http://dx.doi.org/10.7735/ksmte.2014.23.1.038 ISSN 2233-6036

New Modularization Method to Design Supervisory Control of Automated Laboratory Systems

Taeyoung Jung[a*]

자동화 시스템의 관리제어 설계를 위한 새로운 모듈화 기법

정태영[a*]

[a] *Shinsung University, Dangin-city Chungnam, Republic of Korea*

ARTICLE INFO

Article history:
Received 27 August 2013
Revised 13 November 2013
Accepted 21 January 2014

Keywords:
Discrete Event Systems (DES)
Supervisory Control Theory (SCT)
state explosion
AHS system

ABSTRACT

This paper describes the application of discrete event systems theory to the design of an automated laboratory system. Current automated laboratory systems typically consist of several interacting processes that must be carefully sequenced to avoid any possible process conflicts. Discrete Event Systems (DES) theory and Supervisory Control Theory (SCT) can be applied together as effective methods of modeling the system dynamics and designing supervisory controllers to precisely sequence the many processes that such systems might involve. Classical approaches to supervisory controller design tend to result in complex controller structures that are difficult to implement, maintain, and upgrade. In this paper, a new approach to designing supervisory controllers for automated laboratory systems is introduced. This new approach uses a modular controller structure that is easier to implement, maintain, and upgrade,and deals with "state explosion" issues in a novel and efficient way.

1. Introduction

A recent trend in the design of laboratory systems has been to construct them from several interacting and interconnected processes that are primarily guided by computers and equipment controllers. In designing automated laboratory systems for controling several processes, it is important to consider how the control of a sequence of process can be coordinated to achieve an intended purpose. Traditional system control theory has been concerned with systems of continuous variables modeled by differences or differential equations. Discrete event systems (DES) theory, on the other hand, involves the study of systems that consist of logical and symbolic states rather than numeric states[13]. Supervisory Control Theory (SCT) was conceived and developed by P.J. Ramadge and W. M. Wonham[1-3]. Discrete Event Systems (DES) can be applied to the automated laboratory systems to satisfy main controller requirements for their modeling, modification, and improvement.

Many applications for implementing real systems by using the ability of Supervisory Control Theory (SCT) to control the sequence of processes in DES have been proposed in the

* Corresponding author. Tel.: +82-41-350-1534
Fax: +82-41-350-1534
E-mail address: tyjung@shinsung.ac.kr (Taeyoung Jung).

literature[4,5,7,8,11,12].

This paper proposed a main controller based on a DES using SCT in order to address situations in which, a main is requested to not only control each process separately but also to control sequencing of processes without conflict.

The design complexity of such a controller, however, could make difficult to implement and then alter or modify in the future. In addition, any automata designed to model such system processes must be combined in parallel if they are to accurately model their concurrent operation.

Parallel automata composition is a method for combining automata in order to describe the behavior resulting from their simultaneous running; however, implementing the state increment in such parallel composition called the "state explosion", in which the number of the states produced by parallel composition increase exponentially. In such cases, implementation of the automaton resulting from parallel composition can be very difficult, impractical or even impossible.

DES implementation may incur state explosion by combining automata that model a system and a controller in parallel and issues with complexity and state explosion are the main reasons why there have been so few studies to date on the management of DES applications. This paper proposes a new modular structure for the convenient modification and expansion of systems that uses a new approach for reducing exponential state growth and managing complexity.

2. New modular control structure of DES implementation

2.1 Introduction of modular control structure

In order to reduce and manage complexity and state explosion, this paper suggests a new modular design using multiple threads. Based on the work of P. J. Ramadge and W. M. Wonham, who proposed the use of modular supervisory control to reduce computational complexity[2,6,16], the present work presents a modular supervisory control method in which control tasks are separated into smaller subtasks for which subsupervisors are then designed. In order to avoid state explosion that would arise from combining the modular automata into a monolithic supervisor, a decentralized supervisory control method is applied[15,16].

Previous researchers[9,10] have tried to apply DES with modular supervisory control to real systems, and even though the decentralized modular approach has advantages in term of the modification, analysis and maintenance of a system, it has its disadvantages as well.

An individual subsupervisor can observe and control only a part of an entire system. As each subsupervisor is isolated from other subsupervisors and works independently on the basis of local information, the functioning of these may cause conflict at a system and lead to overall system blocking. Thus, this paper proposes a modularized supervisory controller to obtain benefits such as modification and expansion convenience. In this manner, modules can be combined into a centralized controller to avoid the above mentioned problems with decentralized controllers.

In this paper, a method for designing a centralized controller without state explosion is presented as follows.

2.2 New modular design

A schematic diagram of our new-approach modular control structure is shown in Fig. 1. In this system, entire system tasks are divided into several processes based on purpose; for instance, a controller is separated into a high-level and a low-level controller as shown in Fig. 1.

