IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 30, 2018, accepted September 28, 2018, date of publication October 8, 2018, date of current version October 29, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2873751

CPS-Agent Oriented Construction and Implementation For Cyber Physical Systems

YUJIAO HU AND XINGSHE ZHOU, (Member, IEEE)
School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China
Corresponding author: Yujiao Hu (yujiao_hu@mail.nwpu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61751208 and Grant 61502394.

ABSTRACT Cyber-physical systems (CPSs) have attracted many researchers in areas as diverse as aerospace, manufacturing, transportation, and so on. However, modeling methodologies and tools for autonomous objects in CPS are still lacking. In this paper, the *CPS-Agent* is proposed to model objects with consideration of temporal-spatial traits and interaction with physical environment. It is formulated by a five-tuple. Furthermore, considering that no universal methodology of coordination strategy formulation could be used to guide researchers, we present a role-based strategy formulation to make work patterns of *CPS-Agents* more clear. In terms of network communication among *CPS-Agents*, a set of communicative primitives is tailored based on the FIPA-ACL specification. Afterward, to guide engineers in designing systems according to their application requirements in the area of CPS, we design templates and a novel visual support tool for generating C++ files automatically corresponding to customized CPS-Agents, coordination strategies, and coordination groups. Finally, the complete development process based on our methodologies and tool is illustrated by an instance of a car team.

INDEX TERMS Cyber physical system, modeling, support tool, CPS-agent, code generation.

I. INTRODUCTION

Cyber physical systems (CPS) integrate computing power, communication networks and control systems. Example are autonomous cars, rescue robots, unmanned aerial vehicles, etc. Objects in CPS are composed of sensors, actuators, and computing kernels. However such composition make objects have inherent constraints, for example, sensing distances, accuracies of actuators, etc. Moreover, objects are active in a dynamic physical environment, therefore conditions about motion of objects are usually related to time and space. Besides these, sometimes objects are expected to coordinate for cooperative tasks. When deeply researching such multi-object systems, engineers need to model objects with consideration of all their features firstly, and build tools to support specification, design, implementation and testing/debugging for CPS.

There are three most popular modeling and tool development methodologies in the software discipline: component-based development (CBD) [2], service-oriented architecture (SOA) [4], and agent-based model (ABM) [1]. CBD is a reuse-based approach to compose loosely coupled independent components into systems. Components are usually software packages or web sources. It is hard to make physical processes into components. SOA is a kind of software design where services are provided to the other components by application components, through a communication protocol over a network. The service is a black box for its customers, but it has more knowledge than customers. The relationship between services and clients is unequal, which is not expected for development of objects in CPS. ABM is used to simulate the actions and interactions of autonomous agents. Theoretically, the agent could be a model of any individual with autonomous feature. The agent is not a software conception, even though it gets a broader range of researches and applications in the software discipline. The past decades have witnessed contributions about the agent framework development in the software discipline, including *JADE* (Java Agent Development Framework), *JIAC V* (Java-based Intelligent Agent Componentware), etc. The development platform for ABM is usually *Eclipse*, and the development programming language is *JAVA*.

However, while software agent development technologies being applied into CPS development, some problems cannot be ignored. Firstly, software agent technologies depend on powerful computing capability, and they focus on discrete processes, whereas CPS development engineers should

care more about continuous dynamic physical processes. Secondly, events in software agent design are more like interrupts in operating systems, whereas events are usually tagged with a time label and a space label during development of CPS. In this paper, we introduce a new model *CPS-Agent*, which keeps the advantages of autonomous and social features of the agent, and complement disadvantages of agents in modeling physical attributes and processes. The CPS-Agent could be classified into three types according to practical application requirements, including *Reactive CPS-Agent*, *Decision-making CPS-Agent* and *Hybrid CPS-Agent*. In addition, common software agent development platform *Eclipse* and programming language *JAVA* are not suitable for development of objects in CPS, because most devices in CPS are embedded, they support C/C++ for developing software. Due to the non-applicabilities of platform and programming language, we further design a visual support tool where engineers are able to customize CPS-Agents, coordination strategies and coordination groups. The tool will generate XML files and C++ files automatically to assist secondary development for engineers. Moreover, the tool supports linking, compiling and running programs. As our best knowledge, no literature has proposed a tool like ours.

The sociality is one of the most important features for objects in CPS, too. In other words, while performing cooperative tasks, objects will follow a coordination strategy to realize tasks and exchange information by network communication. As for coordination strategies, no universal methodology of coordination strategy formulation could be used to guide researchers. But an interesting phenomenon is found that objects in CPS usually realize different sub-tasks and their behavior interaction follows certain patterns. Therefore we propose a role-based strategy formulation to make the patterns more clear and simplify complexity of task assignments. In terms of network communication, there are two more mature agent communication specifications based on message transmission, *KQML* (Knowledge Query and Manipulation Language) and *FIPA-ACL* (Agent Communication Languages established by Foundation for Intelligent Physical Agents). In this paper, based on the more promising *FIPA-ACL* specification, we design a set of communication primitives and communication message format to support CPS-Agents communication.

The contributions of this paper lie in:

1) The *CPS-Agent* is introduced to model objects in CPS.

2) A role-based strategy formulation methodology is proposed to direct researchers to design coordination strategies.

3) To realize inter-operability and mutual understanding among CPS-Agents through network communication, a set of communication primitives based on *FIPA-ACL* is tailored.

4) Novel C++ code templates for CPS-Agent construction and an innovative visual support tool are designed.

5) An instance of a car team is used to specifically illustrate development processes on the tool.

The paper is organized as follows. In the following section, a literature review is conducted. Afterwards, in section III, we give definition of the CPS-Agent and formulate it by a five-tuple. Then we classify the CPS-Agent into three categories and given architecture descriptions for them. Section IV presents coordination support mechanisms about coordination strategies, coordination groups and coordination communication primitives. In section V, the design principles and development processes of the support tool are introduced in detail. Section VI illustrates development processes on the tool through an instance of a car team. Finally, section VII gathers our conclusions and open issues for future researches.

II. RELATED WORK

Researchers have presented some contributions in the domain of CPS in recent years. Some representative literatures include an experience report of developing applications on CPS [21], an automatic transformation from MUML (Modelica MechatronicUML) to Modelica [20], a MechatronicUML tool suite to support MechatronicUML modeling and analyze tasks [6], a domain-specific model [8], a security assessment tool [22], a multi-discipline development process of CPS [11], security protection approaches for underlying hardware in CPS through obfuscation-based methodologies and loop-based high-level transformations [23]–[25]. Papers [6], [8], [11], [20], and [21] are a series of work researched in *University of Paderborn, Germany*. They use *MechatronicUML* to show their models and software of applications in papers. *MechatronicUML* is a tool that *Eclipse* provided to visualize CPS processes. Specifically, the paper [21] talks about practical experience of developing cooperative CPS of the research group, and it summarizes challenges in CPS development, including a) application architectures getting more complex, b) the controller having to consider information from other systems and their control strategies, c) interaction by communication based on messages becoming more dependent on physical dynamics, d) engineers having to consider different system parts while developing the system, e) a discipline-spanning collaboration being needed. Papers [6], [8], and [20] use a same scenario of overtaking between two cars to describe their software architecture. A development process which integrate mechanical, electrical and software parts is shown in [11]. The paper uses a *RailCab Convoy* example to illustrate opinion. However, all these results are component-based, and they usually use JAVA on Eclipse. Moreover, they provide much research thought, but we couldn't find a universal available software for developing CPS applications except their examples.

