the occurrence of failures and behaviors that may violate safety-critical rules can be realized in a timely manner or even be predicted (Zhao *et al.*, 2010). This could then enable on-time start of recovery procedures. Note that existing runtime monitoring technologies are software system oriented. Their monitoring targets are computation processes, and they cannot monitor the operation of physical processes. This aspect should be dealt with in future research.

References

Acosta, F.J., Weis, F., Bourcier, J., 2014. Towards a Model@Runtime middleware for cyber physical systems. Proc. 9th Workshop on Middleware for Next Generation Internet Computing, Article 6. https://doi.org/10.1145/2676733.2676741

Afanasov, M., Mottola, L., Ghezzi, C., 2014. Towards context-oriented self-adaptation in resource-constrained cyberphysical systems. Proc. IEEE 38th Annual Int. Computers, Software and Applications Conf. Workshops, p.372-377. https://doi.org/10.1109/COMPSACW.2014.64

Ahmadi, H., Abdelzaher, T.F., Gupta, I., 2010. Congestion control for spatio-temporal data in cyber-physical systems. Proc. 1st ACM/IEEE Int. Conf. on Cyber-Physical Systems, p.89-98. https://doi.org/10.1145/1795194.1795207

Aiello, M., Frankova, G., Malfatti, D., 2005. What's in an agreement? An analysis and an extension of WS-agreement. *In*: Benatallah, B., Casati, F., Traverso, P. (Eds.), Service-Oriented Computing-ICSOC 2005. Springer-Verlag Berlin Heidelberg, p.424-436. https://doi.org/10.1007/11596141_32

Al-Safi, Y., Vyatkin, V., 2007. An ontology-based reconfiguration agent for intelligent mechatronic systems. *In*: Mařík, V., Vyatkin, V., Colombo, A.W. (Eds.), Holonic and Multi-agent Systems for Manufacturing. Springer-Verlag Berlin Heidelberg, p.114-126. https://doi.org/10.1007/978-3-540-74481-8_12

Andersson, B., Pereira, N., Tovar, E., 2008. How a cyber-physical system can efficiently obtain a snapshot of physical information even in the presence of sensor faults. Proc. Int. Workshop on Intelligent Solutions in Embedded Systems, p.1-10. https://doi.org/10.1109/WISES.2008.4623298

Asadollah, S.A., Inam, R., Hansson, H., 2015. A survey on testing for cyber physical system. *In*: El-Fakih, K., Barlas, G., Yevtushenko, N. (Eds.), Testing Software and Systems. Springer International Publishing, Cham, Switzerland, p.194-207. https://doi.org/10.1007/978-3-319-25945-1_12

AUTOSAR, 2014. AUTomotive Open System ARchitecture (AUTOSAR). http://www.autosar.org/about/technical-overview/ [Accessed on Nov. 20, 2016].

Axelsson, J., Kobetski, A., 2014. Architectural concepts for federated embedded systems. Proc. European Conf. on Software Architecture Workshops, p.25:1-25:8. https://doi.org/10.1145/2642803.2647716

Barbosa, J., Leitão, P., Adam, E., *et al.*, 2015. Dynamic self-organization in holonic multi-agent manufacturing systems: the ADACOR evolution. *Comput. Ind.*, **66**:99-111. https://doi.org/10.1016/j.compind.2014.10.011

Bellifemine, F., Caire, G., Poggi, A., *et al.*, 2008. JADE: a software framework for developing multi-agent applications: lessons learned. *Inform. Softw. Technol.*, **50**(1): 10-21. https://doi.org/10.1016/j.infsof.2007.10.008

Broy, M., 2013. Cyber-physical systems: concepts, challenges and foundations. https://artemis-ia.eu/publication/download/877-magazine-14.pdf [Accessed on Nov. 20, 2016].