The low-level automata comprising the low-level controller describe the operation sequence of physical hardware and each low-level automaton represents a model of the low-level dynamics of the associated process and observes low-level events of the process. In other words, the low-level automata control sequences of events in the low-level process in order to manage corresponding operations such as the machine's point-to-point operations. Instead of being combined together in parallel, low-level automaton is connected in parallel to an interactive component as shown in Fig. 1.

The high-level controller is composed of high-level automata that connect with the low-level controller through an interactive component. Each high-level automaton observes events occurring within the low-level controller and control sequencing of its operations. As an example, high-level automata managing low-level automata that describe the motions of several machines will decide which machine will start first, then second, then third, etc. The high-level automata

are not combined in parallel either, like the low-level automata, they are each connected in parallel with a single component as shown in Fig. 1.

The interactive component communicates with a graphical user interface (GUI) with which a user can send commands to a controller through an interactive component. Commands sent from the GUI are altered so that they are recognizable by the controllers. While in a decentralized control system, each subsupervisor separately controls its corresponding target, the new modular system proposed here is designed for eventually in a centralized control system, and, therefore, its high-level automata do not separately control single low-level automata. Instead, the high-level automata are connected to a component called "Collect & Compare" in a high-level controller. This high-level controller receives and compares state information produced by the high-level automata through the "Collect & Compare" component and, upon recognizing the occurrence of a desirable event in the current state situation, allows it.

2.3 Concurrent operation of Automata

Typically, a result automaton of parallel composition is required to describe the concurrent behaviors of automata. However, by combining automata in parallel, the total number of states increases exponentially. If, on the other hand, the automata run concurrently without parallel combination, the result would be equivalent to a parallel composition automaton that can describe concurrent behavior without incurring state explosion.

In addition, concurrent operation of low-level automata will increases the efficiency of system operation because the low-level automata can control hardware in parallel. Several methods exist for running automata concurrently, including applying multiple processes or threads or, using multiple computers or CPUs. In the system proposed in this paper, multiple threads are used for running automata in parallel because doing so saves memory and computer resource and because automata run by multiple threads can easily communicate with each other[18,19]. Multiple threads are created and assigned to high- and low-level automata separately, which run on their own threads concurrently. In the next section, the new modular design for a DES application will be introduced and explained in detail using case studies.

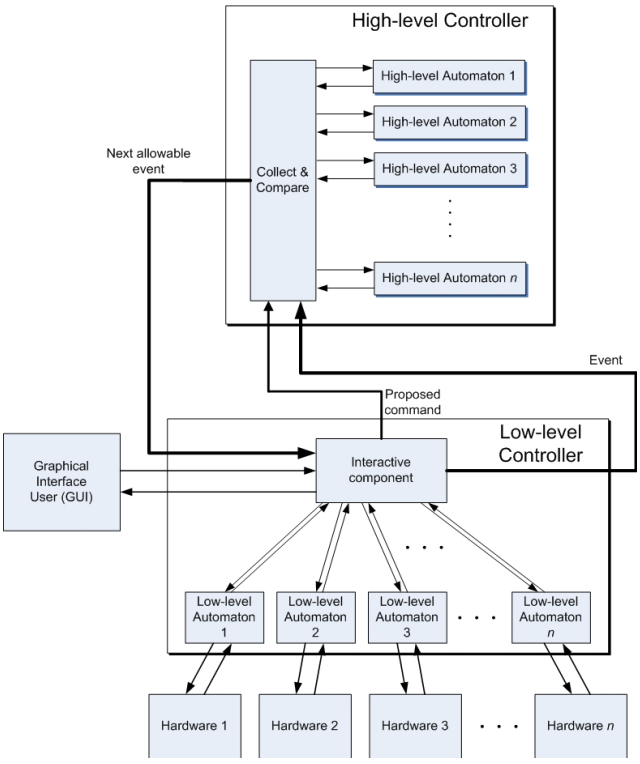


Fig. 1 Modular control structure using new approach

- Case Study I : Two machines and one buffer

A system consisting of two machines and one buffer represents a simple, but useful example for demonstrating the structure and operation of the new modular design for DES implementation. Fig. 2 shows a model system consisting of two machines (M_1 and M_2) and one buffer (B) that can hold only one part at a time. When M_1 is started, it processes a part from its supply, and, upon finishing this task, places the part into Buffer (B) before returning to its idle state and awaiting the next process. M_2 is then started, the part from the Buffer is removed, and processing of the part is begun. An "overflow" problem occurs if M_1 tries to place a part into a full buffer, while an "underflow" occurs if M_2 tries to remove a part from an empty buffer.

In this example, it is assumed that all machine starts and

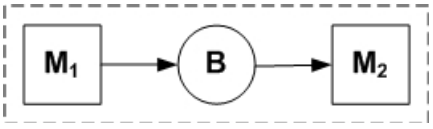


Fig. 2 Two machines and one buffer example

finishes can be observed but only starts are controllable.

