IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 3, 2017, accepted November 9, 2017, date of publication November 15, 2017,
date of current version December 22, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2774001

Configurable Platform for Optimal-Setting Control of Grinding Processes

WEI DAI[1,2], (Member, IEEE), GANG HUANG[1], FEI CHU[1], (Member, IEEE), AND TIANYOU CHAI[2], (Fellow, IEEE)

[1]School of Information and Control Engineering, China University of Mining Technology, Xuzhou 221116, China
[2]State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China

Corresponding author: Wei Dai (daiwei_neu@126.com)

This work was supported in part by the Natural Science Foundation of China under Grant 61603393 and Grant 61503384, in part by the Nature Science Foundation of Jiangsu Province under Grant BK20160275 and Grant BK20150199, in part by the Postdoctoral Science Foundation of China under Grant 2015M581885, in part by the Open Project Foundation of State Key Laboratory of Synthetical Automation for Process Industries under Grant PAL-N201706.

ABSTRACT For grinding processes, optimal-setting control (OSC) is becoming a hot topic. However, there is no configurable platform to assist researchers and engineers to design such a controller. This paper proposes a novel software platform named OSC to address this problem. The major superiority is that the platform not only provides a configurable environment by developing a powerful controller design tool and a Petri net model to schedule algorithm modules for parallel computation but also integrates several mainstream intelligent and data-driven algorithms (e.g., case based reasoning, fuzzy logic, and neural network) within a unified framework. The overall framework and key technologies are introduced in detail. Using a hardware-in-the-loop experiment system, the platform is verified and validated through a case of application where an intelligent optimal-setting controller is developed for a classical grinding process.

INDEX TERMS Grinding process, optimal-setting control, configurable platform, Petri net.

I. INTRODUCTION

In the mineral processing industry, the grinding process (GP) is used to reduce the particle size of the ore such that the valuable mineral constituent can be exposed and then recovered in subsequent operations. Its product particle size directly affects the performance of the whole concentrator in terms of the product concentration grade and the production capacity [1]. In addition, as a typical process with high energy consumption and low efficiency, the GP accounts for 45-70% of power consumption and 40-60% of production cost for the whole concentrator. Therefore, the GP is a significant procedure that substantially affects economic profit.

In the past half century, based on standard hardware platforms including distributed control systems (DCS) and supervisory control and data acquisition (SCADA), many popular control technologies, such as multi-loop PID control, multivariable decoupling control, and model predictive control, have been widely used to stabilize the process operation by keeping the outputs of control loops following their set-points [2], [3]. Unfortunately, these control systems are limited in improving grinding quality and production efficiency, as well as in enhancing economic profit. These limitations lie in the fact that "*with respect to the economic performance, the controller performance is most probably not as important as the right selection of the set-points*" [4]. The determination of the optimum set-points for process control is known as optimal-setting control (OSC). Owing to increasing demands on product quality and economic benefit, the OSC issue has attracted much attention of scholars and engineers [5]–[7].

Traditionally, the OSC systems are always deployed on top of the existing process controllers. To develop a suitable OSC system, one always first designs and tests the control methods in a simulation environment provided by some scientific computation or simulation software such as MATLAB and SCILAB. Then, those methods and their user interfaces (UI) must be developed by using generic programming languages (e.g., C++, C#, Visual Basic) according to the specific process controllers. This is a case-by-case development mode, which requires the developer to have knowledge in both the control and software areas. It thus results in a protracted development cycle.

To end this, by integrating some special advanced control or intelligent control technologies, several commercial off-the-shelf (COTS) control software packages, such as Smart

Grind, PROSCON ACT OCS©software, ECS/FuzzyExpert, etc., are developed as design tools to assist people in implementing their OSC system. Although these tools have been successfully applied in many real plants, there are several bottlenecks limiting their extensive application. First, the control behaviors are built into the implementation and hence, are not customizable or not modularized, although the concept of component-based software integration has already been adopted in the software development. Second, they cannot share and reuse the existing software applications or components containing control behaviors. In brief, the functional capability of those tools only involves parameter adjustment of control methods being used. Consequently, it still needs to adopt case-by-case mode to develop the control system when the process dynamic needs a different OSC strategy. This makes the scholars with control knowledge but without software knowledge hard to apply their effective control strategies [8]–[12] into industrial systems.

To separate the control behaviors from software functions at the code level, this paper proposes a configurable platform for OSC, called OSControl. The goal is to enable control engineers or scholars to work independently on the aspect of control systems without software development technologies. This paper presents the overall solution and key technologies. The main innovations and advantages can be illustrated as follows:

1) This platform is investigated and designed especially for the development of GP OSC systems. The platform integrates mainstream algorithms within a unified framework, which makes the platform focused on and expert in finding the optimum setpoints.
2) This platform is a graphic control configuration software that adopts a module-based approach to build controller. Each control algorithm can be encapsulated in a unified control module for reuse.
3) A Python script engine is embedded to extend and customize the control algorithm. In addition, an external interface is provided to call existing external applications (including MATLAB and SCILAB) or components in a type of dynamic link library (DLL).
4) A Petri net model is employed to enable control modules assembling the controller to be scheduled and invoked adaptively in parallel environment.

The above features make the platform elegant and convenient to use. With these efforts, we try to solve the aforementioned problems of existing tools and provide an excellent software platform to design and develop a GP OSC system.

In the paper, the details of OSC approaches integrated by the platform are not discussed. This paper mainly focuses on the design and implementation of the platform framework, which is the greatest contribution of this work. More details of algorithms can be found from our earlier works [3], [8], [12]–[15]. The paper is organized as follows. An overview of this platform is given in Section II. The overall framework and key technologies are introduced in Sections III and IV, respectively. Section V validates and verifies the platform through a case of application in a hardware-in-the-loop experiment system. Finally, conclusions with a future perspective are presented in Section VI.

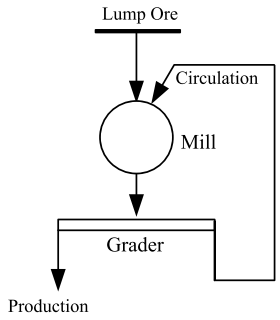


FIGURE 1. Flowsheet of grinding process.

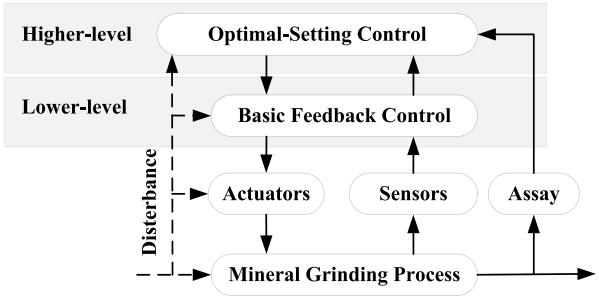


FIGURE 2. Hierarchical control structure of grinding process.

II. GRINDING PROCESS AND CONTROL PROBLEM

In concentration plants, the GP is a prerequisite procedure for many kinds of metal beneficiations. Its raw material is the lump ore generated from the crushing procedure. A classical GP usually consists of a great mill and a grader as shown in Fig. 1. The mill is a metal cylinder rotating around its axis at a fixed speed with heavy media inside. Owing to the combined effect of knocking, chipping, and tumbling caused by the grinding media, the lump ore can be crushed to fine particles. According to different grinding media, the mill is classified as ball mill, rod mill and so on. The grader is a classification unit used to filter powders grinded from the mill and transfer the coarse materials to the mill for re-grinding. The grader usually employs hydrocyclone or spiral classifier.

