IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 23, 2017, accepted September 17, 2017, date of publication September 20, 2017,
date of current version October 12, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2754507

Model-Based Development of Knowledge-Driven Self-Reconfigurable Machine Control Systems

NAN ZHOU[1], DI LI[1], SONG LI[1], SHIYONG WANG[1], AND CHENGLIANG LIU[2]
[1]School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou 510641, China
[2]Institute of Mechatronics, Shanghai Jiao Tong University, Shanghai 200240, China

Corresponding author: Song Li (mesongli@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 51605168, in part by the Natural Science Foundation of Guangdong Province under Grant 2015A030308002, in part by the Science and Technology Planning Project of Guangdong Province under Grant 2015B010917001, and in part by the Fundamental Research Funds for the Central Universities under Grant 2017MS016.

ABSTRACT To accommodate the trend toward mass customization launched by intelligent manufacturing in the era of Industry 4.0, this paper proposes the combination of model-driven engineering and knowledge-driven engineering during the development process of self-reconfigurable machine control systems. The complete tool chain for model development, execution, and reconfiguration is established. For the design phase, a machine-control-domain-specific modeling language and the supporting design environment are developed. With regard to the execution stage, a runtime framework compliant with the IEC 61499 standard is proposed. On the ground of the modeling environment and the reconfigurable run-time framework, a self-adaptive control module is developed to establish the close-loop self-reconfiguration infrastructure. The ontological representation of knowledge base toward this end is described, along with extendable SQWRL rules specified to automatically initiate the reconfiguration process in the cases of external user demands and internal faults. A prototype motion control kernel in the low-level layer of machine control system architecture is developed with the proposed modeling language and is then deployed to the run-time framework. Two case studies on self-reconfiguration of the proof-of-concept motion control kernel are demonstrated, which prove the feasibility of our proposal.

INDEX TERMS Reconfiguration, machine control system, domain-specific modeling language, ontology, IEC 61499.

I. INTRODUCTION

Industry 4.0 has been boosting the manufacturing paradigm shift from sole mass production to mass customization, leading to increasing adoptions of the intelligent production mode. In this emerging scenario, manufacturing cells are required to be self-adaptive to changeable processing tasks within shorter transition cycles, compared with the traditional plants. The requirement on self-adaptiveness covers the communication infrastructure, control system as well as the physical components [1]. As the basic software units of manufacturing cells, machine control systems should be able to flexibly or even automatically reconfigure its structures and functions in the case of occurrences of external user demands and/or internal faults. Therefore, reconfigurable machine control system is one of the critical enabling technologies for the vision of intelligent manufacturing to become reality.

Typically, software for a machine control system is developed to provide stringent real-time performance under highly hardware-dependent architecture. However, manual coding, as the traditional software development pattern, is error-prone and time-consuming due to its low abstraction level. On the one hand, considerations on real-time behavioral diversities, such as the co-existence of cyclic position sampling and discrete event handling, complicate the process to a great extent. On the other hand, agile reconfiguration requires reusability and modularity of machine control software, which is usually built as a monolithic artifact. Therefore, innovative design philosophy and run-time architecture should be put forward to cope with the aforementioned challenges.

The concept of model-driven engineering (MDE) is considered as a critical solution to modern software development since it can ensure a high abstraction level, maintainability as well as flexibility. Model-integrated computation (MIC) is a

widely adopted model-based design paradigm for embedded real-time system, firstly put forward in [2]. MIC paradigm adopts domain-specific modeling language (DSML) instead of Unified Modeling Language (UML) for application developers who may have little knowledge of software ngineering. Definitions of DSML are based on proper conceptualizations and architectural abstractions of the target domain. In this paper, the IEC 61499 standard [3] is regarded as an appropriate guideline for this purpose. IEC 61499 complements the dominant IEC 61131 standard [4] mainly via introducing event-triggered function blocks (FB) as the basic model artifacts and a management model for reconfiguration of application logics. Therefore, although IEC 61499 is the standard for industrial automation applications running on the upper layer of machine control systems, the features it provides make this standard applicable for developing low-level reconfigurable machine control system.

MDE can pave the way to reconfigurable machine control systems. Then, intelligent systems can further require to minimize or eliminate human intervention during reconfiguration. Therefore, self-reconfigurability is necessitated. However, to achieve such a property, autonomy of the software should be implemented, which is beyond the scope of MDE. Therefore, other enabling technologies should be investigated. Recently, the academic and industrial community of industrial control has been directed towards an emerging tendency about applying knowledge engineering in the industrial software systems for achieving intelligences, flexibilities, etc. To utilize knowledge-driven approaches in the control software, an ontology being focused on specific application domain should be developed as a formal and machine-understandable knowledge base for the interacting software modules. New facts, such as reconfiguration plans, can be inferred by a decision-making module during runtime according to the ontology and collected information, including external user requests or internal state changes. In this way, self-adaptiveness can be realized. However, existing work has been mainly concerned with the higher layer of machine controls, such as the application logics or coordination tasks. Functions in the low-level layer, such as motion control, real-time communication, still rely on rigid and closed structure. This layer usually integrates various kinds of pre-defined configuration options which may be rarely utilized and hardly extended. These drawbacks will limit the self-reconfigurability of the whole system.

This article is an extended version of our previous work published in [5]. This extended version has the new contributions by introducing the combination of model-driven and knowledge-driven approaches as a holistic solution to implementing self-reconfigurability of low-level machine control systems. In addition, OPC UA, the standard industrial information modeling and communication protocol, is adopted for bridging the real-time control modules and self-reconfiguration modules in our proposal.

The remaining part of the article is structured as follows: Section II provides literature reviews on self-reconfigurable control systems. The overall architecture of our proposal targeting low-level machine control system is introduced in Section III. Following that, the meta-model definitions and the supporting run-time framework as the basis for reconfiguration are described in Section IV. Section V elaborates the ontological presentation of the meta-models as well as the process of self-reconfiguration based on knowledge-driven information collection and analysis. Section VI concludes the paper and outlines future work.

II. LITERATURE REVIEW

The paradigm of reconfigurable manufacturing system (RMS) is introduced in the 1990s [6], [7]. Since then, the industrial and academical community has seen several explorations on adopting RMS for higher level of flexibility and intelligence in response to fluctuating market conditions. Pritschow *et al.* identify in [8] the critical requirements for reconfigurable control systems from the perspectives of communication interfaces and architecture. In their opinion, software modules and communication protocol supporting the plug-and-play feature are regarded as the fundamental elements in RMS. They also clarify the necessary extensions of basic reconfigurable control system for realizing self-adaptability, including the modules of monitoring, decision and evaluation. Their work provides the general principles in the direction toward self-reconfigurable control systems.

Most of the existing related research concerns with the process control aspect in manufacturing systems. Feldmann *et al.* [9] present the pros and cons of respectively applying model-based engineering and semantic web service orchestration in automation control systems. Combinations of both two approaches to increase agility during the engineering and run-time phases are discussed as well. However, since their control systems are developed on the basis of IEC 61131-3 solution, dynamic reconfiguration of functions and structures during run-time may not be supported.