In classical methods, a system like two machines and one buffer is modeled using automata that consider all possible state transitions in their design and, as such, contain both desirable and undesirable state transitions. The automata are then combined in parallel to describe concurrent automata behaviors. As the parallel composition result automaton has undesirable states and transitions, these are removed in order to produce a controller that allows only desirable state transitions.

Owing to the possibility of state explosion being generated by parallel composition, this classical method becomes impractical, for complicated systems that must be modeled by many automata.

In the new approach suggested in this paper, an automaton that includes only desirable state transitions is used to represent each process as shown in Fig. 3. However, even though the automata are designed to have only desirable state transitions, it can not guaranteed that a parallel composition result automaton composed of such automata will not have a blocking state.

In order to determine whether or not blocking states are present, therefore, the UMDES software library developed at University of Michigan was used to develop a result automaton and inspect it to examine whether it has blocking states. Using UMDES made the development and testing of this parallel composition result automaton for inspecting states simple.

Implementation of the result automaton of parallel composition, however, proved to be impractical because it has many states. We suggest here, therefore, a new modular design approach for practical and convenient implementation of a system that does not use parallel composition. The automata shown in Fig. 3 can be considered to be high-level automata because they control the operational sequences of M_1 and M_2, instead of the machines' point-to-point movements. The figure includes labels for several states, including "I_1", "P_1", "E" and "F", which stand for "M_1 idle", "M_1 processing", "Buffer Empty" and "Buffer Full' respectively.

New modular design approach is used to implement the system consisting of two machines and one buffer shown in Fig. 4.

In this system, the overall controller is composed of a high-level and a low-level controller. In the low-level controller, two low-level automata control the movements of two machines, M_1 and M_2. Each of the two low-level automata is connected to an interactive component; the M_1 low-level automaton is connected to and communicates with machine 1 (M_1). An "s1" event that means "M_1 starts" occurs; the M_1 low-level automaton begins to operate and exchanges

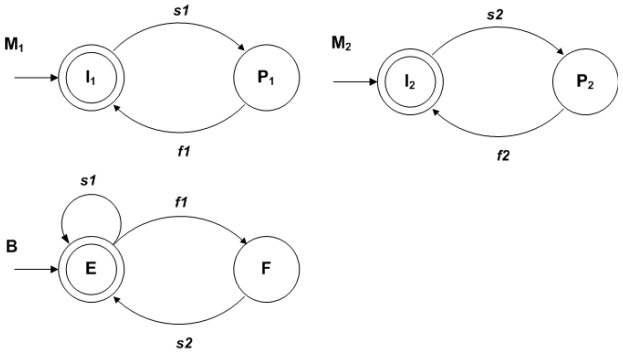


Fig. 3 Example of automata modeling two machines and one buffer

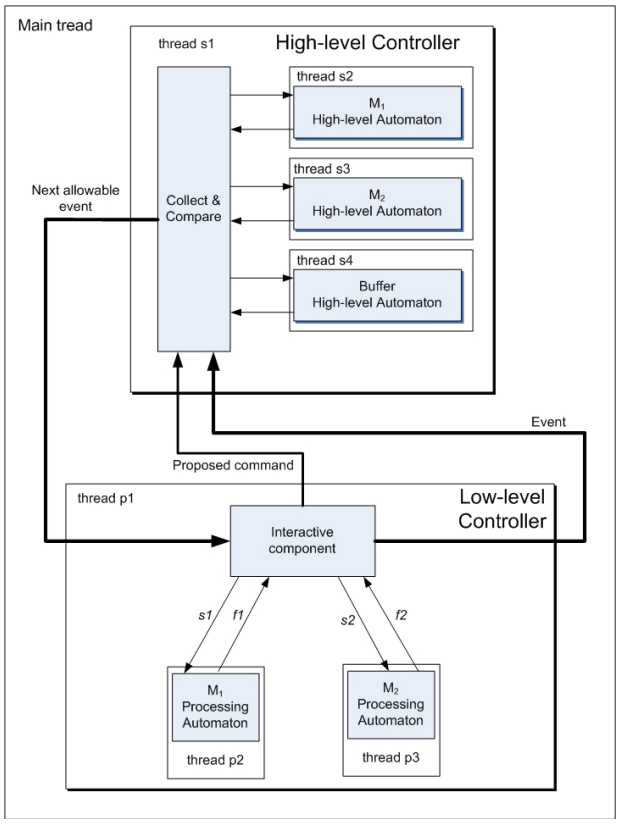


Fig. 4 Example of modular control structure for two machines and one buffer

commands with machine M_1 and so M_1 begins performing its task. After finishing its task, M_1 returns to its idle state and an "f1" event that means "M_1 finishes" is returned to an interactive component used to interconnect low-level automata. Any new low-level automaton added to control new processes can be connected with the interactive component, enabling the low-level controller to be conveniently modified and expanded.

An interactive component communicates with a Graphical User Interface (GUI) through which a user can send machine commands for execution. This interactive component receives commands sent from a user and then, changes user commands into low-level type commands called "proposed commands" that serve as translations readable by the controller of the user commands. In this system, proposed commands sent by the interactive component, are relayed first to a high-level controller used to control the sequence of low-level automata.