Bruneton, E., Coupaye, T., Leclercq, M., *et al.*, 2006. The FRACTAL component model and its support in Java. *Softw. Pract. Exp.*, **36**(11-12):1257-1284. https://doi.org/10.1002/spe.767

Bures, T., Gerostathopoulos, I., Hnetynka, P., *et al.*, 2013. DEECO: an ensemble-based component system. Proc. 16th ACM Sigsoft Symp. on Component-Based Software Engineering, p.81-90. https://doi.org/10.1145/2465449.2465462

Bures, T., Gerostathopoulos, I., Hnetynka, P., *et al.*, 2014. Gossiping components for cyber-physical systems. *In*: Avgeriou, P., Zdun, U. (Eds.), Software Architecture. Springer International Publishing, Cham, Switzerland, p.250-266. https://doi.org/10.1007/978-3-319-09970-5_23

Buttazzo, G., 2011. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer US, New York, USA, p.1-22.

Chen, B., Cheng, H.H., 2010. A review of the applications of agent technology in traffic and transportation systems. *IEEE Trans. Intell. Transp.*, **11**(2):485-497. https://doi.org/10.1109/TITS.2010.2048313

Chen, B., Cheng, H.H., Palen, J., 2009. Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems. *Transp. Res. C-Emerg.*, **17**(1):1-10.

https://doi.org/10.1016/j.trc.2008.04.003

Cucinotta, T., Mancina, A., Anastasi, G.F., *et al.*, 2009. A real-time service-oriented architecture for industrial automation. *IEEE Trans. Ind. Inform.*, **5**(3):267-277. https://doi.org/10.1109/TII.2009.2027013

Curbera, F., Duftler, M., Khalaf, R., *et al.*, 2002. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Comput.*, **6**(2):86-93. https://doi.org/10.1109/4236.991449

Dillon, T.S., Zhuge, H., Wu, C., *et al.*, 2011. Web-of-things framework for cyber-physical systems. *Concurr. Comp.-Pract. E.*, **23**(9):905-923. https://doi.org/10.1002/cpe.1629

Dobrev, P., Famolari, D., Kurzke, C., *et al.*, 2002. Device and service discovery in home networks with OSGi. *IEEE Commun. Mag.*, **40**(8):86-92. https://doi.org/10.1109/MCOM.2002.1024420

Dubey, A., Karsai, G., Mahadevan, N., 2011. A component model for hard real-time systems: CCM with ARINC-653. *Softw. Pract. Exper.*, **41**(12):1517-1550. https://doi.org/10.1002/spe.1083

Fang, X., Misra, S., Xue, G., *et al.*, 2012. Smart grid—the new and improved power grid: a survey. *IEEE Commun. Surv. Tutor.*, **14**(4):944-980. https://doi.org/10.1109/SURV.2011.101911.00087

Ferreira, P., Doltsinis, S., Anagnostopoulos, A., *et al.*, 2013. A performance evaluation of industrial agents. Proc. 39th Annual Conf. of the IEEE Industrial Electronics Society, p.7404-7409. https://doi.org/10.1109/IECON.2013.6700365

Fouquet, F., Morin, B., Fleurey, F., *et al.*, 2012. A dynamic component model for cyber physical systems. Proc. 15th ACM Sigsoft Symp. on Component Based Software Engineering, p.135-144. https://doi.org/10.1145/2304736.2304759

Giordano, A., Spezzano, G., Vinci, A., 2016. A smart platform for large-scale cyber-physical systems. *In*: Guerrieri, A., Loscri, V., Rovella, A., *et al.* (Eds.), Management of Cyber Physical Objects in the Future Internet of Things. Springer International Publishing, Cham, Switzerland, p.115-134. https://doi.org/10.1007/978-3-319-26869-9_6

Greer, C., Wollman, D.A., Prochaska, D.E., *et al.*, 2014. NIST framework and roadmap for smart grid interoperability standards, release 3.0. Specical Publication 1108r3, US National Institute of Standards and Technology, Gaithersburg, USA. https://doi.org/10.6028/NIST.SP.1108r3

GRID4EU, 2012. Specification and Requirements. http://grid4eu.blob.core.windows.net/media-prod/6578/Grid4EU_dD1.1_Demo_1_V1.0.pdf [Accessed on Nov. 20, 2016].