During the GP operation, there are two important production indices, namely, product particle size (PPS) and grinding production rate (GPR), which define the process quality and efficiency. The control objective is to control the PPS within an optimal range specified by the mineralogical economics while maximizing the GPR. To realize the above objective, the hierarchical control structure shown in Fig. 2 has been widely adopted for years. At the higher-level, using OSC, the optimization is performed on the operating setpoints of the control loops in response to the changes of environmental conditions. At the lower-level, basic feedback control is in charge of guaranteeing the loop tracking performance.

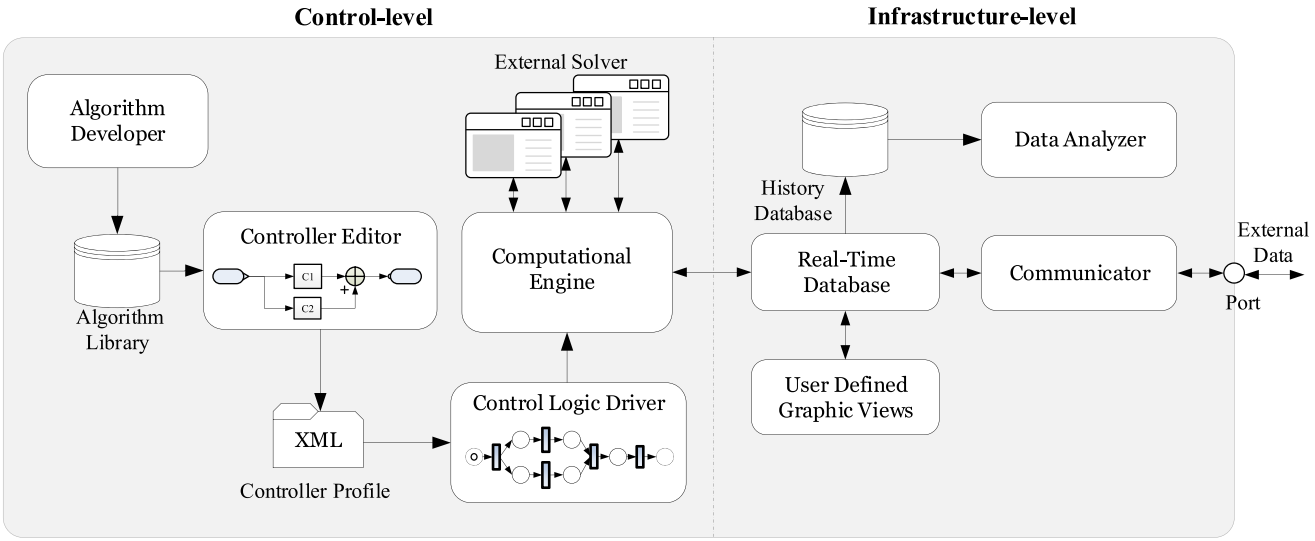


FIGURE 3. Overall solution of software platform for OSC.

As is well known, according to the IEC61131-3 standard [16], a great deal of mature configuration software such as Step 7, Logix5000 and Control Builder are provided by automation system vendors for design and development of basic feedback control system. To the best of our knowledge, however, there is no unified and effective tool for the OSC system. In GP control field, although there exist several COTS packages (e.g., Smart Grind, PROSCON ACT OCS©software, ECS/FuzzyExpert), they have limited ability in control strategy extension and user-defined algorithms according to various industrial process dynamics. Therefore, only a few plants have realized the set-points optimization, and most of them are still under manual operation. However, due to the subjectivity and arbitrariness of humans, it is difficult for the operator to obtain the proper operating point. Real plants, consequently, always operate under a non-optimized economic status and produce with high loss and low quality. The increasingly fierce competition of the world market has inevitably led to an urgent demand for a configurable platform for the GP OSC.

III. OVERALL DESIGN

The developed platform is intended as a tool especially for the algorithm configuration rather than algorithm parameter adjustment of OSC. From an engineering point of view, control algorithms must be designed in a modular way. That is to say, a control method should be made up of several inter-connected sub-modules or sub-control units (such as filter, predictive model, feedback controller and feedforward controller) [7], [8], [12]. Hence, engineers often first develop each sub-control unit one by one and then connect them together to construct a controller. Accordingly, this platform adopts a module-based architecture to build the optimal-setting controller. In this architecture, each control unit is encapsulated into a unified module so that the addition, removal and replacement of modules of controllers can be realized conveniently. Our overall solution is presented in Fig. 3.

From Fig. 3 it can be seen that the platform includes two levels, i.e., infrastructure-level and control-level. In the infrastructure-level, by designing four function components, namely, `Real-Time Database`, `User Defined Graphic Views`, `Data Analyzer`, and `Communicator`, certain essential functions are realized.

1) `Real-Time Database` is similar to a data information pool shared by all components.

2) `User Defined Graphic Views` is used to customize some visual interfaces for monitoring the grinding system operation conveniently.

3) `Data Analyzer` focuses on discovering and evaluating the performance of the controller using statistical techniques.

4) `Communicator` is in charge of accessing external data from communication ports and updating the variables of `Real-Time Database`.

The configurability of software is mainly realized by the control-level, which includes five function components, i.e., `Algorithm Library`, `Algorithm Developer`, `Controller Editor`, `Control Logic Driver`, and `Computational Engine`. Those novel components and their combination make the building of a large scale, complex controller easier.

1) `Algorithm Library` integrates several mainstream control algorithms (e.g., case-based reasoning, fuzzy logic, neural network, rule-based reasoning, and even PID), and encapsulates them into algorithm modules for reuse in the implementation phase.

2) `Algorithm Developer` is provided to customize new algorithm modules and encapsulate them into the `Algorithm Library`.

3) `Controller Editor` provides a graphical configuration environment to build the controller by reusing the algorithm modules from the `Algorithm Library`. In addition, it will generate a profile of the controller in EXtensible Markup Language (XML).

4) `Control Logic Driver` is in charge of scheduling the invoking of a sequence of algorithm modules during runtime according to the controller configuration.

5) `Computational Engine` provides basic solving routines and acts as an agent to call the external solvers specified by the algorithm modules.

The brief view of the work mechanism is as follows:

In the development phase, the special algorithm modules can be built first by using the `Algorithm Developer` when it cannot be found in the `Algorithm Library`. Then, the controller is configured in drag-and-drop type using the `Controller Editor`. After having finished the controller configuration, an XML file containing controller descriptions, algorithm module ID, labels, input/output variables, and the handle of the solving routine will be created. In the running phase, the `Control Logic Driver` will first parse this XML file and determine which of the algorithm modules can be enabled. Then, the platform calls the `Computational Engine` to solve the routine specified by the enabled algorithm module. The solve results will update the `Real-Time Database`, and the `User Defined Graphic Views`. The `Communicator` will then access the updated data and provide OPC, ODBC and DDE interfaces to communicate with various basic feedback control systems. Thus, the control results can be monitored conveniently and transmitted to the basic feedback control systems to drive the control devices.