All of the high-level automata are connected to a component called "Collect & Compare", which is an algorithm rather than a dynamic system that receives current feasible events from the high-level automata and compares each of these with a proposed command from an interactive component in order to determine which events are feasible for the current states.

Because threads are assigned to both high- and low-level automata, they are able to run concurrently. For instance, after M_1 puts a part into a buffer (B), M_2 removes the part from B and begins processing; because two low-level automata can run concurrently, M_1 and M_2 can perform these tasks simultaneously.

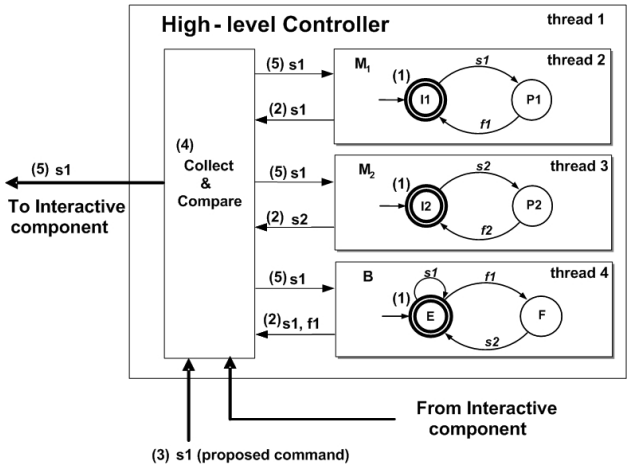


Fig. 5(a) Work flow of high-level controller

This method increases the efficiency of work. Given this setting, how can a high-level controller determine which events are feasible for the current states? Figs. 5(a), (b), and (c) show schematically how a high-level controller judges whether or not events are allowable. In the Figs. 5(a), (b), and (c), the numbers in parentheses indicate the respective work steps:

- Step 1: Fig. 5(a)

(1) The high-level controller starts at the initial states of the high-level automata, M_1, M_2, and B, which are labeled "I_1", "I_2", and "E" respectively.

(2) The high-level automata send the current feasible events corresponding to their initial states to the "Collect & Compare" component: M_1 sends "s1", as the current feasible event for its initial state "I_1"; M_2 sends "s2" and B sends "s1" and "f1".

(3) The interactive component sends "s1" (meaning that a user wants to execute M_1) as a proposed command to "Collect & Compare" component.

(4) The "Collect & Compare" component compares the current feasible events "s1", "s2", and "f1" sent from the high-level automata with the proposed command "s1". As a rule of parallel composition, "s1" is a feasible common event on the current states of the high-level automata. Using its programmed algorithm. "Collect & Compare" component can determine which events are common and which are private as well as when an event becomes feasible on the current states of the high-level automata.

(5) As "s1" has been determined to be feasible, "Collect & Compare" component sends "s1" simultaneously to the three high-level automata and an interactive component. Based on this, the high-level automata change their states; in the case of an "s1" event, only the M_1 high-level automaton changes its state from "I_1" to "P_1". The interactive component executes the allowable event "s1" in order to induce M_1 into performing its task.

- Step 2: Fig. 5(b)

(1) As an M_1 automaton has changed its state, the new current states of M_1, M_2, and B become "P_1", "I_2", and "E" respectively.

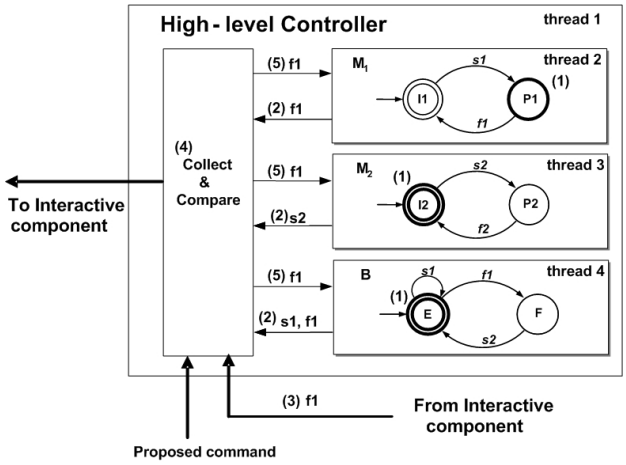


Fig. 5(b) Work flow of high-level controller

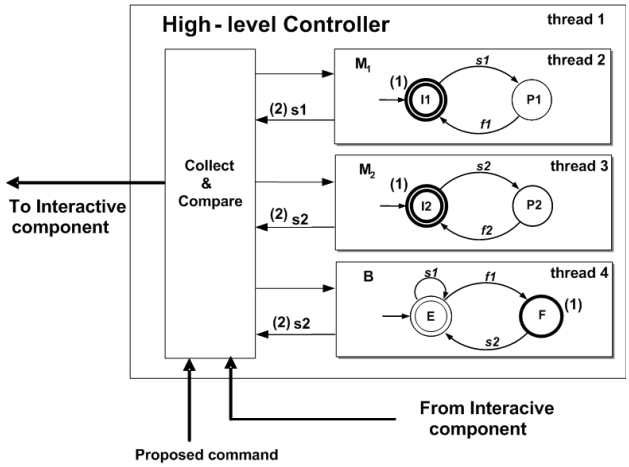


Fig. 5(c) Work flow of high-level controller

(2) For these current states, the feasible events for M_1, M_2, and B are "f1", "s2", and "s1, f1" respectively. The high-level automata, M_1, M_2, and B, send these new feasible events to the "Collect & Compare" component.

