2000 Japan-USA Symposium on Flexible Automation
July 23-26, 2000, Ann Arbor, Michigan, USA

SOFTWARE TOOLS FOR VERIFICATION OF MODULAR FSM BASED LOGIC CONTROL FOR USE IN RECONFIGURABLE MACHINING SYSTEMS*

Eric W. Endsley, †Morrison R. Lucas, Dawn M. Tilbury
Engineering Research Center for Reconfigurable Machining Systems
Department of Mechanical Engineering and Applied Mechanics
University of Michigan
Ann Arbor, Michigan, 48109-2125
{ericend, mrlucas, tilbury}@umich.edu

ABSTRACT

This paper describes a set of software tools created for constructing and verifying a modular FSM for use as logic control. Previously we developed a method for representing the logic of a machining system by several modules. This method allows each module to be designed and verified independently of the other modules. Once the modules are verified to be correct, the system can be verified to be deadlock free and absent of some undesired behaviors, without having to construct a single large FSM representing the entire system. These tools are written in Java to facilitate integration with other software being developed for the mechanical design of reconfigurable machining tools at the Engineering Research Center for Reconfigurable Machining Systems at the University of Michigan.

INTRODUCTION AND MOTIVATION

Rapidly changing technology and consumer preference are driving manufacturers to produce products with more variation, at lower quantities, and for a shorter period of time. The traditional practice of purchasing a dedicated machining system and producing the same part with it for years or even decades is no longer practical. As the desired number of parts decreases, the portion of the cost of the machining system associated with each part becomes large. Manufacturers need machining systems with smaller design and construction costs, which are capable of producing a wider variety of parts.

The Engineering Research Center for Reconfigurable Machining Systems (ERC/RmS), at the University of Michigan, aims to develop the theory and enabling technology for reconfigurable machining systems (Koren, 1999). Instead of building a machining system from scratch each time a new part is needed, an older system can be reconfigured to produce the new part. Work is ongoing in reconfigurable machine design (Moon and Kota, 1998), and reconfigurable control design at the system level (Park *et al.*, 1999).

The control software required to run these machining systems is a surprisingly large portion of the acquisition cost of the system. The control code for each machining system is custom designed for that system. Control design accounts for around half of the time and cost associated with designing and building a machining system, and an even larger portion of the ramp up time.

The bulk of the effort in designing the control for machining systems is directed at logic control. The logic control synchronizes events and actions which occur within the system. A machining system with 10 stations may have more than 10,000 discrete I/O points. Traditionally, all of the I/O was hardwired back to a central PLC which was programmed in ladder logic. Reconfiguration of a machining system built this way is very difficult. Much of the wiring will need to be redone, and ladder logic does not lend itself well to identifying what parts of the program needs changing.

*THIS RESEARCH WAS SUPPORTED IN PART BY THE NSF-ERC UNDER GRANT NUMBER EEC95-92125.

†Eric Endsley would like to acknowledge support from a National Science Foundation Graduate Fellowship

Our previous work (Lucas *et al.*, 1999) developed a method for representing the logic of a machining system by several modules. This method allows each module to be designed and verified independently of the other modules. Once the modules are verified to be correct, the system can be verified to be deadlock free and absent of some undesired behaviors, without having to construct a single large FSM representing the entire system.

This paper presents the current state of an ongoing effort to develop software tools to assist in the design of, and perform verification on logic controllers constructed using FSM modules. It begins by describing how FSMs are packaged as modules. With that groundwork in place, the paper will proceed to describe the software tools developed and the verification procedure employed within the these tools. The paper will conclude by describing some of the future plans for these tools.

MODULAR CONTROLLERS

Within the proposed framework, the logic control of a machining system is broken down into modules. These modules will usually correspond to modules of the mechanical system or conceptual groups within the control system. Having control modules associated with each mechanical module in a design library will aid both the design of new systems and the reconfiguration of existing systems. When a particular mechanical module is pulled from the library for use, the module containing the logic control for that physical device can be retrieved and inserted into the control program.

The core of a control module is an event / action FSM like the one which appears in Figure 1. An event / action FSM is different from a traditional FSM in that a transition between states can have a set of events which are forced to occur when that transition occurs. The events which are forced to occur are called actions.

The event / action FSM is intuitive for use in logic control, as we think of a logic controller reacting to events by causing other events. For example, when the FSM for Module A is in state 1, the initial state, Module A will react to the event 1.ret by transitioning to state 4 and generating the event 2.ret. From state 1, Module A will react to event 1.drill by generating the events 2.on and 3.drilling.