IV. IMPLEMENTATION OF KEY TECHNOLOGIES

The platform is a large and complicated system involving not only process data monitoring and visualization but also controller development and operation. For facilitating the software development and maintenance, a plug-in framework based on a managed extensibility framework (MEF) [17] is adopted. In such framework, each function component mentioned in Section III acts as a plug-in, and a messaging manager based on a publish-subscribe pattern [18] is developed to solve the coupling among these plug-ins. This section only focuses on showing the implementation of certain key technologies.

A. CONTROLLER EDITOR

Due to the fact that one controller consists of several algorithm modules, the `Controller Editor` should abstract each module to a function block (FB). According to different control functions, three types of FBs, i.e., input FB (IFB), output FB (OFB) and algorithm FB (AFB), are abstracted.

a) IFB: represents the source of whole controller and only generates output data. The purpose of IFB is to produce the initial data to start a single solving process.

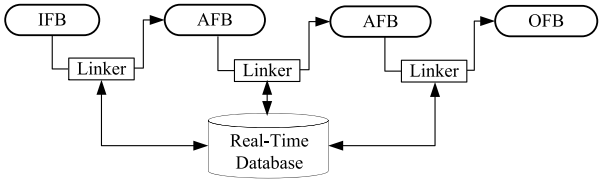


FIGURE 4. Pipeline of data and control algorithms.

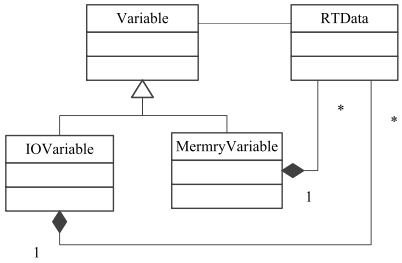


FIGURE 5. Simplified class diagram of Real-Time Database.

b) AFB: represents control behavior, and it receives input data and generates output data. Each control algorithm module can be expressed as an AFB.

c) OFB: represents the end of whole controller and only receives the final results. The purpose of OFB is to update the control commands and end a single solving process.

To construct the whole controller, an abstract class `Linker` depicted by a line with arrow is employed to define the connectivity of data among the FBs and receive the solving result of each AFB at runtime. Consequently, a series of FBs can be connected by pipelines as shown in Fig. 4. To expose data to other function components, the `Linker` connects with the `Real-Time Database` and encapsulates the operations of data exchange between the FBs' internal data and the `Real-Time Database`.

As seen from Fig. 4, real-time database, function block and linker are very important for the `Controller Editor`.

1) REAL-TIME DATABASE

`RTData` abstracts the properties and methods of the `Real-Time Database`, and its class diagram in unified modeling language (UML) [19] can be found in Fig. 5. Owing to the existence of a great deal of system variables containing data properties, such as data ID, name, type and description, the platform employs a `Variable` class to abstract the properties and methods of system variables. In the `Real-Time Database`, the variables can be divided into two types. One type represents the variable used only for algorithm computation, and the other represents the variable used for communicating with external applications. According to the above two types of data, two concrete subclasses of the `Variable` class, namely, `MermryVarible` and `IOVariable`, are developed. In addition, to guarantee the efficiency of search, insertion, and deletion operations, a

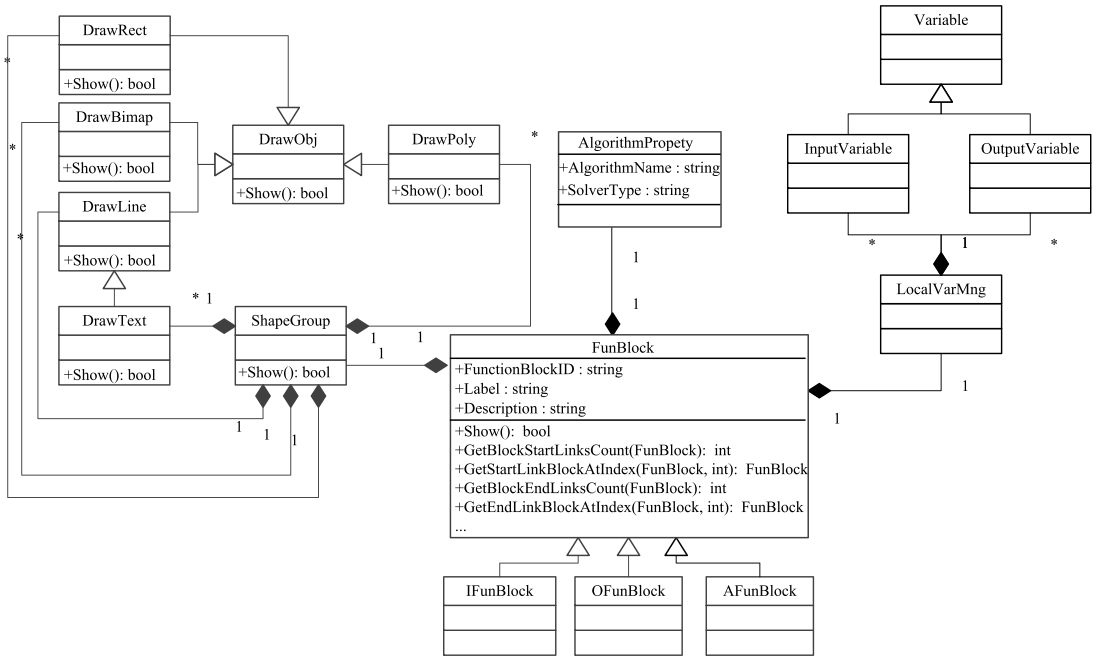


FIGURE 6. Simplified class diagram of function block.

red-black tree is employed to implement the `RTData`, and the data of each tree node is modeled as an instance of `Variable`.

2) FUNCTION BLOCK

`FunBlock` is one of the kernel classes, and it abstracts the properties and methods of FBs. The model structure of FunBlock is demonstrated in Fig. 6. `FunBlock` has three class members, i.e., `ShapeGroup`, `AlgorithmPropety`, and `LocalVarMng`. The `ShapeGroup` provides image data to define algorithm features showing in the `Controller Editor`. The `LocalVarMng` specifies the inner input and output parameters of algorithms, and provides unified access interfaces. The `AlgorithmPropety` provides the properties of algorithms, i.e., *AlgorithmName* and *SolverType*. *AlgorithmName* acts as a handle of the algorithm routine. Using this handle, the platform can determine which one of the algorithm modules existing in the `Algorithm Library` should be reused. The *SolverType* specifies which solver should be employed to solve the algorithm routine at runtime. It is noteworthy that the `Algorithm Library` has a unique assigned name for the different algorithm modules. Hence, two instances of `FunBlock` can be distinguished easily when they use the same algorithm. The reason is that the *FunctionBlockID*, *Label* and *Description* properties express the uniqueness of each instance.

To access the linkers connected, `FunBlock` provides four member functions. *GetBlockStartLinksCount*(*FunBlock*) and *GetStartLinkBlockAtIndex*(*FunBlock, int*) are called successively to find the linkers that denote the target of the dataflow; meanwhile, *GetBlockEndLinksCount*(*FunBlock*) and *GetEndLinkBlockAtIndex*(*FunBlock, int*) are called successively to find the linkers that denote the source of dataflow.