In [10] an agent-based approach is proposed to facilitate self-reconfiguration of automation control systems. A multi-layer architecture for automation agents containing low-level real-time process control and high-level coordination control is presented. The high-level control relies on a knowledge base, which is an ontological representation of physical situations and component functions. The low-level control is based on the IEC 61499 architecture, and this layer interacts with physical control process directly. The high-level layer observes the environment of the system and initiates reconfiguration process when needed, by translating reconfiguration steps into IEC 61499 management commands. The reconfiguration steps are inferred by specific reasoners in the high-level layer, while the commands are executed in the low-level layer. In this way, the proposed system can self-adjust dynamically when failures occur. A similar approach is proposed by Lepuschitz *et al.* [11]. They further take timing constraints into accounts during the reconfiguration process.

Dai *et al.* [12] apply autonomic service management in the context of service-oriented IEC 61499 function block

run-time environment for baggage handling systems. On-the-fly self-reconfigurability is achieved by the closed loop structure between a function block execution environment and a rule-based query engine. Integrations of service-oriented architecture and mapping FBs as services enable reconfiguration of the application logics in a platform-agnostic way, although performances may be deteriorated.

The related work reviewed above mainly focuses on the application logics layer of control systems. In the low-level control layer for motion control or real-time communication, more stringent timing constraints should be meet, which complicates the implementations of reconfigurability in this layer. Usually various kinds of pre-defined configurations may be integrated in the software to achieve limited reconfigurability. Regulin *et al.* [13] present a multi-master approach for dynamical reconfiguration of EtherCAT-based communication process. Different possible configurations of control devices are integrated in the corresponding masters. Then, these masters are hard-wired in the communication layer. When reconfiguration is required, the corresponding master will be activated in accord with the current physical layout of control devices. Such an approach is straightforward whereas has little extendibility and adaptability.

Lesi *et al.* [14] propose a plug-n-play numerical control architecture as the foundation for implementing RMS. In their architecture, the complete numerical control functions are identically duplicated in decentralized axis controllers. In this way, new axis modules can be introduced dynamically without affecting other working controllers. In their implementations, these axis controllers coordinate via regular wireless network. Therefore, timing constraints of multi-axis motion control may not be guaranteed.

Other related work in the low-level reconfigurable control system can be seen in [15] and [16]. To sum up, these works usually place the emphasis on the architectural design and the specification of component models. The development methodology, however, still remains quite primary from the perspective of abstraction and automation level. This contribution aims to address the problem of design methodology for low-level machine control systems. A model-based developing language is presented. Then, supporting tools and run-time architecture should be developed. Since knowledge-driven approach has been proved to be promising in enhancing self-adaptiveness in the domain of process control, our architecture will integrate such an approach for achieving higher level of intelligence.

We propose our solution on the ground of the MIC paradigm and its supporting toolkits. Reference [17] presents a model based integration framework to cover the entire life-cycle of CNC system. Their framework is conformed to the MIC paradigm. The modeling language of the proposed framework, CNCML, takes into account domain features and formal behavioral semantics synthetically. In [18] a similar DSML for cyber-physical robot control system is proposed according to the MIC pradigm. However, the final systems of these work rely on generated code, which can not be modified during execution. Lack of built-in mechanism for dynamic reconfiguration in these DSMLs lowers the flexibility of the framework. Thramboulidis [19] proposes a model-integrated mechatronics (MIM) paradigm for manufacturing systems developments. The proposed paradigm facilitates concurrent design of mechanical, electrical and software components of mechatronic systems by utilizing models as the center artifacts of system development. Runtime deployment and re-deployment of models are implemented to achieve dynamic reconfigurability. Although the MIM paradigm has several similarities with the MIC paradigm, it does not explicitly consider formal semantics definitions.

III. OVERVIEW OF METHODOLOGY

For the vision of self-reconfigurable machine control systems to become reality, several critical modules covering the different phases of the systems should be considered. Firstly, a modeling language and its supporting development environment should be established for the design stage. Then, with regard to the execution stage, a run-time framework is required for supporting dynamical reconfiguration. With these elementary modules available, the knowledge-driven self-adaptive control module can be developed and integrated in the run-time framework. In this way, self-reconfigurability can be achieved. The overall architecture of our proposal is illustrated in Fig. 1.

We firstly explore the model-based design approach to machine control systems for reconfigurability through the MIC paradigm. Application of MIC paradigm for the development of machine control system needs considerations on both design layer and execution layer. In the design phase, developers construct logics and functions of the machine control system by means of domain-specific models. The constructed system model can be translated into analyzable input models of other verification tools to find out design issues and system defects in early stage. Therefore, modeling language tailored to machine control domain should be defined to facilitate such a model-based development process. Structural and behavioral abstractions are the primary tasks during the specification of a DSML since they are corresponding to the syntactical and semantical definitions, respectively. Then, a runtime framework is indispensable for bridging design phase and runtime phase. In the runtime stage, models are executed and coordinated under non-functional and functional constraints.

According to the requirements for the two layers stated above, we introduce an executable architecture for realizing MIC based development of machine control system, as shown in Fig. 2. The design-phase implementation is corresponding to the part of modeling environment in Fig. 1, and the execution-phase implementation is related to the real-time control module deployed in machine controllers. The architecture focuses on the design- as well as execution-phase implementation of the MIC paradigm. For design-phase implementation, a modeling language specific to the machine control domain is defined, both syntactically

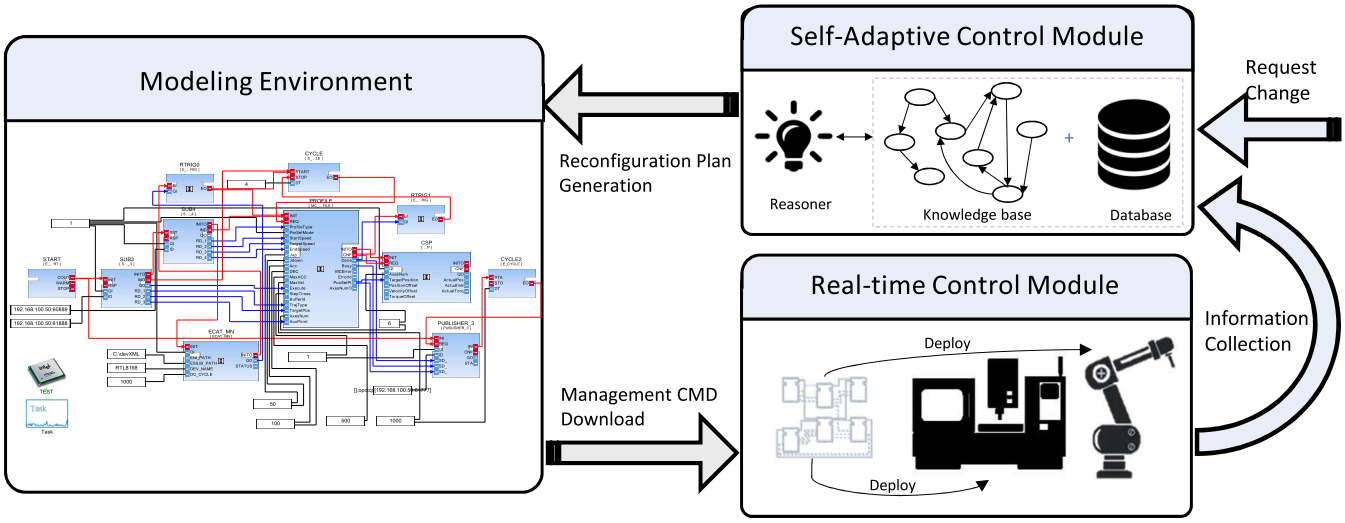


FIGURE 1. The overall architecture for self-reconfigurable machine control systems.