As our best knowledge, there is no contribution designing a agent-based platform for CPS development. However, issues about software agent frameworks were very hot around the year 2000. Surveys [12], [28] give researchers some methodologies for agent design. The paper [26] described a framework to help evaluation the appropriate development methodologies of agent-based systems with respect to platforms. The original concept of a graphical tool for monitoring

and demonstrating distributed agents was built in [27]. The work [14] tended to build a plug-in for *Eclipse*, aiming to build an independent integrated development environment, and realize agent development process, from specification, design, implementation to testing/debugging. In addition, a survey of agent development platforms is given in [13]. It compare and evaluate 24 platforms in terms of *properties*, *usability*, *operating ability*, *pragmatics*, *security management*. Because of lack of space, we just give a detail discussion about the most popular platform, *JADE*. *JADE* is a fantastic platform fully implemented in JAVA, it supports visual graphical form, distributed agents development, agent communication using RMI (Remote Method Invocation) to transmit messages that are encapsulated as JAVA objects. Some researches are going on based on *JADE*. The work [5] tried to design a novel agent-oriented communication language on *JADE*. The paper [29] wanted to explore the security challenge based on the JADE framework. There are still many contributions we won't list. Even though they have gotten great results, they are not suitable when being applied to development CPS for following reasons. Firstly, the JAVA is the preferred programming language in software agent development, but most embedded devices support C/C++ rather than JAVA. Secondly, they didn't design an available tool for development CPS, especially, they didn't consider the important issue of physical constraints in CPS. Thirdly, even if *JADE* provide distributed agent development methodologies, the requirement of communication between agents by RMI is not practical, due to objects in CPS communicate with others by network protocols, such as TCP/IP, CAN, etc. Finally, messages cannot be encapsulated as JAVA objects and then be sent in CPS, they have to convert to transfer formats that are required by the network protocols.

In addition, with consideration of cooperative tasks in CPS, there are not universal methodologies of coordination strategy formulation to guide researchers. However, some effective strategies are given in some papers. Researchers in different disciplines have their own opinions. In the domain of control, [19] built mathematical model and functions to realize local and global coverage strategies, in the scenario of distributed dynamic coverage and avoidance control for multiple agents with anisotropic sensing. The work [18] proposed a method based on a family of two-dimensional analytical expression to solve distributed motion planning of multiple agents in different classes. The paper [17] gave a set of Lyapunov-like barrier functions and velocity control functions for agents to guide them converge to pre-defined positions while keeping connection with leader and avoiding collision. In the computing domain, a review [9] of popular motion planning algorithms and outstanding research groups is given. Path elongation algorithm amidst obstacles and two task assignment algorithms are given in [10] to realize prioritized targets and motion planning for UAVs. Some results combined computing and control methodologies. For example, [15] and [16] proposed two-level coordination structure, the high-level planning used a prioritized A* algorithm to compute waypoints and the low-level trajectory generation adopted mathematical model based on Lyapunov barrier functions.

As for communication specification, *KQML* and *FIPA-ACL* are the more mature specifications where the *communicative acts set* play the most important role. As for *KQML*, communicative acts have semantic meaning, but they are not specified strictly. *KQML* defines more than 41 communicative acts, and the amount could still increase. Hence, because of uncertain amount and relax specification, the inter-operation based on *KQML* between agents doesn't come true. *FIPA-ACL* requires that when an agent wants to transmit one message with a communicative act, then it must implement the acts corresponding to the communicative act. *FIPA-ACL* have 22 communicative acts in specification [7], and each communicative act has strict description and formal model. They could be divided into five types, *information or data transmission, information request, negotiation, action execution* and *error handling*. Considerable and strict specification make *FIPA-ACL* be more promising.

III. MODELING AND PERSONALIZATION CATEGORIES

A. CPS-AGENT MODEL

A CPS-Agent refers to an autonomous executor. It integrates computing, communication and control (3C) capabilities, and it could adapt to changes of the physical environment and interact with physical processes continuously.

While the CPS-Agent being applied in CPS development, it will model an autonomous object. The object could be either software or embedded hardware, and it can usually realize some tasks independently. However, we won't model all such objects using the CPS-Agent. For example, a temperature sensor won't be modeled as a CPS-Agent here. We think it is automatical rather than autonomous, because its functions and control rules are too simple. But the CPS-Agent model is qualified for modeling such objects. In this paper, we consider a more complex object, such as a car, a driverless vehicle, etc. These objects are more autonomous than the temperature sensor. At the same time, their multi-layer architectures, well-designed software systems and hundreds of physical components make them have powerful 3C capabilities and promote them to become more autonomous while interacting with physical environment and coordinating with other individuals.

After determining the modeled object, we need to extract some important information, including *static physical information, dynamic spatio-temporal information*, shown in Fig. 1. The static physical information depends on resource configuration of the autonomous object, but the CPS-Agent is more concerned with the impact of resource configuration on performance than with the parameters of resource configuration. The dynamic information will be catched at running time, and it is usually connected with time and space. Note that the spatio-temporal feature is an unique attribute

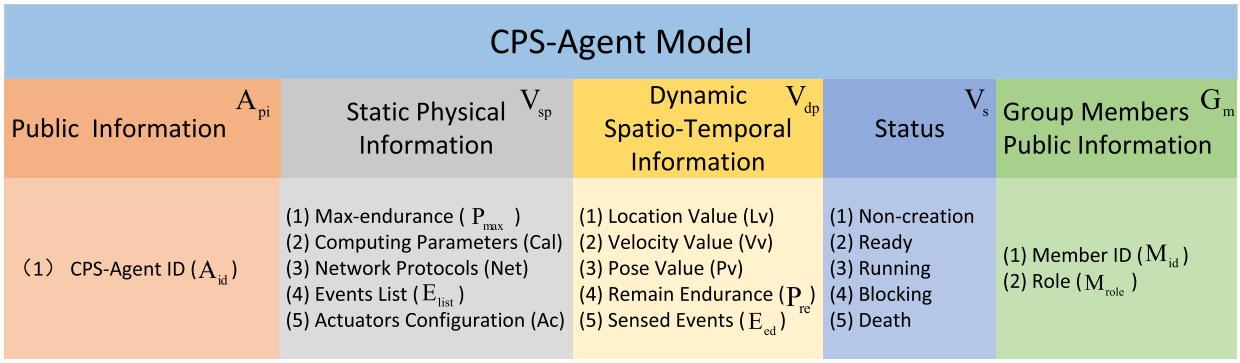

FIGURE 1. Summary version of the CPS-Agent formulation.

for CPS development, because a) working duration of the modeled object is usually limited by fuel storage or battery capacity, and effective activity scope is limited by controllers; b) the runtime condition of the object is always bind with time and space; c) actions and decisions are tagged with time and space. The key information extraction makes the view of application development engineers change from an advanced embedded device with many complex components to a CPS-Agent (part of). Following that, engineers could research the issue from a CPS-Agent perspective. Engineers in other disciplines could get opportunities to join these hot issues.