According to the different types of FBs, we construct three concrete subclasses: `AFunBlock`, `IFunBlock` and `OFunBlock`. `IFunBlock` and `OFunBlock` are designed with the purpose of the IFB and OFB, while `AFunBlock` is the main control algorithm class whose instance implements actual control behaviors.

Presently, the platform has implemented several basic math algorithms, and classical control algorithms. New algorithms can be created in the `Algorithm Developer`, which provides an algorithm script editor to develop algorithms by using an open-source implementation of the Python programming language for the .NET Framework, namely, IronPython. Additionally, two open source libraries for scientific computing, namely, NumPy and SciPym, are integrated in the `Algorithm Developer`, which results in excellent ability of the platform in scientific computing. Furthermore, owing to the fact that the MATLAB is a popular software in process control filed, a MATLAB script technology is also supported in this platform. Besides, the `Algorithm Developer` can directly load the dynamic link library (DLL) files that have encapsulated the algorithm routines by using generic programming languages (such as C++ and C#).

3) LINKER

`Linker` is another kernel class, and it provides the direction of dataflow according to the two properties of *startFB* and *endFB*. The model structure of `Linker` is shown in Fig. 7. Its class members include `DrawLinker`, `VarLink` and

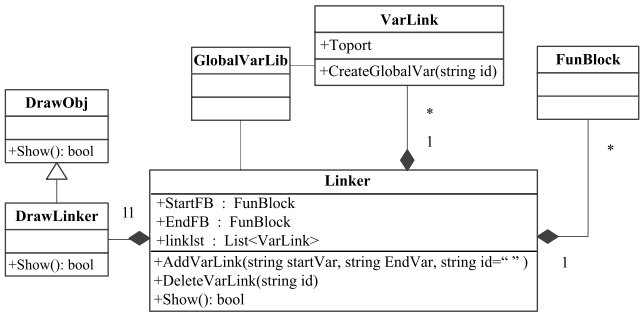


FIGURE 7. Simplified class diagram of Linker.

`FunBlock`. `DrawLinker` defines the image data shown in the `Controller Editor` as `ShapeGroup` does. `VarLink` provides the information of the dataflow, and its instance contains a variable coming from the start block and a variable coming from the end block. `Linker` maintains a list of `VarLinks` and provides the interfaces for adding and removing `VarLink` to and from the list, respectively.

B. PETRI NET MODEL FOR CONTROLLER

When the controller is more complicated than the one shown in Fig. 4, it is necessary to employ another mathematical model to reflect the behavior of the controller. Petri net, as graphical and mathematical tools, provides a uniform environment for modeling, formal analysis, and design of industry control systems. In addition, the Petri net can be used to perform a formal check of the properties related to the behavior of the underlying system, e.g., precedence relations among events, concurrent operations, and appropriate synchronization. In view of the features of Petri net, a simple Petri net defined in **Definition 1** is adopted for modeling the controller.

Definition 1 (Petri Net Structure): A Petri net structure is a tuple

$$NS = (P, T, F, M_0)$$

where

P	is a finite set of places p;
T	is a finite set of transitions t;
$F \subseteq (P \times T) \cup (T \times P)$	is a set of flow arcs f;
M_0: $P\{1, 2, \ldots\}$	is an initial marker.

In this platform, each element defined under the Petri net model has a special meaning explained in **Definition 2**.

Definition 2:

- p denotes an available data resource that can be used in algorithm calculations;
- t denotes an FB, including IFB, OFB and AFB;
- f denotes the dataflow direction defined in Linkers.

The normal control approach in industry systems is that signals are free of backward effects on the generalized plant, as shown in Fig. 8 (a). That means the behavior of the controller system which emits a control signal is not influenced by the presence of measurement. In the Petri net model for controllers, as shown in Fig. 8 (b), some places are added at signal transmission channels (input and output channels). And a token will be put on the input places when the measurement is available, then stay here until it is consumed.

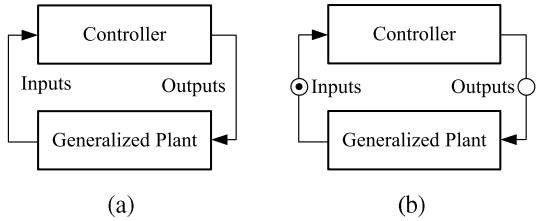


FIGURE 8. Control loops: (a) Normal control loop. (b) Control loop with Petri net model.

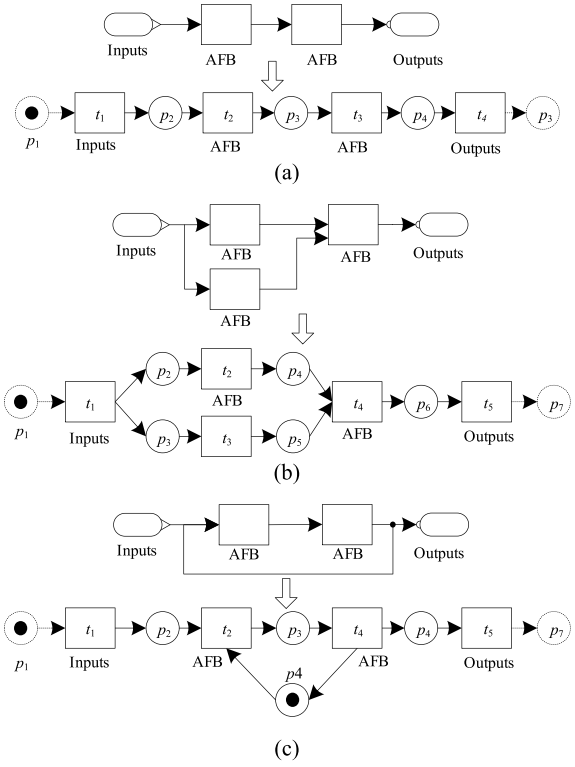


FIGURE 9. Basic mapping relationship between graphic controller and Petri net model: (a) Serial structure. (b) Parallel structure. (c) Feedback structure.

In the platform, the control strategy must be mapped into a Petri net model and then be checked whether it is available before calling the computational engine. Specific steps of mapping an optimal-setting controller into a Petri net model are as follows: First, some virtual places and virtual arcs are added in order to complete an entire Petri net model. Second, IFB and OFB will be separately mapping into input and output transitions t. In addition, there must be only one output place, while the input place can be instantiated more than one meet the demands of a multi-sampling control system. Then, AFB and linker will be separately mapped into

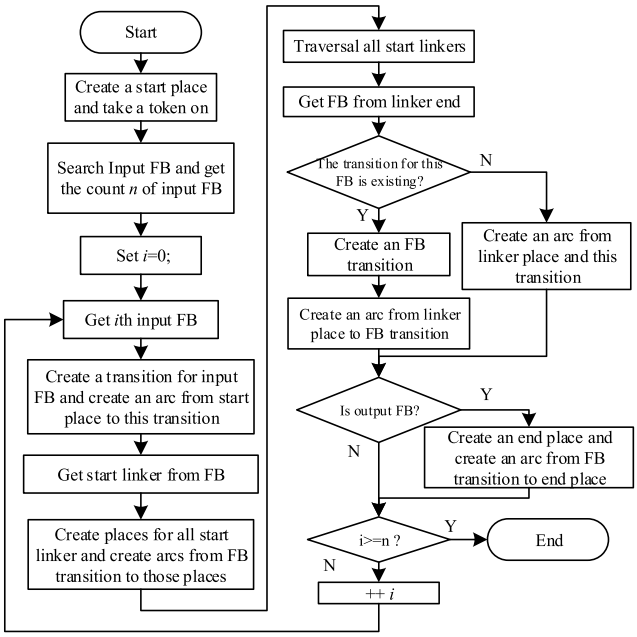


FIGURE 10. Flow chart of Petri net model mapping.