(3) In the low-level controller, the movement in M_1 triggered by the previous "s1" event finishes, and an "f1" event, indicating that the buffer is full, is triggered. The interactive component then sends "f1" to "Collect & Compare" component.

(4) The "Collect & Compare" component uses its feasibility algorithm to determine that "f1" is feasible on the current states of the three high-level automata.

(5) "Collect & Compare" component then sends "f1" to the high-level automata, M_1, M_2, and B; based on this, M_1 and B change their states.

- Step 3: Fig. 5(c)

(1) In step 2 above, the M_1 and B automata changed their states. The states of M_1, M_2, and B automata are now "I_1", "I_2", and "F" respectively, as shown in Fig. 5(c).

(2) The feasible events for the new current states of M_1, M_2, and B are "s1", "s2", and "s2" respectively. As a buffer is full and can hold only one part at a time, only "s2" is feasible among these and, therefore, it is feasible to move M_2 to remove the part from the buffer; if an "s1" event were feasible for the current set of states, the operation of M_1 could cause an "overflow" problem to occur. A high-level controller for the current states event waits for a proposed command or event from an interactive component; as this controller, allow only "s1" to be next event, it blocks occurrences of the other events for the time being.

- Case study II : Six machines and four buffers

The "Two machines and one buffer" example examined above is a simple but very useful tool for explaining the basic concept underlying the proposed modular control structure for DES implementation. The "Two machines and one buffer" example, however, is too simple to determine all of advantages of applying the new approach in order to manage complexity and state explosion. Therefore, a "Six machines and four buffers" example[20], is introduced in this section. Using this example can demonstrate how a DES application can expand and modify its modular structure in order to develop increasingly complex systems. An application result of new approach will then be compared with a classical approach in order to determine its advantages in terms of managing complexity and state explosion. Fig. 6 shows a diagram representing the "Six machines and four buffers" example discussed here. As shown in Fig. 6, the system consists of three "Two machines and one buffer" groups.

These are connected with each others through a shared third buffer called "B_3". Except for the presence of B_3, the operation of each group is similar to that in the single "Two machines and one buffer" example discussed in previous section. Machines M_2 and M_4 place parts into "B_3", M_5 removes parts from the buffer", and each group can be modeled using automata similar to the prior example. Fig. 7

shows automata representing the three groups and the third buffer, "B_3". Each of these has only desirable state transitions.

In Fig. 7, the W_2 and W_4 states of the "B_3" automaton indicate that "M_2 is working" and "M_4 is working" respectively. The modular control structure of this example has been designed using our new approach, as shown in Fig. 8.

The modular structure shown in Fig. 8 can be expanded from "Two machines and one buffer" example. As shown in Fig. 4, the modular structure for the "Two machines and one buffer" example consists of two low-level automata, M_1 and M_2, and three high-level automata, M_1, M_2 and Buffer. In order to fabricate a controller for the "Six machines and four buffers", we can designate a low-level automaton, M_3, to control the movements of Machine 3(M_3) and attach low-level automaton, M_3, to the interactive component, which of course must be modified to interact with the new M_3 low-level automaton.

An M_3 high-level automaton is placed on a high-level controller in order to control sequences of low-level automata that include the new M_3 low-level automaton; this is attached to the "Collect & Compare" component and the source codes of the "Collect & Compare" component are then modified to account for the new M_3 high-level automaton. The same process is repeated for Machines 4, 5, and 6(M_4, M_5, M_6), with low- and high-level M_4, M_5, and M_6 automata attached to the interactive and "Collect & Compare" components respectively, and the interactive and "Collect & Compare" components are modified to account for the respective new low- and high-level automata.

The modular structure in Fig. 8 shows the expansion from the "Two machines and one buffer" to the "Six machines and four buffers" model. As mentioned above, this new modular structure for low- and high-level automata has the advantages of being easily modified and expanded as a system.

Because the low- and high-level automata are connected, respectively, to an interactive component and a "Collect & Compare" component, every new machine added to an original system requires that new low- and high-level automata be attached to an interactive and "Collect & Compare" components.

By following this schema, the new modular control structure can manage increasingly complicated systems.