Here a five-tuple is given to formulate the CPS-Agent, as in (1). A summary version of formulation is presented in Fig. 1.

$$CPS - Agent := \langle A_{pi}, V_{sp}, V_{dp}, V_s, G_m \rangle \tag{1}$$

A_{pi} represents public information of the CPS-Agent, and it has one element, i.e. CPS-Agent ID A_{id}. A_{id} is the only identifier to distinguish CPS-Agents apparently, because it is the only public information that the CPS-Agent publish to all CPS-Agents. It is expressed by combinations of string and number.

V_{sp} is a set, and it records static physical information of the modeled object. The information mainly includes performance values and static spatio-temporal information, i.e. max working endurance, effective activity scope, etc. V_{sp} is described from five aspects, as in (2). In (2), P_{max} expresses the max endurance of CPS-Agent. *Cal* records parameters related with computation, such as computation velocity. *Net* represents network protocols or customized communication methods that the CPS-Agent supports. E_{list} records all events that the CPS-Agent could perceive. *Ac* records configuration information of actuators and static spatio-temporal attributes, for example, fixed acceleration, maximum velocity.

$$V_{sp} := \langle P_{max}, Cal, Net, E_{list}, Ac \rangle \tag{2}$$

V_{dp} in (1) records dynamic information of a CPS-Agent from five aspects, too. These information is usually related with time and space, as shown in (3). In (3), *Lv* samples the location of the CPS-Agent at the current time, and the coordinate system is selected according to the application requirements. *Vv* shows traveling velocity at the current time. *Pv* represent all pose angles at the current time, for example, a UAV has pitching, yawing and roll angles; a driverless vehicle has only orientation angle. P_{re} reminds engineers the remaining working endurance. E_{ed} saves the list of perceived but unprocessed events so far.

$$V_{dp} := \langle Lv, Vv, Pv, P_{re}, E_{ed} \rangle \tag{3}$$

V_s in (1) represents status of a CPS-Agent in time. This item is used to help manage and schedule CPS-Agents in CPS. It is divided into five statuses, as in (4).

$$V_s := \{Non - creation, Running, Ready, Blocking, Death\} \tag{4}$$

Description for statuses in (4): *Non-creation*: A CPS-Agent at this status is not created, or it has been created but isn't loaded into the executable CPS-Agent queue. *Ready*: A CPS-Agent is ready to execute, but needs to be triggered by sensitive events. *Running*: A CPS-Agent is running to achieve its task goal. *Blocking*: A CPS-Agent will not run until something it is waiting for happens. *Death*: A CPS-Agent has released the resources it ever possessed and never runs again. The detailed transition procedures are showed in Fig. 2. But because of limitation of pages, we won't explain in detail.

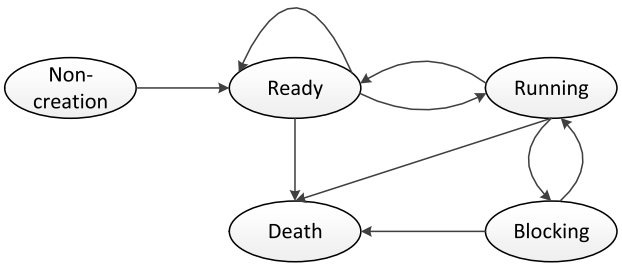

FIGURE 2. Statuses transition graph of the CPS-Agent.

G_m in (1) records team members, i.e. multiple CPS-Agents are possible to form a group for cooperative tasks, then each

CPS-Agent will have partners. According to coordination strategy that the group adopt to, partners will play different roles. G_m of the CPS-Agent will record the public information of partners, including *ID* and *role*. G_m could be formulated by (5). M_{id} is used to save A_{id} of partners, M_{role} saves roles of partners.

$$G_m := \langle M_{id}, M_{role} \rangle \tag{5}$$

B. PERSONALIZATION CATEGORIES AND ARCHITECTURE DESIGNS FOR THE CPS-AGENT

We propose three architectures for the CPS-Agent construction, i.e. reactive CPS-Agent, decision-making CPS-Agent and hybrid CPS-Agent respectively.

1) REACTIVE CPS-AGENT

The reactive CPS-Agent take actions according to action library once receiving a sensitive event or signal.

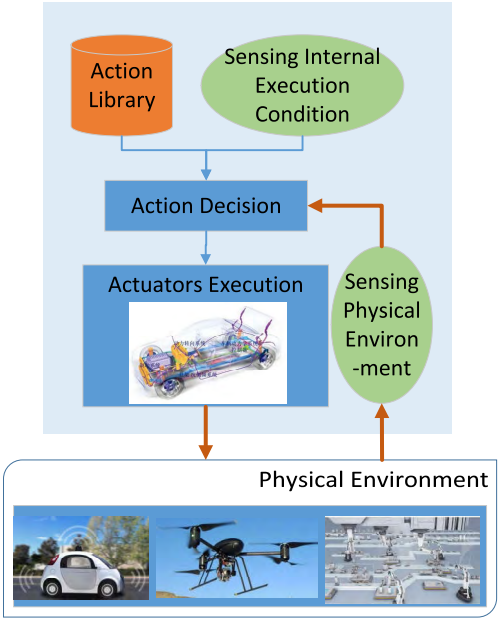

FIGURE 3. The architecture of the reactive CPS-Agent. The orange lines represent the interaction between the CPS-Agent and environment.

The reactive CPS-Agent is a class of simple CPS-Agents with fast response. An example is a service robot that moves around a black rounded rectangle. The robot will stop immediately when sensors perceive obstructions. When the robot perceive obstructions leaving, it will move again. Overall, the solution process of the reactive CPS-Agent is similar to neural arc reflex behavior. The architecture of the reactive CPS-Agent is shown in the Fig. 3. The orange lines represent the interaction between the CPS-Agent and environment. The module of *Sensing Internal Execution Condition* is responsible for getting internal parameters and status from V_{sp}, V_{dp}, V_s and G_m. The module of *Sensing Physical Environment* will get information from sensors that monitor environment changes. The *Action Library* is related to domain and configuration of the object that is modeled by the CPS-Agent. It could evolve with the development of technologies. The *Action Decision* module decides which action should be taken, then transfer the action command to *Actuators Execution*. The *Actuators Execution* will firstly covert the command to understanding parameters for kinetics systems, and affect *Physical Environment*.

2) DECISION-MAKING CPS-AGENT

The decision-making CPS-Agent is a kind of intelligent CPS-Agents. It could infer, plan and make decisions, according to existing view and knowledge.