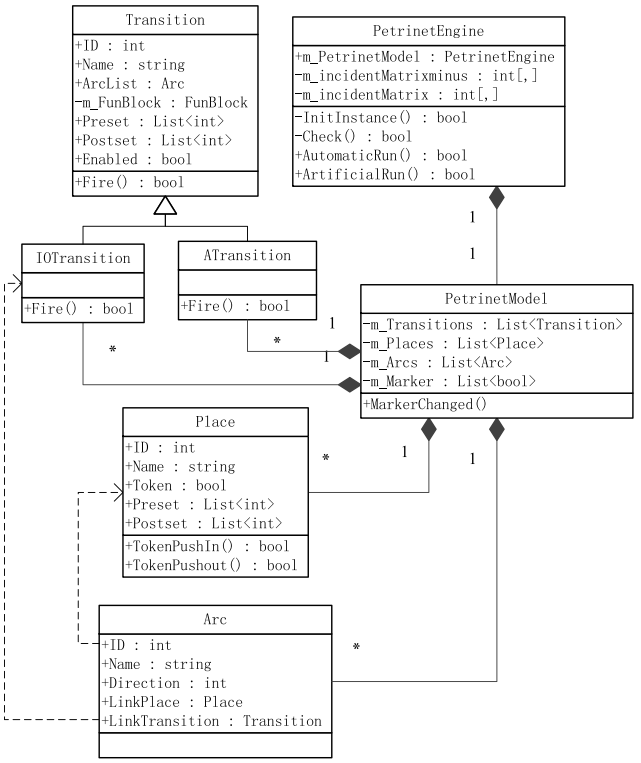


FIGURE 11. Construct of Petri net controller.

transition t and place p. Some arcs are added between transition and places, and the direction of the arcs are in accord with the corresponding linkers. Fig. 9 reveals the way of basic mapping. Fig. 10 presents the flow chart of mapping from the graphic controller model to the Petri net model.

In such Petri net model, the places represent pre-conditions or post-conditions, which denote the availability of data resources. Output place represents the computing result of the overall controller. By using the Petri net model, the enabled and execution states of FBs can be easily separated. Distribution changing of tokens on places will reflect the execution of FBs. For instance, in Fig. 9 (a), moving the token to place p_3 from p_2 means that the algorithm in transition t_2 has been executed and the output data can be used in the next enabled transition t_3.

A Petri net engine is designed to generate an incident matrix to check the net properties and to provide essential feedback information for users when the net anomalies or has error. To distinguish the enabling and implementation of activity FBs, the FB trigger events have been converted to two trigger conditions, i.e., a manual trigger condition and an automatic trigger condition. In the automatic trigger condition, the FB will be fired immediately when the FB is enabling, while the FB will be fired until a mouse event occurs in the manual trigger condition. When the controller is running, the Petri net engine will take charge of the condition of FBs under the Petri net model, which means that the engine will control the token migration and constantly update the net marker. When there are no new migrations to be dealt with, or no token will generate, the engine will reset the initial marker in order to prepare for the next control period.

As shown in Fig. 11, the Petri net model is abstracted to a `PetrinetModel` class, which maintains three class members of `Transition`, `Place` and `Arc`. `Transition` abstracts the properties and methods of a transition. `IOTransition` and `ATransition` are concrete classes from `Transition` to represent IFB/ OFB transition and AFB transition, respectively. The virtual function *Fire*() is called to activate transition. The two concrete classes of `Transition` implement the virtual function *Fire*() to execute the algorithm or specified function. `PetrinetEngine` maintains an instance of `PetrinetModel`, and provides the interfaces for analyzing and operating the model. The switch of the trigger condition is controlled by two member functions, *AutomaticRun*() and *ArtificialRun*(). The properties, *m_incidentMatrix* and *m_incidentMatrixminus* provide the information of incidence matrix for analyzing and controlling the dynamic behavior of Petri net. During the running, when a `Transition` instance calls the member function *Fire*() and returns true, its preset will call the *TockenPushOut*() and set the property *Token* false, while its postset will call the *TockenPushIn*() and set the property *Token* true; then, the places in postset call the *MarkerChanged* function to change property *m_Marker* that represents the current net marker. `PetrinetEngine` is a listener for *m_Marker*, and it immediately updates the property *Enabled* of all `Transaction` instances after the *m_Marker* is changed. Only when property *Enabled* is true can corresponding `Transaction` instances call

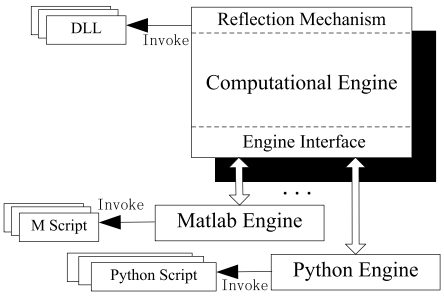


FIGURE 12. Diagram of computational framework.

member function *Fire*() later. In `IOTransition` class, the *Fire*() will read or write variables through the communication interface; in the `ATransition` class, the *Fire*() will parse the algorithm through the interfaces provided by the `Computational Engine`.

C. COMPUTATIONAL ENGINE

`Computational Engine` provides a uniform computational framework to produce a scalable computational interface for dynamically loading different types of algorithms. The framework is illustrated in Fig. 12. From Fig. 12, we can see that the platform provides support for not only type of DLL but also types of Python script and MATLAB script by calling the Python engine and the MATLAB engine. The computational framework has used the abstract factory design pattern to manage computational interface in a flexible and graceful manner. As shown in Fig. 13, `Calculator` declares an interface for the engine object. `MatlabEngine`, `DLLCalculator` and `PythonEngine` define distinct engine objects that should be created by `EngineFactory` responding to the different algorithm types. `ComputationalEngine` is a client, where function *CreateEngine*() can get an engine object through `EngineFactory`; function *Calculate*() can parse the algorithm provided by the corresponding FB.

D. COMMUNICATION INTERFACE

According to technical requirements, the platform provides two interfaces for external data communication. The first interface is developed based on OLE for process control (OPC) technology, which allows the platform to communicate with local basic feedback control systems such as PLC/DCS systems. The other interface is developed based on open database connectivity (ODBC) technology, which guarantees the platform can communicate with other software systems, such as data processing/analyzing system. There is a distributed computing framework when this platform works together with plant-wide optimization applications. Taking into account the heterogeneity of different systems, a message oriented middleware interface is developed to achieve the data exchange by using message queuing technology and to ensure the reliability and security of data transmission.