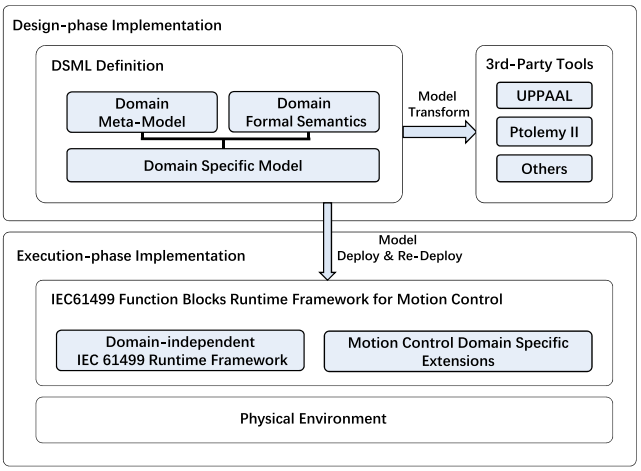


FIGURE 2. The architecture for realizing MIC paradigm based development of machine control system.

and semantically. Syntactical specification is achieved with meta-modeling language while semantical specification is done by model transformations to other formal models in third-party tools. In this paper, the semantical specifications will not be elaborated since they are out of the scope.

With regard to the implementation of runtime framework, we firstly propose a general, domain-independent execution environment which realizes reference models and architecture of the IEC 61499 standard. Management model is implemented to allow run-time creation, deletion of IEC 61499 FB types/instances and other primary elements. Decoupling of design-time and run-time FBs container based on resource model and the concept of function block chain model are put forward to integrate various applicable execution semantics. Besides, event connections are extended with the annotation of priority to enhance runtime determinacy of FB execution sequence. Based on the achieved execution environment, libraries of algorithms, FBs and resources for low level motion control tasks are developed. The motion control FBs play the role of primary units during the development and execution of machine control system.

Depending on the proposed architecture, developers can conduct model composition and synthesis for implementing machine control system on a higher abstraction level. Model deployment is carried out after the design process to construct the final system with reconfigurability, flexibility and robustness. Such an architecture can pave the way to the self-reconfigurability. A self-adaptive control module can be further developed and integrated on top of the runtime framework. As shown in Fig. 1, this module contains a knowledge base represented with ontology, a database and a reasoner. In addition, a communication interface is also required to collect information from the real-time control modules. In our current work, OPC UA is employed to this end. To implement such a module, counterparts of the meta-models for our proposed DSML should be represented with ontology in the knowledge base. Besides, triggering conditions of a reconfiguration process are also defined in the knowledge base, in the form of rules attached to the ontology. In this contribution, two types of scenarios are considered for triggering reconfigurations in the real-time control modules. The first type is changing of user demands on functionalities of machine controls, while the second type is internal errors regarding the communication process. The triggering conditions in these scenarios are termed as symptoms in the following section. The symptoms are collected via OPC UA and will be analyzed to generate corresponding reconfiguration action requests. Such an approach has been proved to be feasible in related work concerning the layer of application logics [11], [12].

After describing the overall architecture of our proposal, we apply it in the development of low-level machine control systems to accomplish self-reconfigurability on the layer of

motion control and real-time communication. Such a layer is the common infrastructure in the control kernel of CNC, robot and other mechatronic devices. Usually, this layer requires several indispensable components shown in Fig. 3.

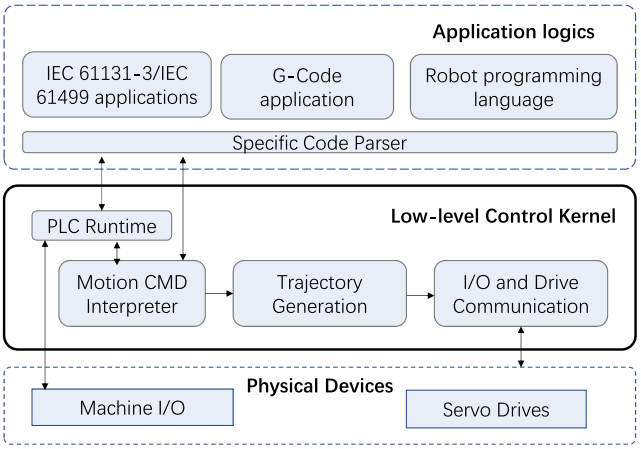


FIGURE 3. The functional modules of machine control system.

We focus on the functions regarding motion control of the low-level kernel in the machine control systems, including trajectory generation and drive communication. In distributed motion control system based on real-time ethernet, drive communication is responsible to the control loop in traditional centralized systems. As for the trajectory generation component, a set-point interpolator and velocity planner are required. Depending on the execution order of these two functions, two different trajectory generation algorithms can be defined [20]: Acceleration/Deceleration-After-Interpolation (ADCAI) and Acceleration/ Deceleration-Before-Interpolation (ADCBI). ADCBI is usually adopted in the scenarios where more precise and smooth movements of axes are demanded. On the contrary, ADCAI is less computation-intensive but it will introduce machining errors inevitably. We adopt ADCBI in the development of machine control kernel, with supporting two types of velocity profiles for reconfiguration. In the following sections, the drive communication and trajectory generation modules will be developed with our proposed DSML. Then, the model artifacts are deployed to the proposed runtime framework.

IV. MODELING LANGUAGE AND SUPPORTING RUN-TIME RECONFIGURABLE FRAMEWORK

In this section, we elaborate on the meta-model definitions of the machine control specific modeling language and the supporting engineering and runtime environment. Then, we utilize these achievements to construct a proof-of-concept machine control kernel with basic reconfigurability.

A. DEFINITIONS OF MODELING LANGUAGE

We firstly provide the meta-model definitions of our machine control specific modeling language according to the MIC paradigm. The meta-models are defined using the Generic Modeling Environment (GME) [21], which is a meta-programmable environment. The defined meta-models are encoded as an external rule file of GME in the format of extensible markup language (XML). Therefore, the proposed meta-models can be flexibly modified, extended and upgraded without affecting the modeling environment. Therefore, modeling efficiency can be enhanced compared with traditional dedicated tools.

Specifically, in the GME tool, syntax is defined with the built-in meta-modeling language called MetaGME. MetaGME provides facilities for meta-modeling based on UML-style class diagrams, including `Model`, `Atom`, `FCO`, `Connection`, `Proxy`, `Reference` and so on [22]. `Atom` is the elementary meta-model artifact, which can be contained in `Model`. `FCO` (First Class Object) is the generic abstraction of other meta-modeling concepts. `Reference` is the concept similar to pointer in some programming languages, such as C/C++. `Proxy` is an auxiliary concept for describing the identical objects in the complicated meat-models.