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

Haque, S.A., Aziz, S.M., Rahman, M., 2014. Review of cyber-physical system in healthcare. *Int. J. Distrib. Sens. Netw.*, **2014**:217415:1-217415:20. https://doi.org/10.1155/2014/217415

Hellbruck, H., Teubler, T., Fischer, S., 2013. Name-centric service architecture for cyber-physical systems. Proc. IEEE 6th Int. Conf. on Service-Oriented Computing and Applications, p.77-82. https://doi.org/10.1109/SOCA.2013.63

Hsieh, F.S., 2010. Design of reconfiguration mechanism for holonic manufacturing systems based on formal models. *Eng. Appl. Artif. Intel.*, **23**(7):1187-1199. https://doi.org/10.1016/j.engappai.2010.05.008

Hu, F., Lu, Y., Vasilakos, A.V., *et al.*, 2016. Robust cyber–physical systems: concept, models, and implementation. *Fut. Gener. Comp. Syst.*, **56**:449-475. https://doi.org/10.1016/j.future.2015.06.006

Huang, J., Bastani, F., Yen, I.L., *et al.*, 2009a. Extending service model to build an effective service composition framework for cyber-physical systems. Proc. IEEE Int. Conf. on Service-Oriented Computing and Applications, p.1-8. https://doi.org/10.1109/SOCA.2009.5410453

Huang, J., Bastani, F., Yen, I.L., *et al.*, 2009b. Toward a smart cyber-physical space: a context-sensitive resource-explicit service model. Proc. 33rd Annual IEEE Int. Computer Software and Applications Conf., p.122-127. https://doi.org/10.1109/COMPSAC.2009.125

Huang, J., Bastani, F.B., Yen, I.L., *et al.*, 2010. A framework for efficient service composition in cyber-physical systems. Proc. 5th IEEE Int. Symp. on Service Oriented System Engineering, p.291-298. https://doi.org/10.1109/SOSE.2010.46

Inam, R., Carlson, J., Sjödin, M., *et al.*, 2014. Predictable integration and reuse of executable real-time components. *J. Syst. Softw.*, **91**:147-162. https://doi.org/10.1016/j.jss.2013.12.040

Jammes, F., Mensch, A., Smit, H., 2005. Service-oriented device communications using the devices profile for web services. Proc. 3rd Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing, p.1-8. https://doi.org/10.1145/1101480.1101496

Jia, D., Lu, K., Wang, J., *et al.*, 2015. A survey on platoon-based vehicular cyber-physical systems. *IEEE Commun. Surv. Tutor.*, **18**(1):263-284. https://doi.org/10.1109/COMST.2015.2410831

Jin, X., Chun, S., Jung, J., *et al.*, 2014. IoT service selection based on physical service model and absolute dominance relationship. Proc. IEEE 7th Int. Conf. on Service-Oriented Computing and Applications, p.65-72. https://doi.org/10.1109/SOCA.2014.24

Karnouskos, S., Bangemann, T., Diedrich, C., 2009. Integration of legacy devices in the future SOA-based factory. *IFAC Proc. Vol.*, **42**(4):2113-2118. https://doi.org/10.3182/20090603-3-RU-2001.0487

Karnouskos, S., Colombo, A.W., Jammes, F., *et al.*, 2010. Towards an architecture for service-oriented process monitoring and control. Proc. IECON 36th Annual Conf. on IEEE Industrial Electronics Society, p.1385-1391. https://doi.org/10.1109/IECON.2010.5675482

Khaitan, S.K., McCalley, J.D., 2015. Design techniques and applications of cyberphysical systems: a survey. *IEEE Syst. J.*, **9**(2):350-365. https://doi.org/10.1109/JSYST.2014.2322503