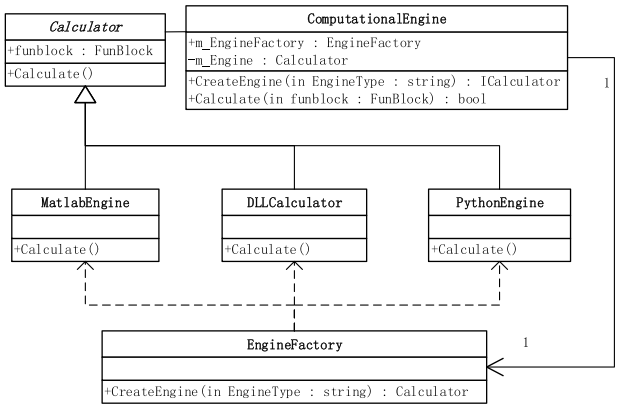


FIGURE 13. Simplified class diagram of Computational Engine.

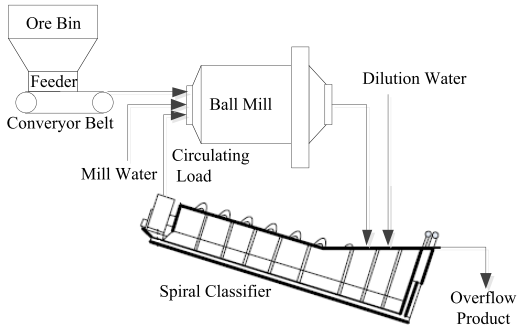


FIGURE 14. Schematic diagram of mineral grinding circuit.

Currently, the OPC technology has already been widely used in industry and performs excellently in the LAN (Local Area Network). Thus, this technology could be regarded as the bridge of the data between the low-level PLC/DCS control systems and high-level OSC systems. When using the OPC technology, there is no need to develop different data communication channels for distinct PLC/DCS systems. The OPC interface of platform is realized by using Kepware's OPC development kit. For achieving the ability of data interaction with other data processing and analyzing applications, relational database implementation models such as Oracle, SQL Server, Sybase, DB2, and MySQL in data communication between different systems is a very popular solution. Therefore, the ODBC interface is employed here to provide access to different database systems such that the operation data can share with other applications.

E. PARALLEL COMPUTATION

In response to demands for higher performance in parallel computation, multi-thread technology is used here. To make the manager thread more efficient, the Task Parallel Library (TPL) that takes an AFB computation process as a task is employed. The TPL cooperates multi-thread by using the task scheduler and automatically distributes tasks across the computer's available CPUs quickly and painlessly.

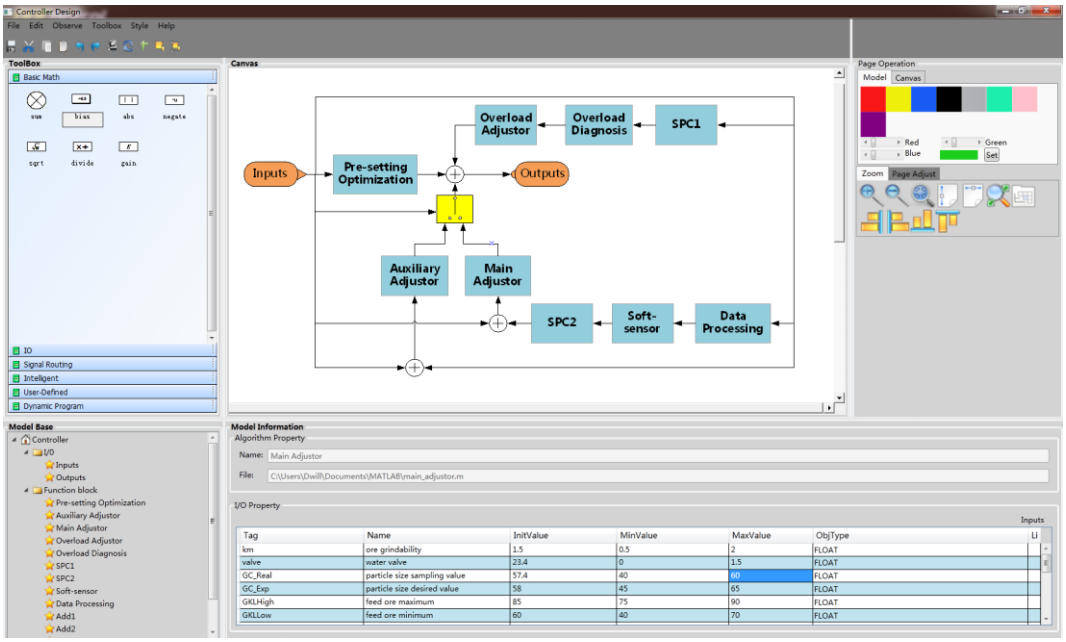


FIGURE 15. Snapshot of the controller editor.

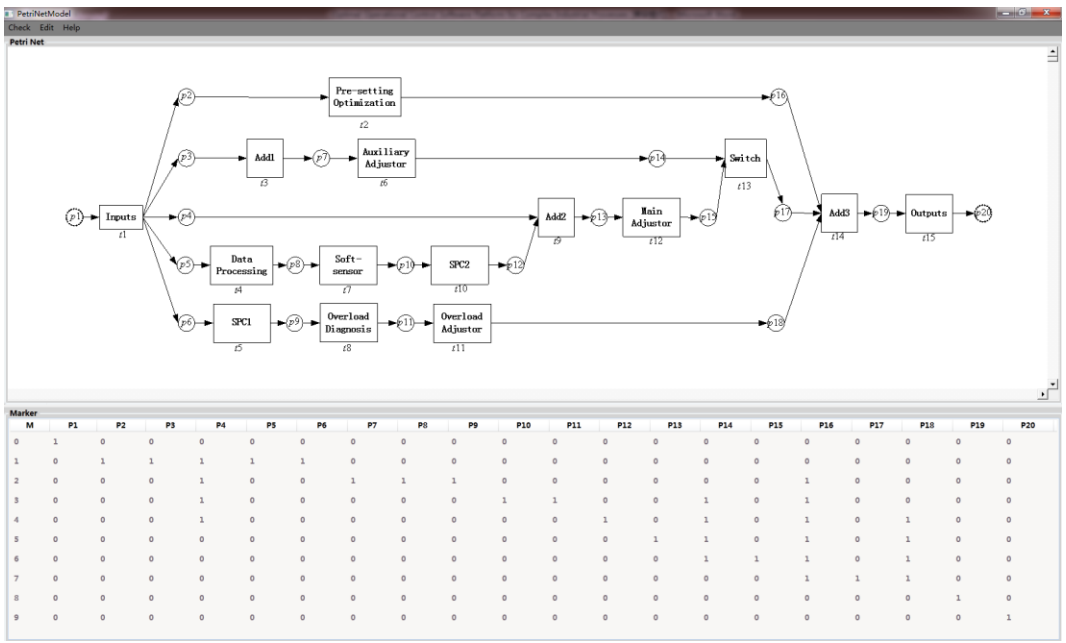


FIGURE 16. Snapshot of controller model based on Petri net.

V. EXPERIMENT EXAMPLES

In this section, we discuss an application for a classical closed-loop GP with a ball mill and a spiral classifier, as shown in Fig. 14.