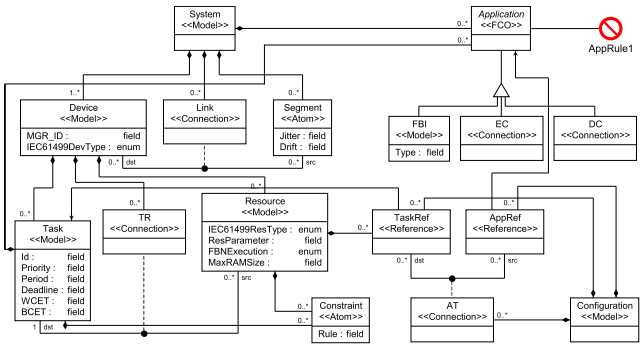


FIGURE 4. The meta-models of our DSML in MetaGME notation (core parts).

The root element in the DSML definitions is the system model, as shown in Fig. 4. The system model is composed of device models, application models, communication segments and the relations between these models. Such definitions are compliant with the IEC 61499 standard. The device model can contain multiple resource models and task models. The resource model is the abstraction of a hardware computing unit on which several tasks are executed. In this sense, the definition of resource model in our DSML is different from what it is in the IEC 61499 standard. The task model is the extension of the IEC 61499 standard for describing scheduling objects and for capturing timing constraints during the design process. The task models should be assigned to a specific resource model by developers. The application model in the system layer consists of function block network interacted via event and data connections. The FBs and related connections will be allocated to specific tasks, as defined in the configuration model. Some constraints are simply defined as attributes of the models, such as the timing parameters for tasks (deadline, worst-case execution time, etc.), while

more complicated requirements can be specified with the atom called constraints in Fig. 4. Finally, meta-models for the function block model, are described in Fig. 5. Since the definitions of FB have been extensively studied, they will not be elaborated in this section.

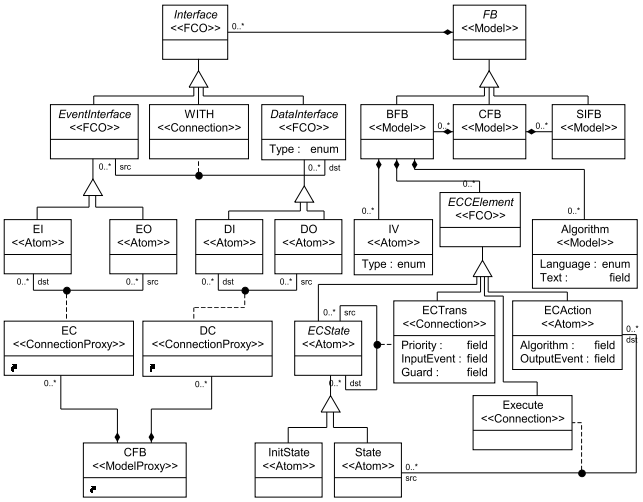


FIGURE 5. The meta-models of FB in our DSML in MetaGME notation (core parts).

On completing the meta-model definitions of the DSML for machine control system, the corresponding domain modeling environment can be set up automatically via built-in meta-model interpreter of GME. Developers can compose their machine control applications in the environment in a more intuitive and effective manner. A proof-of-concept motion control kernel is developed with the proposed DSML and the supporting environment, as shown in Fig. 6. The kernel runs on common hardware platform instead of proprietary embedded solutions. In our case, an industrial personal computer (i.e., the `SoftMotionController` model in Fig. 6) and the X86 platform (the `MotionControlCore` model in Fig. 6) are employed, where the operating system is Windows 8 patched with real-time extensions (RTX64 3.0) from the Interval Zero Inc. The control kernel interacts with three servo drives directly via the EtherCAT protocol, which is a widely adopted real-time ethernet protocol in the industrial automation domain. We model a basic motion control task in this case, which contains the `PROFILE` FB for trajectory planning based on the S-shaped velocity control algorithm, as well as the `CSP` FB for cyclic synchronized position control via EtherCAT. The `ECAT_MN` FB represents the EtherCAT master, and the `STATUS` port of `ECAT_MN` is connected to the `SD_1` port of the publisher FB `PUBLISHER_3`. The publisher FB `PUBLISHER_3` will transmit the values of `SD_1`, `SD_2` and `SD_3` to other subscribers via OPC UA. In this case, the subscriber is corresponding to the communication interface of the self-adaptive control modules. The example models will be deployed to our proposed run-time framework for implementing machine control functions.

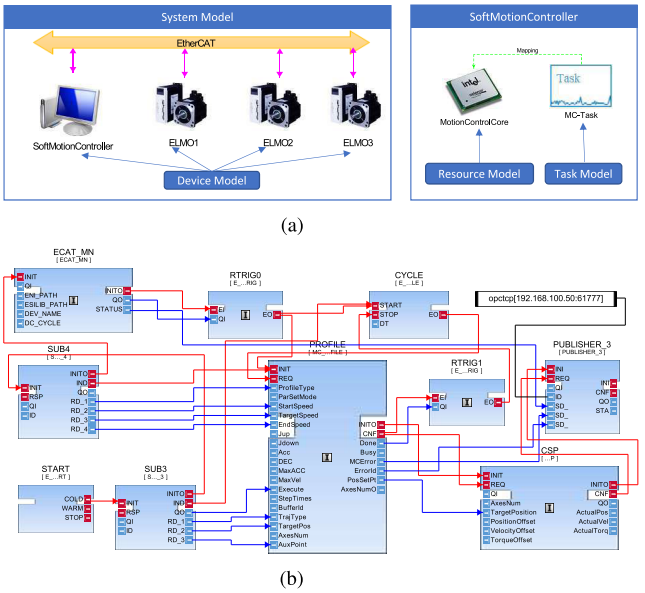


FIGURE 6. Models for implementing motion control kernel. (a) The system model and device models. (b) The application model.

B. DESIGN OF RUN-TIME RECONFIGURABLE FRAMEWORK

In this section, a run-time framework compliant with the IEC 61499 standard is presented for implementing the MIC paradigm in execution phase. Firstly, a domain-independent IEC 61499 execution environment is implemented. Then, an extended framework containing algorithms, libraries of FB types and resource types specific to the machine control domain are integrated. Reconfigurability is realized through implementing the IEC 61499 management model. The architecture of the proposed framework is illustrated in Fig. 7. The framework contains IEC 61499 model execution layer, algorithm libraries, EtherCAT master stack and OS-specific layer. The IEC 61499 model execution layer is responsible for implementing the reference models of IEC 61499. The algorithm library provides various kinds of algorithms for motion controls, such as trajectory planning, kinematics, control law, etc., which are invoked by FBs in the upper IEC 61499 model execution layer. The OS layer is defined to provide a programming interface with consistent style across multiple platforms for algorithm layer, through encapsulating hardware-dependent functions.