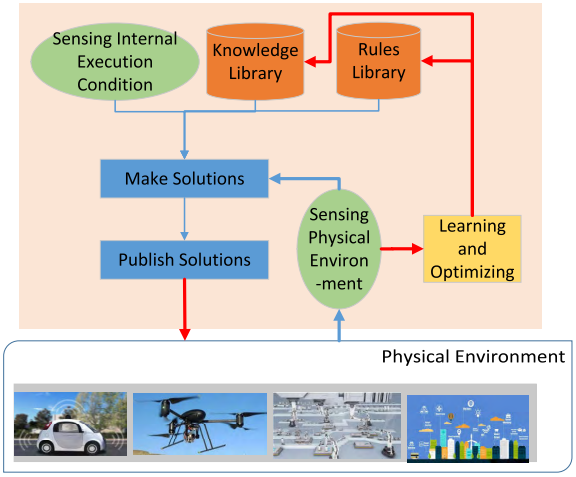

FIGURE 4. The architecture of the decision-making CPS-Agent. The red lines represent feedback and optimization processes.

The architecture is shown in the Fig. 4. Modules *Sensing Internal Execution Condition* and *Sensing Physical Environment* are similar to the reactive CPS-Agent, hence we won't describe again. *Knowledge Library* and *Rules Library* provide the basis for reasoning. The module *Make Solutions* makes reasoning according to all available information, and gets solutions. The *Publish Solutions* module interacts with *Physical Environment*. After solutions affect environment, the module *Learning and Optimizing* will a) gather data from the *Sensing Physical Environment*, b) use a learning algorithm to train data, c) optimize the *Knowledge Library* and *Rules Library* using trained results. In the Fig. 4, the red lines represent feedback and optimization processes.

3) HYBRID CPS-AGENT

The hybrid CPS-Agent combines the advantages of reactive and decision-making CPS-Agents. It is an intelligent and agile CPS-Agent.

The architecture of the hybrid CPS-Agent is presented in the Fig. 5. Modules of the reactive CPS-Agent and decision-making CPS-Agent are logically connected together to realize an intelligent and agile hybrid CPS-Agent. In the Fig. 5, the yellow lines show an optional process to deal with emergencies, aiming to have fast response; the red lines

TABLE 1. Communicative acts description.

Communicative Acts	Semantic Description
Request	The sender is requesting the receiver to perform some action. The content of the message is a description of the action to be performed, in some language the receiver understands.
Agree	Agreement to a previously submitted *request* to perform some action. The agent sending the agreement informs the receiver that it does intend to perform the action.
Cancel	*cancel* allows a CPS-Agent i to inform another CPS-Agent j that i no longer intends that j perform a previously requested action.
Refuse	Refusing to perform a given action, and explaining the reason for the refusal. The *refuse* act is performed when the agent cannot meet all of the preconditions for the action to be carried out, both implicit and explicit.
Failure	Telling another CPS-Agents that an action was attempted but the attempt failed. The reason for the failure is represented in the content of message.
Inform	The sender informs the receiver that a given proposition is true or an action is required to complete. *Inform* is a mandatory order.
Cfp	The action of calling for proposals to perform a given action or realize a task.
Propose	Making a proposal or responding to an existing proposal during a negotiation process by proposing to perform a given action subject to certain conditions being true.
agree-proposal	Acceptance of a proposal that was previously submitted to coordination group (typically through a *propose* act).
refuse-proposal	The action of rejecting a proposal to perform some action during a negotiation.

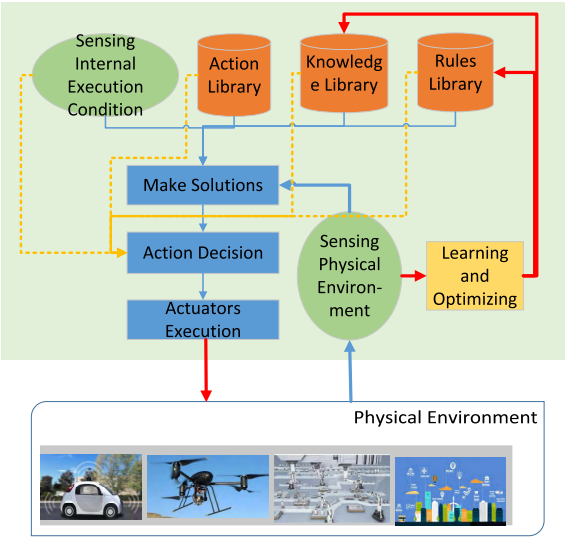

FIGURE 5. The architecture of the hybrid CPS-Agent. The red lines represent feedback and optimization processes. The yellow lines show an optional process to deal with emergencies.

represent feedback and optimization processes. The common processes are presented by blue lines.

As technologies are developing, development of the hybrid CPS-Agent is a dominant trend. Objects in CPS are usually advanced embedded systems. To model such objects, the CPS-Agent is required to have decision-making capability to deal with complex conditions. But as for emergencies, long-time thinking will badly impact on courses of events. Therefore reacting quickly is also required for the CPS-Agent. Some hot topics are to get trade-off between thinking precisely and reacting quickly.

As for these three kinds of CPS-Agents, we provide three different templates to generate codes automatically in our tool.

IV. COORDINATION SUPPORT FOR MULTIPLE CPS-AGENTS SYSTEMS

The CPS-Agents will coordinate for cooperative tasks. The complete processes involve the CPS-Agent personalization construction, multiple CPS-Agents coordination strategies establishment, communication primitives among CPS-Agents, and CPS-Agent coordination groups creation. A detailed explanation about the CPS-Agent construction has been given in section III. Therefore, we describe specifications of other three parts in the following.

A. COMMUNICATION PRIMITIVES AMONG CPS-AGENTS

The network communication is the basis of coordination. For practical CPS applications, the communication among CPS-Agents depends on the network protocols that they support. Based on that, in order to increase inter-understanding and inter-operation among CPS-Agents, we a) propose a set of communicative acts; b) standardize format of communication primitives.

1) COMMUNICATIVE ACTS

The *communicative act* is the most important mandatory item in once communication. We refer to the FIPA-ACL (FIPA-Compliant Communicative Act) communication specification [7], select and extend a part of communicative acts from FIPA-ACL. The detailed descriptions of these communicative acts are shown in the Table 1.

2) FORMAT OF COMMUNICATION PRIMITIVES

Table 2 shows the standard format of communication primitives. *XX* is a mandatory field while communicating, whereas *(XX)* is non-mandatory. *communicative act* is explained in section IV-A. Every piece of message must contain a communicative act. *:sender* is A_{id} of a CPS-Agent who sends this message. *:receiver* is A_{id} of a CPS-Agent who is expected to receive this message. *:sender* and *:receiver* are in the same group. *:strategy* is promissory coordination strategy for the group. *:strategy* is transmitted to cope with the situation that one CPS-Agent implements multiple strategies (it is possible because one CPS-Agent could perform different tasks and work with different CPS-Agents in multiple physical environments). The content of *:time* and *:space* are encapsulated as class *_Time* object and class *_Space* object respectively.

TABLE 2. Standard format of communication primitives.