In this paper, an intelligence-based OSC method that consists of a control loop pre-setting optimization module, an artificial neural network (ANN) based soft-sensor module, two fuzzy logic based dynamic adjustors and an expert-based overload diagnosis and adjustment module are employed. In keeping with the focus of this paper, the details of the designed controller are not shown here but can be found in [20]. Using the platform, it is convenient to design and implement the controller mentioned above. Fig. 15 shows a snapshot of the controller editor. The snapshot shows on the left-hand side the algorithm toolbox and the construction tree of the controller. In the main window, a canvas is used to configure the controller, and the variable and shape properties of FB are shown on the bottom and right side, respectively.

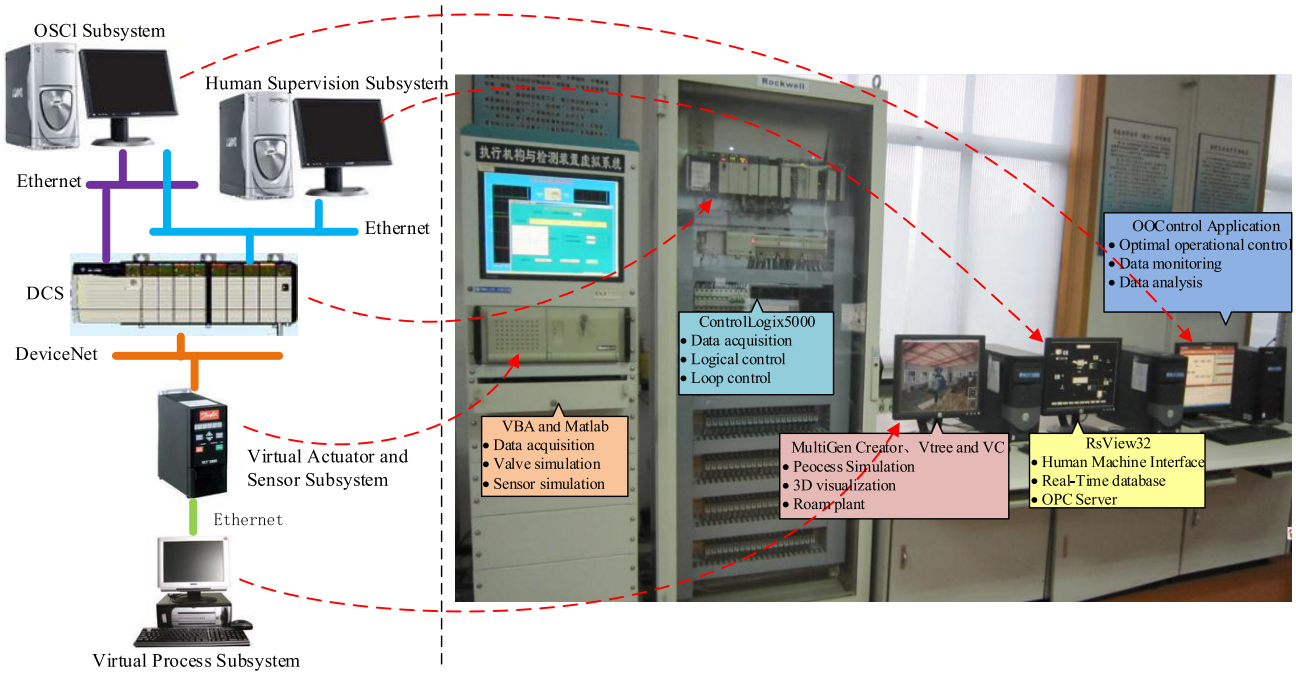


FIGURE 17. Structure of hardware-in-the-loop OSC experiment system.

Fig. 16 shows the Petri net mapping from the graphical controller and all markings.

This application was tested at a hardware-in-the-loop (HIL) experiment system [21] for grinding processes. Fig. 17 shows the actual HIL experiment system and its architecture. From Fig. 17, it can be seen that the HIL experiment system consists of an OSC subsystem, a human supervision subsystem and a DCS subsystem, a virtual actuator and sensors subsystem, and a virtual process subsystem to be controlled. These subsystems are linked via industrial Ethernet and Devicenet. In this experiment system, a widely used DCS, the Controllogix 5000 control system, is applied to realize the lower level basic feedback control. By utilizing RSView32 software, a human–computer interaction monitoring platform is developed in the human supervision subsystem. The hardware components of the virtual actuator and sensors subsystem includes an industry computer equipped with several data acquisition cards and industrial terminal boards from AdvanTech. The user interface is designed based on RSView32 which supports visual VBA script to collect data from the data acquisition cards. In the virtual process subsystem, simulation is implemented by using the MATLAB engine. Additionally, MultiGen Creator, VTree and Visual C++ are employed to display the operational devices and to realize data exchange with the virtual actuator and sensors subsystem.

The OSC subsystem is run on a Windows-PC with an Intel Pentium 4 2.8 GHz processor and 1 GB physical memory. The experiment system has spent 600 minutes on a test, and the OSC subsystem has totally calculated 60 times. Fig. 18 shows the main operation interface displaying the autoregulation

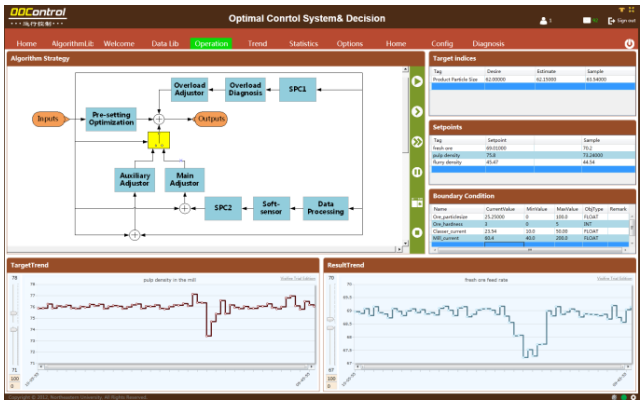


FIGURE 18. Snapshot of main operation interface.

curves of the set points. Fig. 19 shows the control effect and statistical analysis of operational indexes. The operation results show that OSC subsystem enhances the average of PPS from 59.5% $<$ 200 mesh to 62.12% $<$ 200 mesh, and the root mean square error (RMSE) of PPS decreases to 0.8764. In this experiment study, approximately 90.17% of PPS are fully qualified for production, while approximately 6.61% and approximately 3.22% of PPS exceed the maximum and are less than the minimum, respectively. The estimated error and control error in Fig. 19 can also show the control effect from another point of view.

Although this application is an HIL simulation, we believe these results could hold promise that a complete industrial application on this software platform can also work effectively.

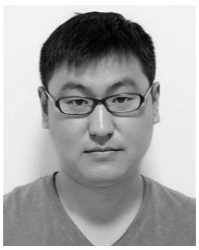
FIGURE 19. Snapshot of operational index statistics interface.