The execution environment implements the reference models defined by IEC 61499 along with several extensions, including FB chain model and event priority. The concept of FB chain model is corresponding to the task model in the design phase, which is proposed as the dynamic information container of FB models while the IEC 61499 resource model being the static information container. Besides, priority-denoted event connection is proposed to enhance determinacy. The binary-heap based priority queue is adopted for implementing event propagation mechanism among FB network. Furthermore, integration of various existing execution

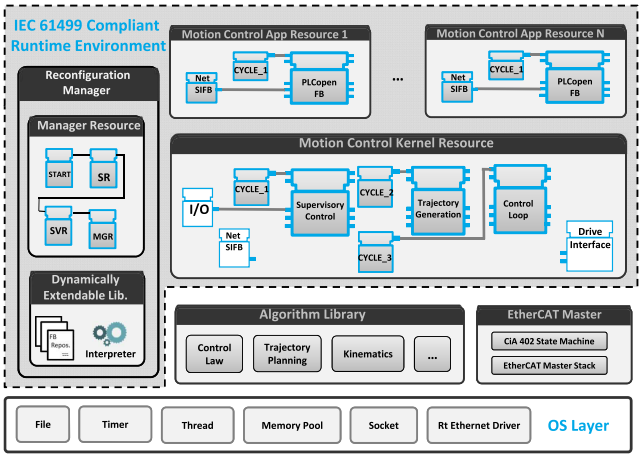


FIGURE 7. The architecture of the proposed run-time framework.

semantics is considered to tame behavioral heterogeneity after applying machine control domain specific constraints. Preliminary implementation of the standard run-time IEC 61499 management functions is also achieved, which makes reconfigurability possible in the final system.

In our proposal, the FB chain model acts as the container of dynamic information, such as the ID of instance name, active event and execution state of FB instances. The FB chain model contains a series of inter-connected FBs which starts from Event Source FB (ESFB) and ends with Event Sink FB (ESKFB), forming an independent execution task during the scheduling process. Each FB model is assigned to only one FB chain for avoiding possible concurrent triggering.

On the other hand, static information of FB and FB networks resides in the resource model, including the ID of type name, interface definition, internal structure, etc. In particular, the type information of FB chain model is also stored in the resource model. Several types of FB chain model are defined to respectively implement a specific kind of execution semantics. Correspondingly, various kinds of resource model are established and each kind of resource model maintains a pointer list referring to specific type of FB chain model. Therefore, multiple FB chain instances with identical execution semantics can exist in one resource model while different kinds of resource models may vary in execution behaviors. In this sense, integration of multiple execution semantics in a single framework can be achieved. Currently, the cyclic-scan based semantics and sequential event-trigger based semantics are integrated in the runtime environment.

Implementation of the management function is another critical part of the execution environment since it enables dynamic reconfigurability. The execution environment supports all basic management commands defined in IEC 61499, except operations on data type. Developers can deploy designed models to the environment using these XML-formatted commands, and the execution environment interprets the commands to carry out related operations dynamically. With respect to dynamic creation of FB types, the paper presents an interpreter-based proposal. During design-phase process, the design tool translates user defined algorithms in FB type definition file to C language and then the processed file is deployed to the execution environment for interpreting. The proposal is implemented through the adoption of a high-performance and portable C language interpreter: Tiny C Compiler (TCC).

The extended framework for machine control domain is based on the IEC 61499 execution environment. On the OS layer, hardware-specific axis objects and axes group objects are defined for interacting with drives or stepper motors directly. On the upper layer, algorithms for motion control tasks, including control law (such as adaptive PID control), trajectory planning (such as trapezoidal/S-shaped velocity profiling), kinematics, etc., are encapsulated in port-based modules.

A resource model for low level motion control and several resource models for high level applications are integrated on the IEC 61499 model execution layer. The former model concerns about motion commands interpreting, trajectory planning and drive controlling. The specific functions are implemented in various FB chains. The latter model forms the user-specific application layer of machine control system, containing automation tasks defined by the users, which are usually related to dedicated manufacturing processes. These types of resources are modeled in the same modeling environment, but they adopt different kinds of FB type libraries. In our case, the designed models shown in Fig. 6 are deployed to the run-time framework to implement low-level motion control functions.

V. KNOWLEDGE BASE DEFINITIONS AND SELF-RECONFIGURATION IMPLEMENTATION

The proposed knowledge-driven self-adaptive control module is introduced in this section. Firstly, the meta-models defined in Section IV-A will be mapped to the Terminological box (TBox) in the knowledge base ontologically, while the example application in Section IV-B is mapped to the Assertional box (ABox). Status of the run-time framework deployed in the controllers will be collected by in-house developed OPC UA historian client, which stores the collected information in rational database. The self-adaptive control module will periodically monitor the status via querying database and mapping retrieved values to data properties of individuals in the knowledge. The monitoring results are analyzed to generate symptom assertions. The generated symptoms will be further inferred to figure out corresponding reconfiguration requests in the knowledge base. These requests are forwarded to the modeling environment for figuring out the management commands, which will be executed in the run-time environments for reconfiguration. In this way, self-reconfiguration of low-level machine control system can be achieved.

The ontological models defined in this section include the IEC 61499 library elements, the extension of task model as well as the concepts for self-adaptive controls. Fig. 8 shows

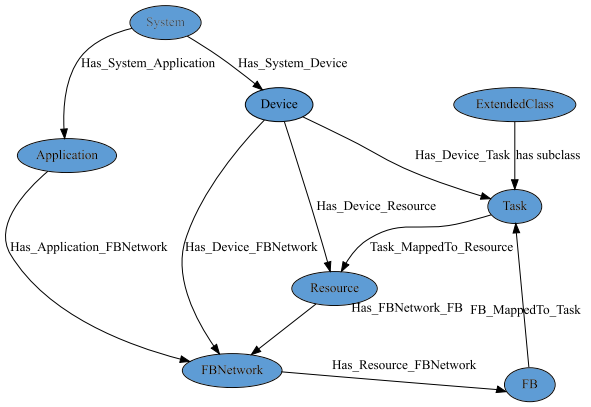


FIGURE 8. The ontological models corresponding to the meta-models of system/device/resource/task.

the ontological models corresponding to the meta-models defined in Fig. 4, while Fig. 9 illustrates the concepts for self-adaptive control. Fig. 10 represents the ontological models of function block. Due to space limit, only critical parts of these ontology are given. With regarding to the self-adaptive control elements, we define two main concepts: `symptom` and `action request`. The concept of `symptom` is proposed to summarize possible situations where reconfigurations are required, while `action request` is for describing the concrete solutions in the case of specific symptoms. The requests will be interpreted by the modeling environment as sequences of IEC 61499 management commands.

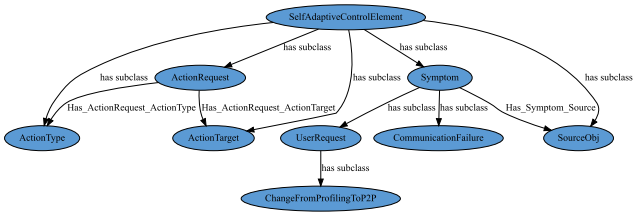


FIGURE 9. The ontological models of concepts involved in the process of self-reconfiguration.