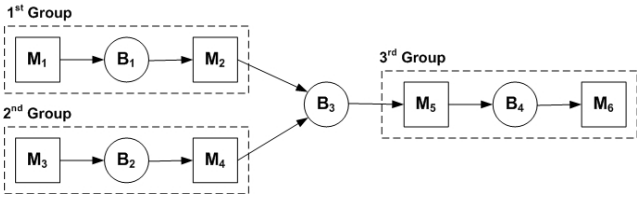


Fig. 6 Example of Six machines and four buffers

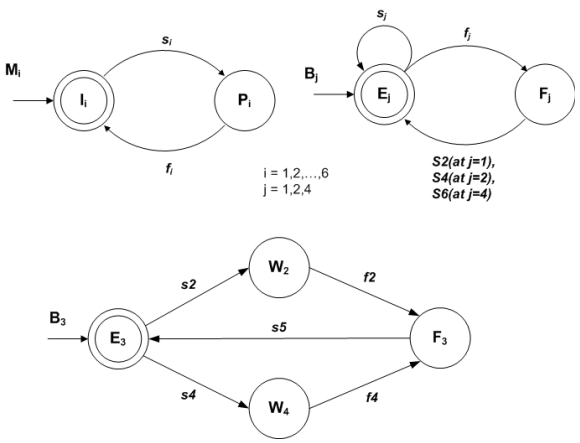


Fig. 7 Automata of the "Six machines and four buffers"

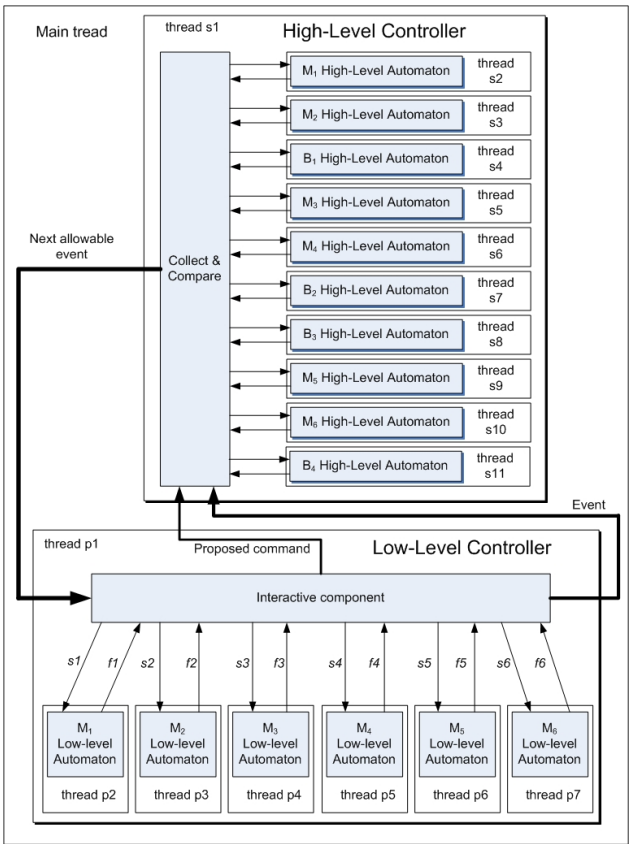


Fig. 8 Modular control structure of "Six machines and four buffers" example

As mentioned in Section 2.1, exponential growth of states is a problem for DES applications. In the case of a "Six machines and four buffers" system, a high-level controller designed using a classical method produces a parallel composition result automaton with total 216 states.

By using our new approach, however, the total number of states in the high-level controller is reduced to 22. Furthermore, whenever a new high-level automaton is added to a high-level controller, the total number of states increases linearly, whereas using the classical approach causes the number of states to increase exponentially.

The number of low-level automata needed to control hardware can be specified through design. In order to control the sequencing of the low-level automata without conflict, a high-level controller composed of high-level automata running concurrently is designed to include desirable states and state transitions. This high-level controller decides which event is feasible to operate relevant low-level automata and when it is feasible to do so.

If we determine that blocking states cannot be found for the high-level controller by checking with UMDES, and if none of the low-level automata have blocking states, we will know without having to examine parallel composition result automaton that none of the sequences of operation of the low-level automata controlled by the high-level controller conflict and, therefore, that the entire system is free of blocking states.

As mentioned in Section 2.4, the automaton consisting of high-level automata is only used to check whether there is a blocking state in a high-level controller, but not to implement a high-level controller.