Kim, M., Stehr, M.O., Kim, J., *et al.*, 2013. An application framework for loosely coupled networked cyber-physical systems. Proc. IEEE/IFIP 8th Int. Conf. on Embedded and Ubiquitous Computing, p.144-153. https://doi.org/10.1109/EUC.2010.30

Lee, E.A., 2008. Cyber physical systems: design challenges. Proc. 11th IEEE Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing, p.363-369. https://doi.org/10.1109/ISORC.2008.25

Leitão, P., 2013. Towards self-organized service-oriented multi-agent systems. *In*: Borangiu, T., Thomas, A., Trentesaux, D. (Eds.), Service Orientation in Holonic and Multi Agent Manufacturing and Robotics. Springer Berlin Heidelberg, p.41-56. https://doi.org/10.1007/978-3-642-35852-4_3

Leitão, P., Restivo, F., 2006. ADACOR: a holonic architecture for agile and adaptive manufacturing control. *Comput. Ind.*, **57**(2):121-130. https://doi.org/10.1016/j.compind.2005.05.005

Leitão, P., Marik, V., Vrba, P., 2013. Past, present, and future of industrial agent applications. *IEEE Trans. Ind. Inform.*, **9**(4):2360-2372. https://doi.org/10.1109/TII.2012.2222034

Lepuschitz, W., Vallee, M., Merdan, M., *et al.*, 2009. Integration of a heterogeneous low level control in a multi-agent system for the manufacturing domain. Proc. 14th IEEE Int. Conf. on Emerging Technologies Factory Automation, p.574-581. https://doi.org/10.1109/ETFA.2009.5347061

Levendovszky, T., Dubey, A., Otte, W.R., *et al.*, 2014. Distributed real-time managed systems: a model-driven distributed secure information architecture platform for managed embedded systems. *IEEE Softw.*, **31**(2):62-69. https://doi.org/10.1109/MS.2013.143

Li, F., Xu, J., Yu, G., 2012. A survey on event processing for CPS. *In*: Wang, R., Xiao, F. (Eds.), Advances in Wireless Sensor Networks. Springer Berlin Heidelberg, p.157-166. https://doi.org/10.1007/978-3-642-36252-1_15

Li, Q., Qin, W., Han, B., *et al.*, 2011. A case study on rest-style architecture for cyber-physical systems: restful smart gateway. *Comput. Sci. Inform. Syst.*, **8**(4):1317-1329. https://doi.org/10.2298/CSIS110310062L

Li, R.F., Xie, Y., Li, R., *et al.*, 2012. Survey of cyber-physical systems. *J. Comput. Res. Dev.*, **49**(6):1149-1161 (in Chinese).

Lin, J., Sedigh, S., Miller, A., 2011. A semantic agent framework for cyber-physical systems. *In*: Elçi, A., Koné, M.T., Orgun, M.A. (Eds.), Semantic Agent Systems. Springer Berlin Heidelberg, p.189-213. https://doi.org/10.1007/978-3-642-18308-9_9

Lopez, P., Medina, J.L., Drake, J.M., 2006. Real-time modelling of distributed component-based applications. Proc. 32nd EUROMICRO Conf. on Software Engineering and Advanced Applications, p.92-99. https://doi.org/10.1109/EUROMICRO.2006.52

Macana, C.A., Quijano, N., Mojica-Nava, E., 2011. A survey on cyber physical energy systems and their applications on smart grids. Proc. IEEE PES Conf. on Innovative Smart Grid Technologies, p.1-7. https://doi.org/10.1109/ISGT-LA.2011.6083194

Martin, D., Paolucci, M., McIlraith, S., *et al.*, 2005. Bringing semantics to web services: the OWL-S approach. *In*: Cardoso, J., Sheth, A. (Eds.), Semantic Web Services and Web Process Composition. Springer-Verlag Berlin Heidelberg, p.26-42. https://doi.org/10.1007/978-3-540-30581-1_4