A. EXTRACTION OF SYMPTOMS

The concept of symptom in our ontology models is expressed as the following axiom in the description logic:

$$\begin{aligned} Symptom \sqsubseteq (&= 1\ Has\_Symptom\_Name.String \\ &\sqcap = 1\ Has\_Symptom\_Source.SourceObj \\ &\sqcap = 1\ Has\_Symptom\_SourceObjName.String) \end{aligned}$$

Where *SourceObj* refers to the object (e.g., FB, resource or device models) which needs to be reconfigured.

In our current proposals, two types of typical symptoms are defined: `UserRequest` and `CommunicationFailure`. The former symptom stands for external changes of user demand regarding functions of machine control systems, such as changing from trapezoid velocity profiling algorithm to S-shaped profile in the trajectory planning FB. The latter one concerns with the failures occurring during the communication process, such as break down of communication link, or topology modifications of field devices.

Currently, user request symptoms are inserted into knowledge manually. On the other hand, the symptoms regarding communication can be extracted automatically by the monitoring function in the self-adaptive control module. We propose an extendable approach to extract and analyze such symptoms from the run-time environment. Firstly, the concerned information of machine control kernel will be configured during the design process by connecting the corresponding data ports of specific FBs to communication FBs (for example, `PUBLISHER_3` in Fig. 6). These communication FBs publish the values of connected ports via OPC UA. An in-house developed OPC UA historian client will subscribe to these value nodes and store the collected values to rational database.

When the developed models are deployed to the run-time environment, the self-adaptive control module can be started, which will repeatedly execute monitoring and analyzing functions. The basic procedure of monitoring and analyzing is illustrated in Fig. 11. The monitoring function will query the database at the interval of 50 milliseconds. In details, it retrieves values from the database with external SQL rule files. Then, the monitoring function executes SPARQL queries on the knowledge and updates the corresponding data properties of individuals in the knowledge base according to the results. The mapping relations are defined as external rules with the Ontop platform[1] and reasoner, which provides a configurable way to relate counterpart information in knowledge base and data base. After updating the knowledge with retrieved information from the database, new symptoms are inferred according to the current facts using SQWRL query rules. In this way, the monitoring function can be application-agnostic since all of the rules are defined externally and can be modified without reprogramming the module.

An example for demonstrating this procedure is described as follow. Regarding the models defined in Fig. 6, when break down of communication link occurs, the `STATUS` port of `ECAT_MN` FB will report an error message with the string value "*SLAVELOST*", which is configured to be published via OPC UA and stored in database. The monitor function firstly relates this value to the data property of corresponding individual in the knowledge though the rule defined in List. 1. The second step is to execute externally pre-defined SQWRL queries. The rule for extracting the symptom of communication failure is defined in List. 2. The antecedent part of this rule can query all the individuals which have a output variable named "STATUS" whose string value is "SLAVELOST". The results of the query are provided according to the consequent part of the rule, which will be used for inserting the symptom of communication failure by the monitoring function. The inferred symptoms will be further analyzed to generate appropriate action requests for reconfiguration.

[1] http://ontop.inf.unibz.it/

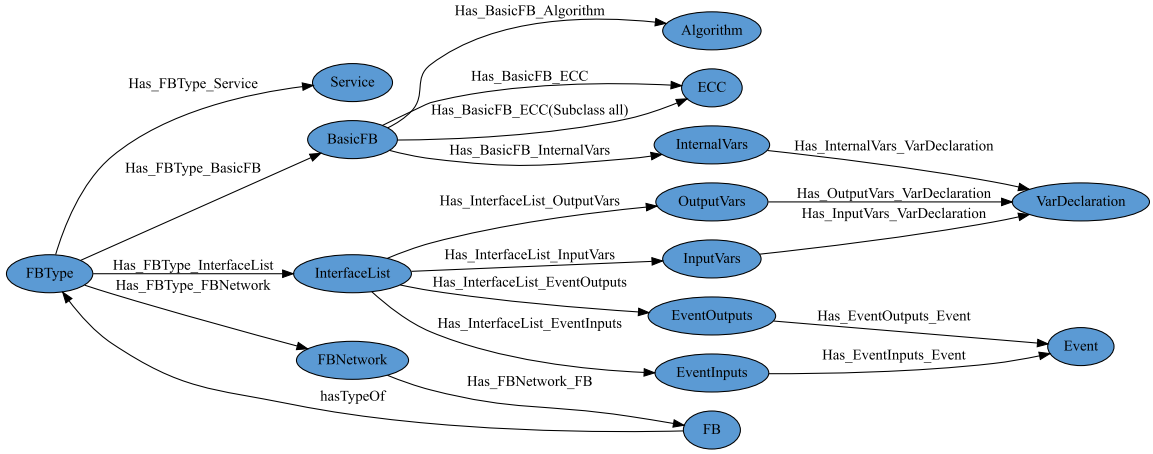


FIGURE 10. The ontological models corresponding to the meta-models of FB.

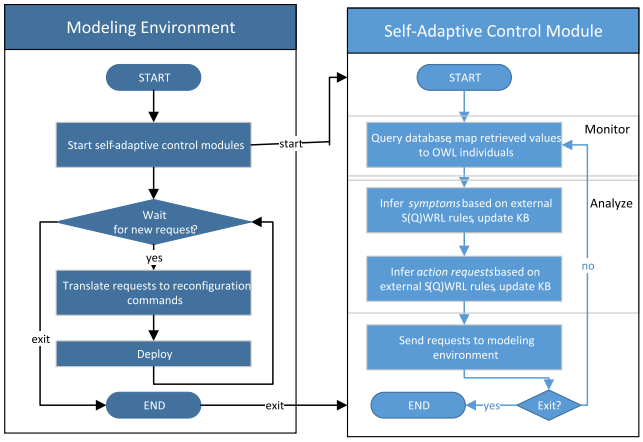


FIGURE 11. The execution procedure of modeling environment and self-adaptive control module during reconfiguration.

```
target          :STATUS a :VarDeclaration;
:Has_VarDeclaration_StringValue {currentvalue}.

source          select currentvalue from
collectionitem where name = 'Variable-STATUS'
```

List. 1. The mapping rule for correlated value of `STATUS` in database and data property of `STATUS` individual in knowledge base.