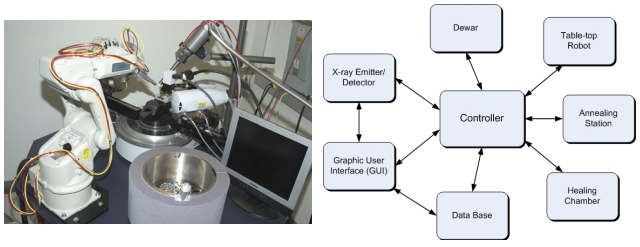
3. DES Application - AHS system

The automated annealing, healing, and screening (AHS) system that we have developed is shown schematically in Fig. 9(b). Its principal components include: an X-ray emitter/ detector for crystallographic data collection; a motorized goniometer for micro-manipulation of crystals under the X-ray beam; an annealing station and a healing chamber for the manipulation of the X-ray diffraction properties of crystals; a database for maintaining a record of the experimental protocols applied to each crystal; a table-top robot for macro-scale crystal handling; a graphical user interface through which users interact with the system; and a central controller. To ensure proper performance of this system, the operation of its components must be coordinated even though several major system components were acquired as turn-key systems intended to be operated in a purely standalone fashion.

In designing a controller for this AHS system, we should consider that each process comprising the system shares a single table-top robot for its operation and that it should be possible to easily add new processes in the future. As only one robot is used, the operation of processes using the robot should be coordinated to avoid any conflict during working. It is also important for the system design to consider managing increments in state occurring when new automata are added to control new processes.

Fig. 10 shows a structure consisting of low-level automata and an interactive component in which each process is modularized and connected to the interactive component. The overall structure consists of low-level automata serving as process components to control the robotic movements needed to mount or dismount a crystal between stations; if a new low-level automaton is added to enable a new process, it is simply connected to the interactive component.

Each new low-level automaton is simply connected to the



(a) (b)

Fig. 9 (a) AHS System (b) Schematic diagram of AHS system

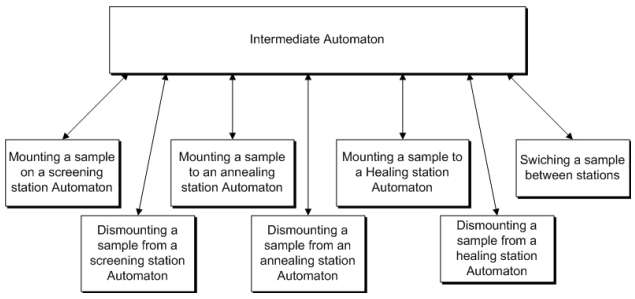


Fig. 10 Low-level automata of the AHS system

interactive component without regard of the other low-level automata and the overall sequence of low-level automata is controlled by high-level automata implemented in a high-level controller. It is a simple matter to adjust any automata that need modification.

The high-level automata of this AHS system are shown in Fig. 11. There are three high-level automata: one for controlling the annealing process, one for healing and one for screening. These high-level automata are designed to control sequences of processes represented by low-level automata and they are connected to a "Collect & Compare" component.

Multiple threads are assigned to each high- and low-level automata so that all of the high- and the low-level automata will run concurrently on their respective threads. In this manner, concurrent automata operation is attained without the use of a parallel composition result automaton.

As shown in Fig. 11, the high-level annealing, healing, and screening automata have seven, seven, and six states, respectively. In order to run these automata concurrently using the classical approach, they would have to be implemented using a parallel composition result automaton of the controller automata.

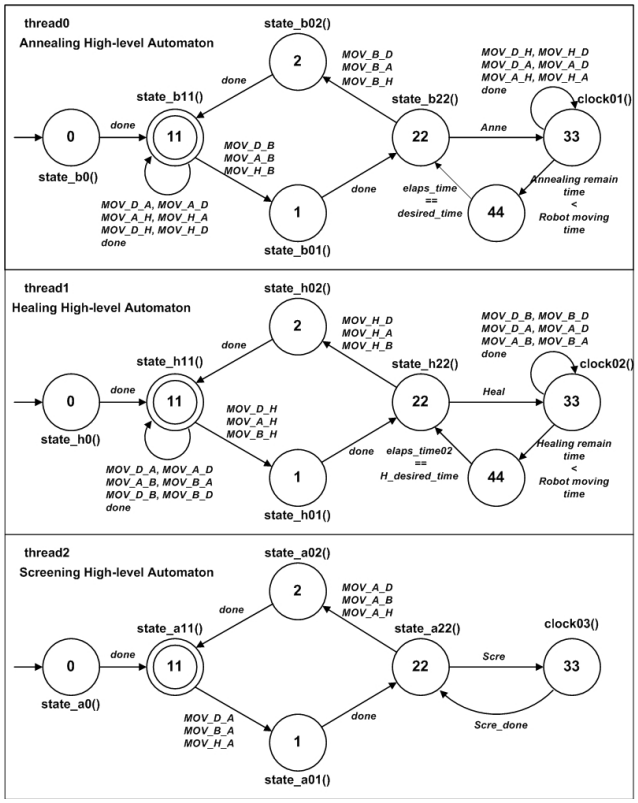


Fig. 11 High-level automata of the AHS system

The parallel composition of these automata generates a large number of states that can be computationally enumerated by using the UMDES computer program.

Using rules of parallel composition, a UMDES algorithm can compute the parallel composition result automaton and the number of states it has; in this case, UMDES determined that there were 171 total different states for the healing, annealing and screening automata. In a classical design approach, all 171 states would have to be implemented to describe the operation of the high-level controller.