Martínez, P.L., Cuevas, C., Drake, J.M., 2010. RT-D&C: deployment specification of real-time component-based applications. Proc. 36th EUROMICRO Conf. on Software Engineering and Advanced Applications, p.147-155. https://doi.org/10.1109/SEAA.2010.22

Martínez, P.L., Barros, L., Drake, J.M., 2013. Design of component-based real-time applications. *J. Syst. Softw.*, **86**(2):449-467. https://doi.org/10.1016/j.jss.2012.09.036

Mendes, J.M., Leitão, P., Restivo, F., *et al.*, 2009. Service-oriented agents for collaborative industrial automation and production systems. *In*: Mařík, V., Strasser, T., Zoitl, A. (Eds.), Holonic and Multi-agent Systems for Manufacturing. Springer-Verlag Berlin Heidelberg, p.13-24. https://doi.org/10.1007/978-3-642-03668-2_2

Mendes, J.M., Leitão, P., Restivo, F., *et al.*, 2010. Composition of Petri nets models in service-oriented industrial automation. Proc. 8th IEEE Int. Conf. on Industrial Informatics, p.578-583. https://doi.org/10.1109/INDIN.2010.5549677

Microsoft, 2015. Smart Energy Reference Architecture Version 2.0. https://msenterprise.global.ssl.fastly.net/wordpress/Reference_Architecture_pdf_whitepaper_2.pdf [Accessed on Nov. 20, 2016].

Miller, B.A., Nixon, T., Tai, C., *et al.*, 2001. Home networking with universal plug and play. *IEEE Commun. Mag.*, **39**(12):104-109. https://doi.org/10.1109/35.968819

Monostori, L., Kadar, B., Bauernhansl, T., *et al.*, 2016. Cyber-physical systems in manufacturing. *CIRP Ann. Manuf. Techn.*, **65**(2):621-641. https://doi.org/10.1016/j.cirp.2016.06.005

Morin, B., Barais, O., Nain, G., *et al.*, 2009. Taming dynamically adaptive systems using models and aspects. Proc. 31st Int. Conf. on Software Engineering, p.122-132. https://doi.org/10.1109/ICSE.2009.5070514

Muccini, H., Sharaf, M., Weyns, D., 2016. Self-adaptation for cyber-physical systems: a systematic literature review. Proc. 11th Int. Workshop on Software Engineering for Adaptive and Self-Managing Systems, p.75-81.

https://doi.org/10.1145/2897053.2897069

Ni, Z., Kobetski, A., Axelsson, J., 2014. Design and implementation of a dynamic component model for federated AUTOSAR systems. Proc. 51st Annual Design Automation Conf., p.94:1-94:6. https://doi.org/10.1145/2593069.2593121

Nikam, S., Ingle, R., 2014. Resource provisioning algorithms for service composition in Cyber Physical Systems. Proc. Int. Conf. on Advances in Computing, Communications and Informatics, p.2797-2802. https://doi.org/10.1109/ICACCI.2014.6968650

Obermaisser, R., Huber, B., 2009. The GENESYS architecture: a conceptual model for component-based distributed real-time systems. *In*: Lee, S., Narasimhan, P. (Eds.), Software Technologies for Embedded and Ubiquitous Systems. Springer-Verlag Berlin Heidelberg, p.296-307. https://doi.org/10.1007/978-3-642-10265-3_27

Otte, W.R., Dubey, A., Karsai, G., 2014. A resilient and secure software platform and architecture for distributed spacecraft. SPIE, **9085**:90850J. https://doi.org/10.1117/12.2054055

Pajic, M., Chernoguzov, A., Mangharam, R., 2012. Robust architectures for embedded wireless network control and actuation. *ACM Trans. Embed. Comput. Syst.*, **11**(4):82. https://doi.org/10.1145/2362336.2362349