B. GENERATION OF ACTION REQUESTS

The concept of action request in our ontology models is expressed as the following axiom in the description logic:

$$\begin{aligned} &ActionRequest \sqsubseteq (\\ &\quad \geq 1\ Has\_ActionRequest\_ActionType.ActionType \\ &\quad \sqcap\ \geq 1\ Has\_ActionRequest\_ActionTarget.ActionTarget) \end{aligned}$$

Where `ActionType={CREATE, DELETE, RESTART, REPLACE}`, and *ActionTarget* refers to the object (e.g., FB, resource or device models) which needs to be reconfigured.

```
FB(?fb) ^ hasTypeOf(?fb, ECAT_MN) ^
hasTypeOf(?fb, ?type)
^ Has_FBType_InterfaceList(?type, ?inf)
^ Has_InterfaceList_OutputVars(?inf, ?vos)
^ Has_OutputVars_VarDeclaration(?vos, ?vo)
^ Has_VarDeclaration_Name(?vo, ?voname)
^ swrlb:stringEqualIgnoreCase(?voname, "STATUS")
^ Has_VarDeclaration_StringValue(?vo, ?vovalue)
^ swrlb:stringEqualIgnoreCase(?vovalue,
“SLAVELOST")
-> sqwrl:select(?fb, CommunicationFailure)
```

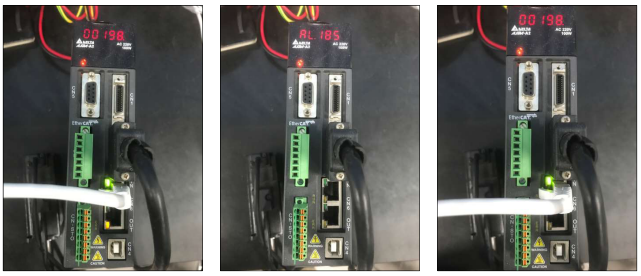
List. 2. The SQWRL query example for extracting the symptom of communication failure.

The analyze function in the self-adaptive control module will execute another group of SQWRL queries to generate action requests for specific symptoms. Based on the query results, the self-adaptive control module insert new facts of action requests in the knowledge and notify the modeling environment to generate the corresponding IEC 61499 management commands. In the way, the close-loop structure among the run-time framework, modeling environment as well as the self-adaptive control module can be realized.

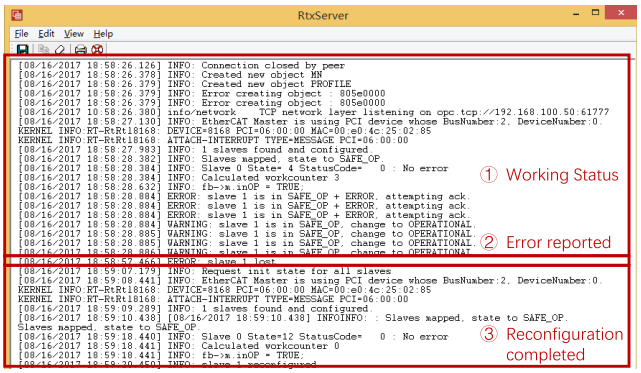
Two cases for automatically generating action requests are discussed in this section. The first one concerns with the previous symptom of communication failure, while the second deals with the symptom of user request of changing velocity profile algorithm. The SQWRL query example for the communication failure symptom is given as List. 3 shows. The name of symptom source in this case is the name of FB `ECAT_MN`. The third parameter in the consequent part of the rule in List. 3 indicates the following additional parameters in this part. For communication failure, the solution in the context of our example models is to restart the FB `ECAT_MN`. This action on the FB `ECAT_MN` will reset the EtherCAT state machines of the master and connected slave devices. In this way, new configuration for the communication process can

```
CommunicationFailure(?sym)
^ Has_Symptom_Source(?sym, ?src)
^ Has_SymptomSource_Name(?src, ?srcname)
-> sqwrl:select(?srcname, ActionRequest, 1,
"RESTART")
```

List. 3. The SQWRL query example for generating the reconfiguration action requests in the case of communication failure.



(a)



(b)

FIGURE 12. Information reported by the servo drive and run-time framework during reconfiguration. (a) The states of the servo drive. (b) The logged information of the run-time environment.

be generated and the slaves can resume to be operational. We conduct the experiment of self-reconfiguration in the case of communication failure with the servo drive from Delta Inc., of which the product type is ASDA2-E. The results are demonstrated in Fig. 12. Fig. 12(a) presents the states of the tested servo drive during various stages of the reconfiguration process. When the drive is in the operational state, the current position of the motor will be displayed. When the link is break down, an error code "AL.185" will be reported. After applying the reconfiguration process wehen the link is resumed, the servo can be recovered.

As for the second case, the SQWRL query example is given as List. 4. This action request contains two steps: firstly delete the existing FB with the type of "VELPROFILE", then a new FB instance with the type of "T_VELPROFILE" is created. The modeling environment will accordingly deal with the other related operations, such as delete/create necessary connections for these FBs. We sample the calculated results of these two profiling algorithms, as Fig. 13 shows. The S-shaped profile is the default option in the FB with the

```
ChangeFromProfilingToP2P(?sym)
^ Has_Symptom_Source(?sym, ?src)
^ Has_SymptomSource_Name(?src, ?srcname)
^ FB(?fb) ^ hasTypeOf(?fb, VELPROFILE)
^ Has_FB_ConstValue_Name(?fb, ?fbname)
^ swrlb:stringEqualIgnoreCase(?fbname, ?srcname)
-> sqwrl:select(?fb, ActionRequest, 3, "DELETE",
"CREATE", "T_VELPROFILE")
```

List. 4. The SQWRL query example for generating the reconfiguration action requests in the case of user request on changing velocity profile.

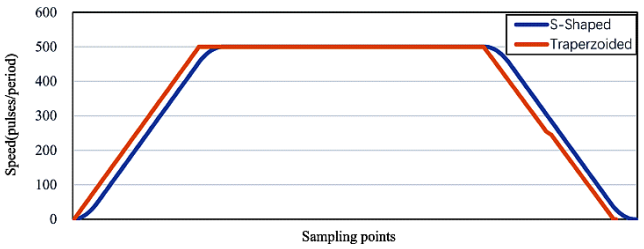


FIGURE 13. The sampled velocity profiles before/after reconfiguration.

type of "VELPROFILE". By replacing the FB with the one which adopts trapezoid profile according to the corresponding action requests, the machine control kernel will be more suitable to deal with simple point-to-point movements with higher computation efficiency.

VI. CONCLUSION

In this paper, we propose a complete set of tools covering the whole process of model-based design and implementation of knowledge-driven self-reconfigurable machine control system, with a bias toward the low level motion control kernel. A domain-specific modeling language compliant to the IEC 61499 standard is put forward, along with the supporting modeling environment. The modeling language and the supporting tool are constructed according to the MIC paradigm. Several extensions of IEC 61499 are introduced to meet the non-functional constraints in this domain, such as the task model for capturing timing requirements and priority-based event connection. The underlying reconfigurable run-time framework is implemented to execute the domain models. Dynamically extendable FB type libraries are constructed through runtime interpreting of FB type definition files. On the basis of the modeling environment and the run-time framework, a knowledge-driven self-adaptive control module is developed. The ontological models for IEC 61499 elements and self-adaptive control are designed as the knowledge. The concepts of symptom and action request are specified to summarize possible situations where reconfigurations are required as well as the concrete steps. The symptoms and action requests can be automatically inferred based on SQWRL queries on the knowledge base. A prototype motion control kernel is developed with our proposed tool sets. Two case studies on self-reconfiguration in the cases of user request and communication failure are demonstrated, which prove the feasibility of our proposal.