(communicative act :sender A_{id} :receiver A_{id} :strategy XX :time (XX) :space (XX) :content (XX) :language XX :ontology XX)

TABLE 3. Members of class _time to describe the content of *:time*.

Members	Types	Description
start	string	The time actions begin to execute
end	string	The time actions end to execute
duration	string	Duration actions last
present	bool	Whether actions execute immediately
wait	string	After wait time, actions execute

TABLE 4. Members of class _space to describe the content of *:space*.

Members	Types	Description
leftUp	string	Top left coordinates of AAS
rightDown	string	Lower right coordinates of AAS
center	string	Central coordinates of AAS
radius	string	Radius of AAS
location	string	Location coordinates, related with *content*
Note AAS: allowed activity scope		

The class members and detail explanations of class *_Time* and class *_Space* are shown in Table 3 and Table 4 respectively. Note that we use class member variables to present some general spatio-temporary requirements, but some variables can be default. *:content* expresses specific requirements related to the scenario. *:language* is the pre-defined layout for *:content*. Engineers are able to customize *:language*, or use common layouts, such as RDF, OWL, KIF, etc. *:ontology* shows the source of terminologies used in *:content*. *:ontology* is designed to avoid ambiguity.

B. ROLE-BASED MULTIPLE CPS-AGENTS COORDINATION STRATEGIES

We cannot put forward a definite certain coordination strategy to support various specific CPS applications, because coordination strategy researching for CPS is an open and hot topic, and it is always related to more specific scenarios. Here, we propose a role-based methodology to design a coordination strategy. Specifically, we have read a lot of papers, and we find an interesting phenomenon that in a coordination scenario, multiple CPS-Agents usually have clear-cut assignments, therefore they play different roles and take autonomous approaches on task realizing. According to such phenomenon, we believe that designing a coordination strategy framework based on roles play is a feasible methodology, i.e. The engineers are suggested to develop a strategy where multiple roles are partitioned according to the requirements of the scenario, and then behavior patterns of the roles during coordination are designed.

Moreover, besides roles design, we select a subset of communication primitives to help realize missions of roles, since primitives have strict semantic definition. In summary, there are three phases for role-based strategy establishment: *naming, roles design, primitive selection*. Afterwards, engineers should add strategy details into function modules of the CPS-Agents, which are mentioned in section III-B.

C. MULTIPLE CPS-AGENTS COORDINATION GROUPS

There are usually some cooperative tasks that require multiple CPS-Agents to work together. Hence these CPS-Agents form a group where they use a promissory and approved coordination strategy for realizing tasks and improving efficiency, and they use communication primitives corresponding to the coordination strategy for inter-communication.

Forming a coordination group has four phases: *naming, members selection, strategy selection, roles playing*. Specifically, engineers need to give a name to the group firstly, and then select members of the group from the established CPS-Agents for coordination tasks, afterwards engineers select a strategy to guide behaviors of the group, finally according to the role-based strategy, engineers assign roles to CPS-Agents, requiring CPS-Agents to work based on the behavior pattern of roles.

V. DESIGN PRINCIPLES AND PROCESSES OF THE SUPPORT TOOL FOR CPS-AGENTS

The support tool is a platform-independent software that aims at helping engineers reduce modeling complexity and simplify development processes for CPS applications. In this part, we focus on the design principles and processes of the support tool. As depicted in the Fig. 6, the tool is based on the CPS-Agent model and coordination support protocols, and it consists of three major design and development steps. The result of this process is the support tool, which will output a set of C++ files corresponding to customized CPS-Agents, coordination strategies and groups by engineers.

In the following we explain the design and development processes for the tool in detail.

A. GRAPHICAL USER INTERFACE DESIGN

The support tool is a novel visual platform. We design graphical user interface (GUI) for the tool based on requirements of the CPS-Agent model, role-based strategies and coordination groups. The Fig. 7 shows a screen-shot of the support tool. The menu bar includes *File*, *CPS-Agent*, *Co-Strategy* and *Co-Group*. The *File* menu has only a function of exiting the tool. The other three menus correspond to the detailed explanation in section III-A, IV-B, IV-C respectively. Each menu option has a drop-down menu that usually includes *establish* and *delete*, which represent creation of a new item and the deletion of an established item, respectively.

The view of the left hand column in the Fig. 7 includes three important spaces: *CPS-Agent Space*, *Coordination*

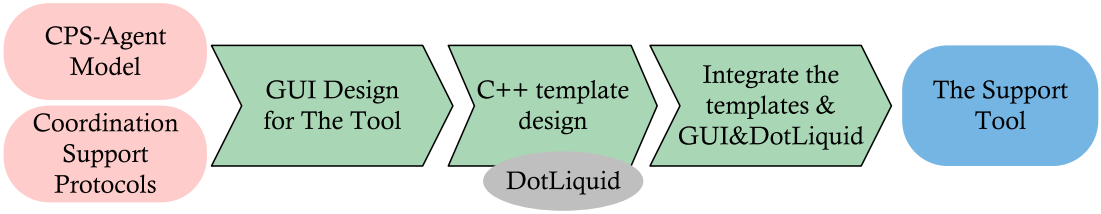

FIGURE 6. The design processes of the support tool.

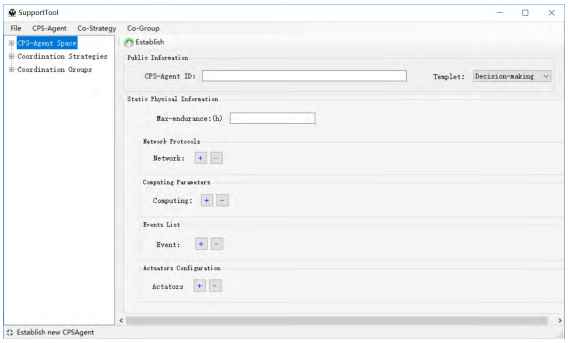

FIGURE 7. The graphical interface of establishing a new CPS-Agent.

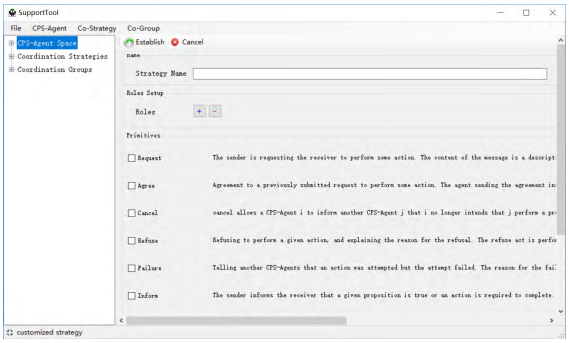

FIGURE 8. The graphical interface of making role-based coordination strategies.

Strategies and *Coordination Groups*. Each space has subdirectories where some established items are displayed. Every subdirectory saves all files related to the item, usually including a XML description file and a set of C++ code files. The contents of a file can be displayed in the right window by selecting the file on the left. The XML file cannot be edited. It helps engineers understand features and attributes of established CPS-Agents, strategies and group. C++ files are able to be edited, compiled and executed. They are independent executables with powerful expansibility. Engineers could design secondary development based on the C++ files, for example, they are able to complement control algorithms or optimization algorithms to the C++ files.