Papazoglou, M.P., Heuvel, W.J., 2007. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, **16**(3):389-415. https://doi.org/10.1007/s00778-007-0044-3

Park, S.O., Do, T.H., Jeong, Y.S., *et al.*, 2013. A dynamic control middleware for cyber physical systems on an IPv6-based global network. *Int. J. Commun. Syst.*, **26**(6): 690-704. https://doi.org/10.1002/dac.1382

Parvin, S., Hussain, F.K., Hussain, O.K., *et al.*, 2013. Multi-cyber framework for availability enhancement of cyber physical systems. *Computing*, **95**(10-11):927-948. https://doi.org/10.1007/s00607-012-0227-7

Pradhan, S., Otte, W.R., Dubey, A., *et al.*, 2014. Towards a resilient deployment and configuration infrastructure for fractionated spacecraft. *ACM SIGBED Rev.*, **10**(4):29-32. https://doi.org/10.1145/2583687.2583694

Puttonen, J., Lobov, A., Lastra, J.L.M., 2008. An application of BPEL for service orchestration in an industrial environment. Proc. 13th IEEE Int. Conf. on Emerging Technologies and Factory Automation, p.530-537. https://doi.org/10.1109/ETFA.2008.4638450

Rajkumar, R., Lee, I., Sha, L., *et al.*, 2010. Cyber-physical systems: the next computing revolution. Proc. 47th ACM/IEEE Design Automation Conf., p.731-736. https://doi.org/10.1145/1837274.1837461

Schirner, G., Erdogmus, D., Chowdhury, K., *et al.*, 2013. The future of human-in-the-loop cyber-physical systems. *Computer*, **46**(1):36-45. https://doi.org/10.1109/MC.2013.31

Seow, K.T., Dang, N.H., Lee, D.H., 2010. A collaborative multiagent taxi-dispatch system. *IEEE Trans. Autom. Sci. Eng.*, **7**(3):607-616. https://doi.org/10.1109/TASE.2009.2028577

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

Shi, J., Wan, J., Yan, H., *et al.*, 2011. A survey of cyber-physical systems. Proc. Int. Conf. on Wireless Communications and Signal Processing, p.1-6. https://doi.org/10.1109/WCSP.2011.6096958

SMB Smart Grid Strategic Group, 2010. IEC Smart Grid Standardization Roadmap. http://www.iec.ch/smartgrid/downloads/sg3_roadmap.pdf [Accessed on Nov. 20, 2016].

Soulier, P., Li, D., Williams, J.R., 2015. A survey of language-based approaches to Cyber-Physical and embedded system development. *Tsinghua Sci. Technol.*, **20**(2):130-141. https://doi.org/10.1109/TST.2015.7085626

Srbljic, S., Skvorc, D., Popovic, M., 2012. Programming languages for end-user personalization of cyber-physical systems. *Automatika*, **53**(3):294-310. https://doi.org/10.7305/automatika.53-3.84

Stojmenovic, I., 2014. Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems. *IEEE IOT J.*, **1**(2):122-128. https://doi.org/10.1109/JIOT.2014.2311693

Tan, Y., Vuran, M.C., 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

Valls, M.G., Lopez, I.R., Villar, L.F., 2013. iLand: an enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Trans. Ind. Inform.*, **9**(1):228-236. https://doi.org/10.1109/TII.2012.2198662

Vegh, L., Miclea, L., 2016. Secure and efficient communication in cyber-physical systems through cryptography and complex event processing. Proc. Int. Conf. on Communications, p.273-276. https://doi.org/10.1109/ICComm.2016.7528290

Vicaire, P.A., Xie, Z., Hoque, E., *et al.*, 2010. Physicalnet: a generic framework for managing and programming across pervasive computing networks. Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symp., p.269-278. https://doi.org/10.1109/RTAS.2010.17

Vicaire, P.A., Hoque, E., Xie, Z., *et al.*, 2012. Bundle: a group-based programming abstraction for cyber-physical systems. *IEEE Trans. Ind. Inform.*, **8**(2):379-392. https://doi.org/10.1109/TII.2011.2166772