Future work in this direction is envisaged as follows:

- Considerations on the influences of the reconfiguration process on the performance of the system;
- Model-based specification of user requirements to facilitate automatic extractions of the symptoms of user requests.

REFERENCES

[1] J. Wan *et al.*, "Software-defined industrial Internet of Things in the context of industry 4.0," *IEEE Sensors J.*, vol. 16, no. 20, pp. 7373–7380, Oct. 2016.

[2] J. Sztipanovits and G. Karsai, "Model-integrated computing," *Computer*, vol. 30, no. 4, pp. 110–111, Apr. 1997.

[3] *Function Blocks*, document IEC 61499-1:2012, International Electrotechnical Commission, Geneva, Switzerland, 2012.

[4] *Programmable Controllers—Part 3: Programming Languages*, document IEC 61131-3:2013, International Electrotechnical Commission, Geneva, Switzerland, 2013.

[5] D. Li, N. Zhou, J. Wan, Z. Zhai, and A. V. Vasilakos, "Towards a model-integrated computing paradigm for reconfigurable motion control system," in *Proc. 14th IEEE Int. Conf. Ind. Inform.*, Jul. 2016, pp. 756–761.

[6] Y. Koren *et al.*, "Reconfigurable manufacturing systems," *CIRP Ann.*, vol. 48, no. 2, pp. 527–540, 1999.

[7] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *J. Intell. Manuf.*, vol. 11, pp. 403–419, Aug. 2000.

[8] G. Pritschow, K.-H. Wurst, C. Kircher, and M. Seyfarth, "Control of reconfigurable machine tools," in *Changeable and Reconfigurable Manufacturing Systems*. London, U.K.: Springer, 2009, pp. 71–100.

[9] S. Feldmann, M. Loskyll, S. Rosch, J. Schlick, D. Zuhlke, and B. Vogel-Heuser, "Increasing agility in engineering and runtime of automated manufacturing systems," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2013, pp. 1303–1308.

[10] Y. Alsafi and V. Vyatkin, "Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing," *Robot. Comput.-Integr. Manuf.*, vol. 26, no. 4, pp. 381–391, 2010.

[11] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan, "Toward self-reconfiguration of manufacturing systems using automation agents," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 1, pp. 52–69, Jan. 2011.

[12] W. Dai, V. N. Dubinin, J. H. Christensen, V. Vyatkin, and X. Guan, "Toward self-manageable and adaptive industrial cyber-physical systems with knowledge-driven autonomic service management," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 725–736, Apr. 2017.

[13] D. Regulin, A. Glaese, S. Feldmann, D. Schütz, and B. Vogel-Heuser, "Enabling flexible automation system hardware: Dynamic reconfiguration of a real-time capable field-bus," in *Proc. IEEE 13th Int. Conf. Ind. Inform. (INDIN)*, Jul. 2015, pp. 1198–1205.

[14] V. Lesi, Z. Jakovljevic, and M. Pajic, "Towards plug-n-play numerical control for reconfigurable manufacturing systems," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–8.

[15] D. Yu, Y. Hu, X. W. Xu, Y. Huang, and S. Du, "An open CNC system based on component technology," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 2, pp. 302–310, Apr. 2009.

[16] S. Wang and K. G. Shin, "Constructing reconfigurable software for machine control systems," *IEEE Trans. Robot. Autom.*, vol. 18, no. 14, pp. 475–486, Aug. 2002.

[17] D. Li, F. Li, X. Huang, Y. Lai, and S. Zheng, "A model based integration framework for computer numerical control system development," *Robot. Comput.-Integr. Manuf.*, vol. 26, no. 4, pp. 333–343, 2010.

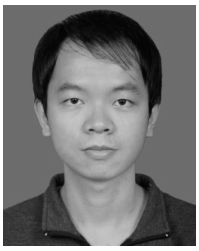
[18] F. Li, J. Wan, P. Zhang, D. Li, D. Zhang, and K. Zhou, "Usage-specific semantic integration for cyber-physical robot systems," *ACM Trans. Embedded Comput. Syst.*, vol. 15, pp. 50-1–50-20, Jul. 2016.

[19] K. Thramboulidis, "Model-integrated mechatronics—Toward a new paradigm in the development of manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 54–61, Feb. 2005.

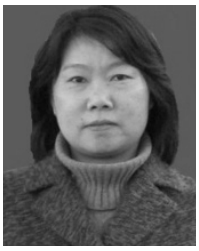
[20] S.-H. Suh, S. K. Kang, D.-H. Chung, and I. Stroud, *Theory and Design of CNC Systems*. London, U.K.: Springer, 2008.

[21] A. Ledeczi *et al.*, "Composing domain-specific design environments," *Computer*, vol. 34, no. 11, pp. 44–51, Nov. 2001.

[22] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proc. IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.



NAN ZHOU received the B.Eng. degree in mechanical engineering and automation from the South China University of Technology, Guangzhou, China, in 2012, where he is currently pursuing the Ph.D. degree with the School of Mechanical and Automotive Engineering. His current research interests include embedded system design theory and methodology, IEC 61499 function blocks, motion control systems, and formal method for industrial cyber-physical systems.



DI LI received the B.Eng. and M.Eng. degrees in aircraft engine and power engineering from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1985 and 1988, and the Ph.D. degree in automatic control theory and application from the South China University of Technology, Guangzhou, China, in 1993. She was a Visiting Scholar with The University of Sydney, Sydney, NSW, Australia, and a Senior Research Fellow with Vanderbilt University, Nashville, TN, USA, and in Japan and Singapore. She is currently a Professor with the School of Mechanical and Automotive Engineering, South China University of Technology. She has directed over 50 research projects, including the National High Technology Research and Development Program of China (863 Program) and the National Natural Science Foundation of China. She has authored or co-authored over 180 scientific papers. Her research interests include embedded systems, computer vision, and cyber-physical systems.

SONG LI received the Ph.D. degree in mechanical engineering and automation from the South China University of Technology, Guangzhou, China, in 2016. His research interests include motion control, computer vision, and cyber-physical systems.

SHIYONG WANG was born in Huoqiu, China, in 1981. He received the B.S. and Ph.D. degrees in mechanical and electrical engineering from the South China University of Technology, Guangzhou, China, in 2010. Since 2010, he has been a Lecturer with the Mechanical and Electrical Engineering Department, South China University of Technology. He has authored over 20 articles and holds five patents. His research interests include smart manufacturing, motion control, and robotics. He was a recipient of the First Prize for Science and Technology Development of Guangdong Province in 2009.

CHENGLIANG LIU was born in Shandong, China, in 1964. He received the B.Eng. degree from the Shandong University of Technology, Shandong, and the M.Eng. and Ph.D. degrees from Southeast University, Nanjing, China, in 1991 and 1998, respectively. He has been invited as a Senior Scholar at the University of Michigan, Ann Arbor, and the University of Wisconsin–Madison, Madison, since 2001. He is currently with the Institute of Mechatronics, Shanghai Jiao Tong University, Shanghai, China. His current interests include mechatronic systems, MEMS/NEMS design, intelligent robot control, web-based remote monitoring techniques, and 3S (GPS, GIS, RS) systems.