At the same time, Fig. 7 presents the first graphical interface design about how to customize a new CPS-Agent. As depicted in the Fig. 8, the user interface (UI) of making a role-based coordination strategy is shown. The UI design of establishing a coordination group for multiple CPS-Agents is presented in the Fig. 9.

B. STRUCTURE ANALYSIS OF THE C++ TEMPLATES

The next step is to design C++ templates,including parts of CPS-Agent construction and coordinate support. Here, we use *DotLiquid* to help generate C++ files automatically. *DotLiquid* [3] is a templating system ported to the *.net framework*. All C++ templates are designed to conform to *DotLiquid syntax*.

Then we focus on analyzing structures of the code templates, We have three templates to coping with the *Reactive CPS-Agent*, the *Decision-making CPS-Agent* and the *Hybrid CPS-Agent* respectively.

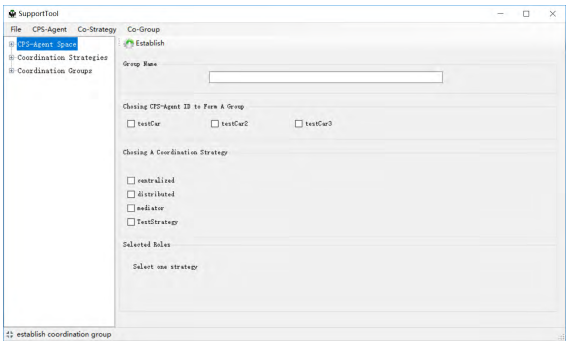

FIGURE 9. The graphical interface of establishing coordination groups.

To be specific, Class *action* includes functions of receiving events and coping with events. Class *think* have function *decide():void*, where engineers could design reasoning and decision-making algorithms. Besides that, Class *Action_CPSAgent*, Class *Think_CPSAgent* and Class *Hybrid_CPSAgent* include parameters definition of the customized CPS-Agent, overloading functions of the parent Classes, and peculiar functions. These significant Classes are designed to assist construction of the customized CPS-Agent and generation of corresponding C++ files of *Reactive CPS-Agents*, *Decision-making CPS-Agents* and *Hybrid CPS-Agents* automatically. Usually, Class *action* is inherited by *Reactive CPS-Agents*. Class *think* is inherited by *Decision-making* CPS-Agents. *Hybrid CPS-Agents* have the advantages of *Reactive CPS-Agents* and *Decision-making CPS-Agents*, hence the template inherits Class *action* and Class *think*.

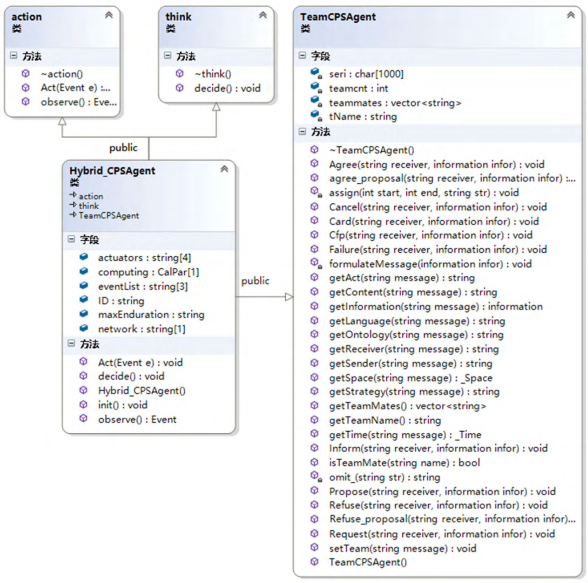

FIGURE 10. A UML class diagram of automatically generated C++ code files of the CPS-Agent *testCar*.

To support coordination for CPS-Agents, Class *TeamCPSAgent* is designed. It makes information translate to a string with a predefined format through the function *formulateMessage(information):void*. It could parse the string to information using the function *getInformation(string message):information*. Details of the information are able to be parsed through functions *getAct(string message), getContent(string message), getLanguage(string message)*, etc. Functions of transferring information with specific semantics are provided by *Agree(string receiver, information infor), Cfp(string receiver, information infor)*, etc. At the same time, members setting and judgement are realized through *setTeam(string message): void* and *isTeamMate(string name):bool* respectively. Class *TeamCPSAgent* is inherited by Class *Action_CPSAgent*, Class *Think_CPSAgent* and Class *Hybrid_CPSAgent*.

Here we illustrate the structure of code templates through automatically generated C++ codes of *testCar*, which will be instanced as a *Hybrid CPS-Agent* with coordination requirements in section VI. The Fig. 10 shows a UML class diagram of code files to present the inheritance relationship between Classes.

In order to analyze deeply of the relationship between C++ files, a code map is presented in the Fig. 11. We illustrate association relationships among *Classes, Structs and Functions* that are represented by gray lines in the Fig. 11. Function *main* is responsible for concrete operations and it creates a thread function *monitor*, which is responsible for monitoring incoming messages. Class *information* standards format and members of messages, its content is described in the Table 2. Class *hybrid_CPSAgent* is not associated with *think*, because the former overrides function *decision():void* of the latter, even if the former inherits the latter. As for class

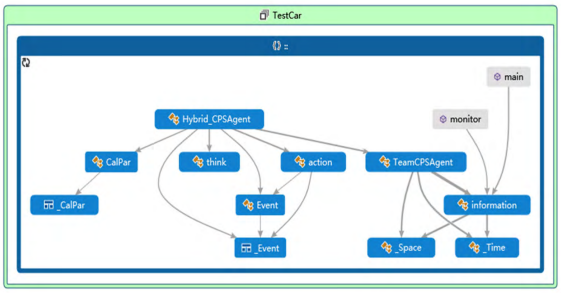

FIGURE 11. The code map of automatically generated C++ code files of CPS-Agent *testCar*. Gray lines represent association relationship.

Event and struct *_Event*, engineers could redefine elements that are need to pay attention. Other classes and structs are ancillary, including class *CalPar*, class *_Time*, class *_Space*, struct *_CalPar*.

C. INTEGRATE GUI & C++ TEMPLATES & DOTLIQUID

After designing and developing GUI for the tool, we need to connect *DotLiquid* to the tool, so that the tool has the powerful capability to generate C++ files automatically. Afterwards, C++ templates are integrated to the tool, too. Following the integration process, a visual support tool is available for engineers.

VI. DEVELOPMENT PROCESSES ON THE TOOL FOR ENGINEERS

In this part, we will present how engineers customize CPS-Agent, coordination strategy and groups based on requirements of CPS applications by an instance of a car team, and show the customized results.

A. INSTANCE BACKGROUND

A car team is responsible for loading and transporting goods. In general, the car drivers make decisions to cope emergencies based on their experience on the roads, with taking into account the performance indicators of the car. During the mission, car drivers must keep attention focused. It is expected to customize an unmanned system to carry out such tired tasks. For the best intention, the first step is to model the car.