Vrba, P., Radakovič, M., Obitko, M., *et al.*, 2011a. Semantic technologies: latest advances in agent-based manufacturing control systems. *Int. J. Prod. Res.*, **49**(5):1483-1496. https://doi.org/10.1080/00207543.2010.518746

Vrba, P., Tichý, P., Mařík, V., *et al.*, 2011b. Rockwell automation's holonic and multiagent control systems compendium. *IEEE Trans. Syst. Man Cybern. C*, **41**(1):14-30. https://doi.org/10.1109/TSMCC.2010.2055852

Vrba, P., Mařík, V., Siano, P., *et al.*, 2014. A review of agent and service-oriented concepts applied to intelligent energy systems. *IEEE Trans. Ind. Inform.*, **10**(3):1890-1903. https://doi.org/10.1109/TII.2014.2326411

Wan, J., Yan, H., Suo, H., *et al.*, 2011. Advances in cyber-physical systems research. *KSII Trans. Internet Inform.*, **5**(11):1891-1908. https://doi.org/10.3837/tiis.2011.11.001

Wan, K., Alagar, V., Dong, Y., 2014. Specifying resource-centric services in cyber physical systems. *In*: Yang, G.C., Ao, S.I., Huang, X., *et al.* (Eds.), Transactions on Engineering Technologies. Springer Netherlands, Dordrecht, the Netherland, p.83-97. https://doi.org/10.1007/978-94-007-7684-5_7

Wang, F.Y., 2008. Toward a revolution in transportation operations: AI for complex systems. *IEEE Intell. Syst.*, **23**(6): 8-13. https://doi.org/10.1109/MIS.2008.112

Wang, T., Cheng, L., Zheng, K., 2012. Automatic and effective service provision with context-aware service composition mechanism in cyber-physical systems. *Adv. Inform. Sci. Serv. Sci.*, **4**(11):151-160. https://doi.org/10.4156/AISS.vol4.issue11.18

Wang, Z.J., Xie, L.L., 2011. Cyber-physical systems: a survey. *Acta Autom. Sin.*, **37**(10):1157-1166 (in Chinese).

Woo, H., Yi, J., Browne, J.C., *et al.*, 2008. Design and development methodology for resilient cyber-physical systems. Proc. 28th Int. Conf. on Distributed Computing Systems Workshops, p.525-528. https://doi.org/10.1109/ICDCS.Workshops.2008.62

Wu, G., Sun, J., Chen, J., 2016. A survey on the security of cyber-physical systems. *J. Contr. Theory Technol.*, **14**(1):2-10. https://doi.org/10.1007/s11768-016-5123-9

Wu, L., Kaiser, G., 2012. An autonomic reliability improvement system for cyber-physical systems. Proc. IEEE 14th Int. Symp. on High-Assurance Systems Engineering, p.56-61. https://doi.org/10.1109/HASE.2012.33

Xiao, K., Ren, S., Kwiat, K., 2008. Retrofitting cyber physical systems for survivability through external coordination. Proc. 41st Annual Hawaii Int. Conf. on System Sciences, p.465-465. https://doi.org/10.1109/HICSS.2008.377

Zhao, C., Dong, W., Qi, Z., 2010. Active monitoring for control systems under anticipatory semantics. Proc. 10th Int. Conf. on Quality Software, p.318-325. https://doi.org/10.1109/QSIC.2010.82

Zhou, X.S., Yang, Y.L., Yang, G., 2014. Modeling methods for dynamic behaviors of cyber-physical system. *Chin. J. Comp.* **37**(6):1411-1423 (in Chinese).

Zhu, W., Zhou, G., Yen, I.L., *et al.*, 2015. A PT-SOA model for CPS/IoT services. Proc. IEEE Int. Conf. on Web Services, p.647-654. https://doi.org/10.1109/ICWS.2015.91