VI. CONCLUSION AND FUTURE PERSPECTIVE

This paper presents a configurable platform called OSControl, which offers a graphical tool for the algorithm designers and engineers to handily design and develop their optimal-setting controllers. After the design of the controller, the software can be automatically modeled by means of the Petri net engine and checked whether the controller is all right. Thus, OSControl can ensure the designed controller fulfill the expected properties before it is implemented by computational engine. By taking a classical closed-loop grinding process as a benchmark, the platform is tested at a hardware-in-the-loop experiment system. The experimental results have clearly shown that the proposed platform can provide a new graphical tool to aid users in easily and rapidly designing and implementing optimal-setting controllers for the GPs. Meanwhile, the Petri net model for optimal-setting controllers is shown for the user to analyze the controller performance. Although the proposed Petri net model shows similarities to a normal simple Petri net, each element in the Petri net has special meanings. We come to the conclusion that such Petri net model are suited for controller verification. The current version of the platform does not allow subnets (hierarchical extension AFB). One of the projected extensions of the controller editor is to allow the reuse of subnets. This will also permit the verification of these subnets before they are used in the controller editor. We believe the prospects of this platform will be very broad in future.

REFERENCES

[1] D. H. Wei and I. K. Craig, "Grinding mill circuits—A survey of control and economic concerns," *Int. J. Mineral Process.*, vol. 90, nos. 1–4, pp. 56–66, Feb. 2009.
[2] A. Pomerleau, D. Hodouin, A. Desbiens, and É. Gagnon, "A survey of grinding circuit control methods: from decentralized PID controllers to multivariable predictive controllers," *Powder Technol.*, vol. 108, nos. 2–3, pp. 103–115, Mar. 2000.
[3] P. Zhou, W. Dai, and T.-Y. Chai, "Multivariable disturbance observer based advanced feedback control design and its application to a grinding circuit," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 4, pp. 1474–1485, Jul. 2014.
[4] D. Hodouin, "Methods for automatic control, observation, and optimization in mineral processing plants," *J. Process Control*, vol. 21, no. 2, pp. 211–225, Feb. 2011.
[5] C. Bouché, C. Brandt, A. Broussaud, and W. I. van Drunick, "Advanced control of gold ore grinding plants in South Africa," *Mineral Eng.*, vol. 18, no. 8, pp. 866–876, Jul. 2005.
[6] X. S. Chen, Q. Li, and S. M. Fei, "Supervisory expert control for ball mill grinding circuits," *Expert Syst. Appl.*, vol. 34, no. 3, pp. 1877–1885, Apr. 2008.
[7] P. Zhou, T. Y. Chai, and J. Sun, "Intelligence-based supervisory control for optimal operation of a DCS-controlled grinding system," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 1, pp. 162–175, Jan. 2013.
[8] W. Dai, T. Chai, and S. X. Yang, "Data-driven optimization control for safety operation of hematite grinding process," *IEEE Trans. Ind. Electron.*, vol. 62, no. 5, pp. 2930–2941, May 2015.
[9] H.-X. Li and S. Guan, "Hybrid intelligent control strategy. Supervising a DCS-controlled batch process," *IEEE Control Syst.*, vol. 21, no. 3, pp. 36–48, Jun. 2001.
[10] Z. J. Wang, Q. D. Wu, and T. Y. Chai, "Optimal-setting control for complicated industrial processes and its application study," *Control Eng. Pract.*, vol. 12, no. 1, pp. 65–74, 2004.
[11] R. Lestage, A. Pomerleau, and D. Hodouin, "Constrained real-time optimization of a grinding circuit using steady-state linear programming supervisory control," *Powder Technol.*, vol. 124, no. 3, pp. 254–263, 2002.
[12] T.-Y. Chai, "Optimal operational control for complex industrial processes," in *Proc. 8th IFAC Int. Symp. Adv. Control Chem. Process.*, Singapore, 2012, pp. 722–731.
[13] W. Dai and T.-Y. Chai, "Data-driven optimal operational control of complex grinding process," *Acta Autom. Sinica*, vol. 40, no. 9, pp. 2005–2014, Sep. 2014.
[14] X. L. Lu, B. Kiumarsi, T.-Y. Chai, and F. L. Lewis, "Data-driven optimal control of operational indices for a class of industrial processes," *IET Control Theory Appl.*, vol. 10, no. 12, pp. 1348–1356, Aug. 2016.
[15] D. Zhao, T.-Y. Chai, H. Wang, and J. Fu, "Hybrid intelligent control for regrinding process in hematite beneficiation," *Control Eng. Pract.*, vol. 22, no. 1, pp. 217–230, Jan. 2014.
[16] *Programmable Controllers—Part 3: Programming Languages*, 2nd ed., International Standard IEC 61131-3, 2003.
[17] C. Nagel, B. Evjen, J. Glynn, K. Watson, and M. Skinner, *Professional C# 4.0 and .NET 4*, 1st ed. Birmingham, U.K.: Wrox Press Ltd., 2010.
[18] P. T. Eugster, P. A. Felber, and R. Guerraoui, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.
[19] K. Siau and Q. Cao, "Unified modeling language: A complexity analysis," *J. Database Manag.*, vol. 12, no. 1, pp. 26–34, 2001.
[20] P. Zhou, T. Chai, and H. Wang, "Intelligent optimal-setting control for grinding circuits of mineral processing process," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 4, pp. 730–743, Oct. 2009.
[21] W. Dai, P. Zhou, D. Y. Zhao, and T. Y. Chai, "Hardware-in-the-loop simulation platform for supervisory control of mineral grinding process," *Powder Technol.*, vol. 288, pp. 422–434, Jan. 2016.



WEI DAI (M'16) received the M.S. and Ph.D. degrees in control theory and control engineering from Northeastern University, Shenyang, China, in 2009 and 2015, respectively. From 2013 to 2015, he was a Teaching Assistant with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University. He is currently with the China University of Mining and Technology, Xuzhou, China. He received the honorary title of the Young Backbone Teacher of the China University of Mining and Technology in 2017.

His current research interests include modeling, optimization and control of complex system, data mining and machine learning.

GANG HUANG received the B.S. degree in electrical engineering and automation from the Xuhai College, China University of Mining and Technology, Xuzhou, China, in 2016, where he is currently pursuing the M.S degree. His current research includes machine learning, data-driven optimal-setting control, and software technology.

FEI CHU (M'17) received the B.Sc. degree in electrical and automation from Qingdao University, China, in 2007, the M.Sc. and Ph.D. degrees in control theory and control engineering from Northeastern University, China, in 2009 and 2014, respectively. He is currently a Post-Doctoral Researcher and a Lecturer with the China University of Mining and Technology, Xuzhou, China. He has authored over 20 papers in peer-reviewed international journals and conferences. His current research interests include modeling and optimization of complex industrial process, statistical process monitoring, and optimality assessment.

TIANYOU CHAI (M'90–SM'97–F'08) received the Ph.D. degree from Northeastern University, Shenyang, China, in 1985.

He is the Founder and the Director of the State Key Laboratory of Synthetical Automation for Process Industries and the National Engineering and Technology Research Center of Metallurgical Automation. He is also the Director of the Department of Information and Science, National Natural Science Foundation, China. He has authored three monographs and over 120 peer-reviewed international journal papers. His current research interests include adaptive control and intelligent decoupling control.

Prof. Chai received three prestigious awards of the National Science and Technology Progress, the 2002 Technological Science Progress Award from the Ho Leung Ho Lee Foundation, the 2007 Industry Award for Excellence in Transitional Control Research from the IEEE Control Systems Society, and the 2010 Yang Jia-Chi Science and Technology Award from the Chinese Association of Automation. He is a member of the Chinese Academy of Engineering, an Academician of the International Eurasian Academy of Sciences, and a fellow of the IFAC.