Specifically, modeling the car is necessary, i.e. It is necessary to abstract the important attributes and characteristics of the car into objects that engineers can understand, so that engineers can participate in the development of CPS without complex control and dynamics knowledge.

B. DEVELOPMENT PROCESSES

This paper tries to finish the first step of developing an unmanned system, which is a classic cyber physical system. We first extract requirements of the CPS applications as the input, and then model cars using the CPS-Agent methodology. For cooperative tasks of the car team, we design a coordination strategy using the role-based strategy establishment

FIGURE 12. A summary of the development process for engineers.

methodology. Afterwards, a coordination group is established to support interaction of the car team. Finally, XML and C++ files are outputted to assist secondary development for engineers. A summary of the development process is depicted in the Fig. 12.

We will present development process in detail for CPS on the tool in the following.

1) CUSTOMIZE MULTIPLE CPS-AGENTS

Engineers would like to customize a new CPS-Agent, and they will extract some important information related to performance of the car. After that, they fill the items in the establishment window of CPS-Agent, shown in the Fig. 7. Following that, the tool will present a XML file corresponding to the CPS-Agent automatically in the right window of the tool, and a pop-up reminds engineers of successful creation of a CPS-Agent, as shown in the Fig. 13. At the same time, the Fig. 13 also presents that the left window of the tool will add a new subdirectory named *testCar* in the *CPS-Agent Space*. All files related to *testCar* are enumerated in this subdirectory.

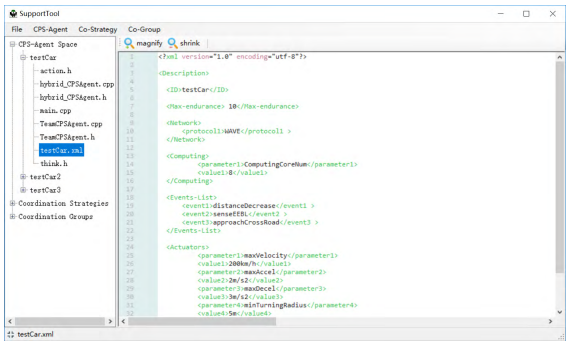

FIGURE 13. The XML file of the customized CPS-Agent.

Note that information about performance of the modeled object is more suitable to represent the features of the object and is easier to be understood by engineers than more concrete configuration conditions of the object. A classical example is to depict *Ac* (*Actuators Configuration*) in V_{sp}. We depict significant information related to performance of the car, including values related to velocities and rotate speed, instead of describing brands and versions of components, such as radar, gun, apron wheel, etc.

In this phase, we have depicted two elements,A_{pi} and V_{sp}, in the five-tuple of the CPS-Agent. The other two elements, V_{dp} and V_s, are changing as system running, hence we need to get information in real time. The remaining element G_m will be described in the following. We use the code template of the *Hybrid CPS-Agent* to model the car.

2) MAKE A ROLE-BASED COORDINATION STRATEGY

Repeating the process of establishing a CPS-Agent, engineers could model all cars in the car team. Then in this step, we will make a role-based coordination strategy. As depicted in the Fig. 8, three steps are required: *naming, roles design, primitive selection*. Here, we make a strategy named *TestStrategy*, and then we designs two special roles for the strategy, *leader* and *sub-leader*, to overtake different tasks. In step three, seven communicative acts are selected to assist multiple CPS-Agent with semantic communication.

After the button *Establish* being clicked, a XML file will be presented in the right window, and all files related to the strategy are listed in the subdirectory in the *Coordination Strategies* too. In this step, there is only one non-editable XML file being generated. The result is shown in the Fig. 14.

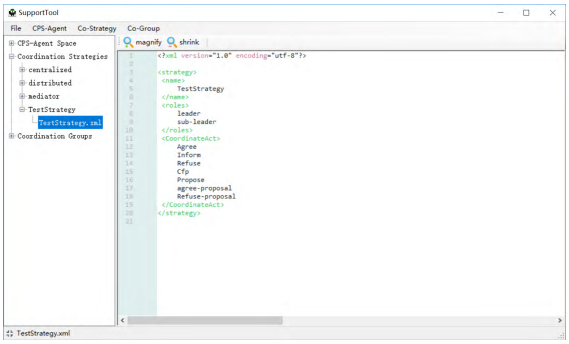

FIGURE 14. The XML file of the customized role-based coordination strategy.

3) ESTABLISH A COORDINATION GROUP

Following that, we will establish a coordination group for realizing cooperative application tasks. As depicted in the Fig. 9, there are four parts for the group establishment: *naming, members selection, strategy selection, roles playing*. No part can be default. For example, we establish a group named *TestGroup*, and select three CPS-Agents from established CPS-Agents as group members. Afterwards, the strategy *TestStrategy* is selected from existing strategies. Finally, as for roles that are designed in *TestStrategy*, we assign them to two group members respectively.

After being established, a XML file will be shown automatically, and all files will be saved in the subdirectory in *Coordination Groups*. The C++ files related to *TestGroup*

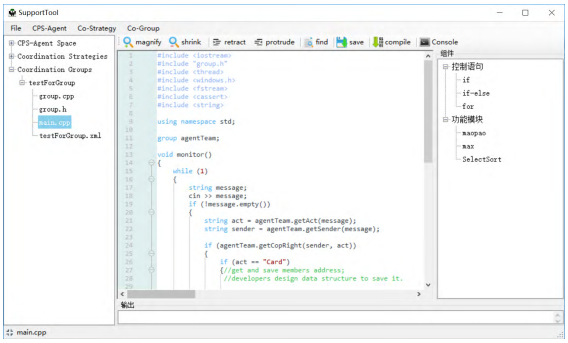

FIGURE 15. Presentation of automatically generated C++ files related to the coordination group.

could be edited, compiled and executed, as shown in the Fig. 15.

C. WORK OF ENGINEERS

Our tool has modeled cars successfully and generate C++ files automatically. But engineers should complement the remaining functions of modules of CPS-Agents in the secondary development. Works of engineers include a) designing functions of communication interface connection with other CPS-Agents and sensors with consideration of practical applications; b) providing domain-related *action libraries, knowledge libraries, rules libraries*; c) customizing *learning optimized algorithms* for the *Decision-making* CPS-Agents and *Hybrid* CPS-Agents; d) designing strategy details and programming to implement the details when CPS-Agents coordinate based on a role-based strategy for cooperative tasks.

VII. CONCLUSIONS AND OUTLOOK

In this paper, we introduced an innovative CPS-Agent methodology to model complex objects in CPS, and then we designed a support tool for generating a XML description file and C++ code files automatically corresponding to the CPS-Agent. An instance of a car team is used to illustrate the correctness and efficiency of our methodology and tool.

As for the CPS-Agent methodology, we a) presented a five-tuple to describe features and performance of objects in CPS, with consideration of spatio-temporary features, interaction with physical processes and physical environment; b) extracted important information of objects as elements in the five-tuple. Following that, to serve coordination requirements, we propose a role-based strategy formulation methodology. Afterwards, A set of communication primitives are defined to achieve inter-understanding and inter-operation among CPS-Agents. The strategy is bind with a subset of primitives. In the end, a coordination group is established for cooperative tasks. The group is related to the established CPS-Agents set and strategies set. Moreover, the support tool provides visual interfaces to assist development for engineers.