Using the new modular design approach suggested in this paper to implement the high-level controller, on the other hand, reduces the total number of high-level automata states to 20; i.e. only 20 states need to be implemented for the high-level controller. Moreover, whenever new high-level automaton is added using the new modular design approach, the total number of states increases linearly; in classical design approach, the total number of states increases exponentially.

By reducing state implementation and growth rate, the proposed modular design approach is able to implement a complicated system without incurring state explosion. In addition, the new design approach provides a way to conveniently modify and expand a system. By using an interactive component to connect low-level automata and a "Collect & Compare" component to connect high-level automata, new high- or low-level automata can be added easily without difficulty or added complexity.

Despite the advantages of this design approach, advanced computer programming skill is required in order to make a computer program that can create and manage multiple threads. "Constellation", a software development tool based on C++ and developed by Real-Time Innovations, Inc.(RTI), was used to implement high- and low-level automata for the AHS system used in this study.

4. Conclusion

In this study, a new modular design approach was used to implement a system controller. The new approach uses an interactive component and a "Collect & Compare" component to effectively manage complexity and avoid the state explosion that can result from the classical design approach.

The modularization of high- and low-level automata suggested in this paper provides a means for the convenient modification and expansion of a system.

As multiple threads are assigned to each automata, their concurrent behavior can be determined without using a parallel composition result automaton. Based on our results, this new multiple-thread modular design approach represents an efficient method for designing a controller based on a DES for complicated systems.

References

[1] Ramadge, P. J., Wonham, W. M., 1987, Supervisory Control of a Class of Discrete-Event Process, SIAMJ. Contr. Optimization, 25:1 206-230.

[2] Ramadge, P. J., Wonham, W. M., 1987, Modular Feedback Logic for Discrete-Event Systems, SIAMJ. Contr. Optimization, 25:1 1202-1218.

[3] Ramadge, P. J., Wonham, W. M., 1989, The Control of Discrete Event Systems, IEEE Proceedings of IEEE, 77:1 81-98.

[4] Balemi, S., Hoffmann, G. J., Gyugyi P., 1993, Supervisory Control of a Rapid Thermal Multiprocessor, IEEE Trans. Automat. Contr., 38:7 1040-1059.

[5] Brandin, B. A., 1996, The Real-Time Supervisory Control of an Experiment Manufacturing Cell, IEEE Trans. Automat. Contr., 12:1 1-14.

[6] Ramadge, P. J., Wonham, W. M.. 1986, Modular Supervisory Control of Discrete Event Systems, Proc. 7th Int. Conf. Analysis and Optimization of Systems, 202-214.

[7] Brandin, B. A., Wonham, W. M., Benhabib, B., 1991, Discrete Event System Supervisory Control Applied to the Management of Manufacturing Workcells, Proc 7th Int. Conf. Computer-Aided Production Eng., 527-536.

[9] Max, H. de Queiroz, Jose E. R., 2002, Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell, Proceeding of the Sixth International Workshop on Discrete Event Systems (WODES'02), 377-382.

[10] Hellgren, A., Lennartson, B., Fabian, M., 2002, Modelling and PLC-based Implementation of Modular Superviosry Control, Proceeding of the Sixth International Workshop on Discrete Event Systems (WODES'02), 371-376.

[11] Lauzon, S. C., Ma, A. K. L., Mils, J. K., Benhabib, B., 1995, Implementing a Discrete-Event-System-based Supervisory Controller for a Flexible Manufacturing Workcell, IEEE Int. Conf. on Robotics and Automation, 2 1429-1434.

[12] Flordal, H., Spensieri, D., Akesson, K., Fabian, M., 2004, Supervision of Multiple Industrial Robots, IEEE Int. Conf. on Control Applications, 2 1404-1409.

[13] Kruzhanski, A. B., 1987, Discrete Event Systems: Models and Applications, IIASA Conference on “Discrete Event Systems”.

[14] Davis, M. D., Weyuker, E. J., 1983, Computablility, Complexity and Languages Academic Press, USA.

[15] Wonham, W. M., Superviosry Control of Discrete-Event Systems, Lecture note.

[16] Cassandras, C. G., Lafortune, S., 1999, Introduction to Discrete Event Systems, Kluwar Academic Publishers, USA.

[17] Du, D. Z., Ku K. I., 2001, Problem Solving in Automata, Languages, and Complexity, John Wiley & Sons, Inc., USA.

[18] Hughes, C., Hughes, T., 2004, Parallel and Distributed Programming Using C++, Addison Wesley, USA.

[19] Nichols, B., Buttlar, D., Farrell, J. P., 1996, Pthreads Programming, O'Reilly, USA.

[20] Hill R. C., Tilbury D. M., 2006, Modular Supervisory Control of Discrete-Event Systems with Abstraction and Incremental Hierarchical Construction, Proc. 8th Int. Workshop on Discrete Event System (WODES‘06), 399-406.