Future works will further evaluate our general approach on other complex applications. Moreover, we will further perfect the CPS-Agent templates, so that they can help engineers reduce development workload and difficulties. After that, we will focus on security problems that are crucial in CPS, including protecting CPS-Agent and the support tool against Trojans, securing underlying hardware related to CPS-Agent against security risks due to piracy threats, etc.

REFERENCES

[1] *Agent-Based Model Wikipedia*. Accessed: Jul. 5, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Agent-based_model

[2] *Component-Based Development Wikipedia*. Accessed: Aug. 18, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Component-based_software_engineering

[3] *DotLiquid Homepage*. [Online]. Available: http://dotliquidmarkup.org

[4] *Service-Oriented Architecture Wikipedia*. Accessed: Jul. 9, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Service-oriented_architecture

[5] F. Bergenti, E. Iotti, S. Monica, and A. Poggi, "Agent-oriented model-driven development for JADE with the JADEL programming language," *Comput. Lang. Syst. Struct.*, vol. 50, pp. 142–158, Dec. 2017.

[6] S. Dziwok, C. Gerking, S. Becker, S. Thiele, C. Heinzemann, and U. Pohlmann, "A tool suite for the model-driven software engineering of cyber-physical systems," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 715–718.

[7] *Communicative Act Library Specification*, document SC00037J, FIPA, 2002.

[8] C. Gerking, W. Schäfer, S. Dziwok, and C. Heinzemann, "Domain-specific model checking for cyber-physical systems," in *Proc. Workshop Model-Driven Eng.*, 2015, pp. 18–27.

[9] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[10] Y. Gottlieb and T. Shima, "UAVs task and motion planning in the presence of obstacles and prioritized targets," *Sensors*, vol. 15, no. 11, pp. 29734–29764, 2015.

[11] C. Heinzemann, O. Sudmann, W. Schäfer, and M. Tichy, "A discipline-spanning development process for self-adaptive mechatronic systems," in *Proc. Int. Conf. Softw. Syst. Process*, 2013, pp. 36–45.

[12] C. A. Iglesias, M. Garijo, and J. C. González, "A survey of agent-oriented methodologies," in *Proc. Int. Workshop Agent Theories, Archit., Lang.*, 1998, pp. 317–330.

[13] K. Kravari and N. Bassiliades, "A survey of agent platforms," *J. Artif. Societies Social Simul.*, vol. 18, no. 1, p. 11, 2015.

[14] P. Lin, J. Thangarajah, and M. Winikoff, "Tool support for agent development using the prometheus methodology," in *Proc. Int. Conf. Qual. Softw.*, Sep. 2005, pp. 383–388.

[15] X. Ma, Z. Jiao, Z. Wang, and D. Panagou, "Decentralized prioritized motion planning for multiple autonomous UAVs in 3D polygonal obstacle environments," in *Proc. Int. Conf. Unmanned Aircr. Syst.*, Jun. 2016, pp. 292–300.

[16] X. Ma, Z. Jiao, Z. Wang, and D. Panagou, "3-D decentralized prioritized motion planning and coordination for high-density operations of micro aerial vehicles," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 3, pp. 939–953, May 2018.

[17] D. Panagou, D. M. Stipanović, and P. G. Voulgaris, "Distributed coordination control for multi-robot networks using Lyapunov-like barrier functions," *IEEE Trans. Autom. Control*, vol. 61, no. 3, pp. 617–632, Mar. 2016.

[18] D. Panagou, "A distributed feedback motion planning protocol for multiple unicycle agents of different classes," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1178–1193, Mar. 2017.

[19] D. Panagou, D. M. Stipanović, and P. G. Voulgaris, "Distributed dynamic coverage and avoidance control under anisotropic sensing," *IEEE Trans. Control Netw. Syst.*, vol. 4, no. 4, pp. 850–862, Dec. 2017.

[20] U. Pohlmann, J. Holtmann, M. Meyer, and C. Gerking, "Generating modelica models from software specifications for the simulation of cyber-physical systems," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2014, pp. 191–198.

[21] U. Pohlmann, H. Trsek, L. Dürkop, S. Dziwok, and F. Oestersötebier, "Application of an intelligent network architecture on a cooperative cyber-physical system: An experience report," in *Proc. IEEE Emerg. Technol. Factory Autom.*, Sep. 2014, pp. 1–6.

[22] N. Saxena, V. Chukwuka, L. Xiong, and S. Grijalva, "CPSA: A cyber-physical security assessment tool for situational awareness in smart grid," in *Proc. Workshop Cyber-Phys. Syst. Secur. Privacy*, 2017, pp. 69–79.
[23] A. Sengupta and S. Kundu, "Guest editorial securing IoT hardware: Threat models and reliable, low-power design solutions," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 12, pp. 3265–3267, Dec. 2017.
[24] A. Sengupta, D. Roy, S. P. Mohanty, and P. Corcoran, "DSP design protection in CE through algorithmic transformation based structural obfuscation," *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 467–476, Nov. 2017.
[25] A. Sengupta, D. Roy, S. P. Mohanty, and P. Corcoran, "Low-cost obfuscated JPEG codec IP core for secure CE hardware," *IEEE Trans. Consum. Electron.*, vol. 64, no. 3, pp. 365–374, Aug. 2018.
[26] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, "Evaluation of agent–oriented software methodologies–examination of the gap between modeling and platform," in *Proc. Int. Workshop Agent-Oriented Softw. Eng.*, 2004, pp. 126–141.
[27] J. Tonn and S. Kaiser, "ASGARD—A graphical monitoring tool for distributed agent infrastructures," in *Proc. Adv. Practical Appl. Agents Multiagent Syst. (PAAMS)*, Salamanca, Spain, Apr. 2010, pp. 163–173.
[28] A. Tveit, "A survey of agent-oriented software engineering," in *Proc. 1st NTNU CSGSC*, vol. 1, May 2001.
[29] X. Vila, A. Schuster, and A. Riera, "Security for a multi-agent system based on JADE," *Comput. Secur.*, vol. 26, no. 5, pp. 391–400, 2007.

YUJIAO HU was born in Yulin, China, in 1993. She received the B.E. degree in computer science and technology from Northwestern Polytechnical University, Xi'an, China, in 2016, where she is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

Her current research interests include multi-agent planning, coordination, and distributed control of complex systems, with application in unmanned aerial systems and cyber-physical environment.

XINGSHE ZHOU (M'04) received the B.S. and M.S. degrees from the School of Computer Science, Northwestern Polytechnical University, Xi'an, China.

He is currently a Doctoral Mentor of the School of Computer Science, Northwestern Polytechnical University, the Director of the Shaanxi Embedded System Key Laboratory, the Director of the Shaanxi Cloud Computing Technology Engineering Research Center, and the Director of the Center for High-Performance Computing. He has authored two books and nearly 200 articles. His research interests include planning and coordination of complex distributed systems in the cyber-physical environment.