In addition to the FSM, each module has a set of port nodes. The port nodes are places where the module is connected to other modules in the control system. Each port node has a name, a set of events which come in through the node, and a set of events which leave through the node.

All of the events and actions which appear in the FSM appear in exactly one event set of one node. The events will appear in an incoming event set, and the actions will appear in an outgoing event set. Within the software tools, all events and actions have a prefix which is the name of the node through which it enters or leaves. Most events entering a module will be actions

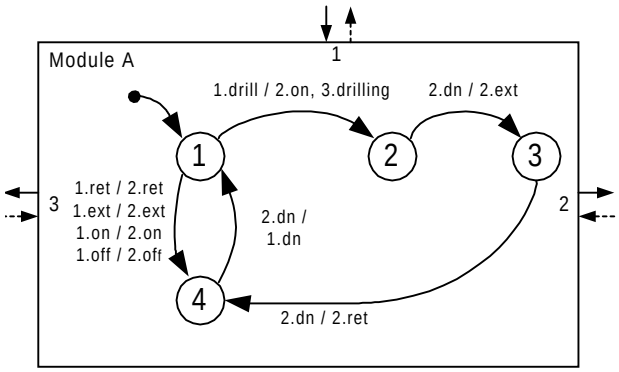

Figure 1. Module A - Drill Macro

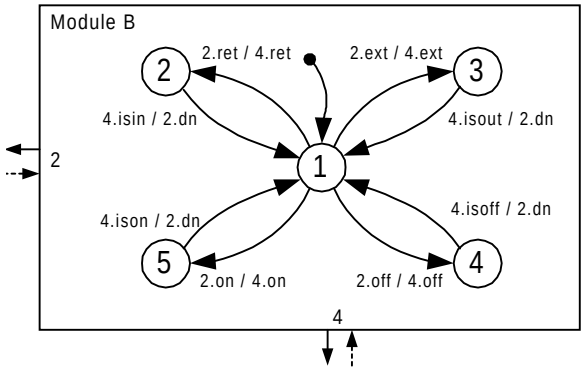

Figure 2. Module B - Spindle Module

from another module, and the actions generated by this module will become events to another module. The remaining events and actions are physical signals to and from the machine.

Figures 1 and 2 show two simple control modules. They contain control logic related to a drill spindle with pneumatic feed. Module B is contains the logic for the physical spindle module. The port attached to port node 4 of Module B is connected to the physical system.

Module A is a simple controller for Module B. It passes most of the events it receives on port node 1 out through port node 2. Module A also provides a new command for the system. In response to the drill event on port node 1, Module A performs a drill cycle by turning on the spindle, extending the drill and retracting the drill. Module A also reports that a hole is being drilled by sending the drilling event out port node 3. Port node 1 of Module A could be connected to a user interface, or to another module in a larger controller.

A controller is constructed by attaching modules together with ports. These ports are objects separate from the modules, which connect a node on one module to a node on another mod-

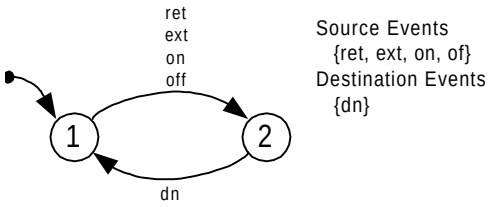

Figure 3. Port Between Module A and Module B

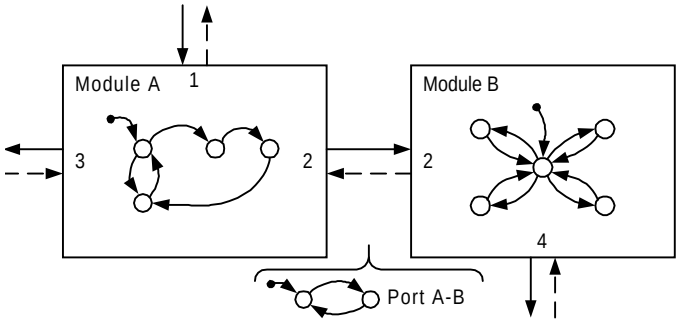

Figure 4. Module A Connected to Module B

ule. A port contains an FSM which defines the allowable sequences of events which pass through the port. Because this FSM just defines a language, and does not perform control, it is a traditional FSM, not an event / action FSM. The port also divides the events which pass through the port into two sets, ones generated by the source of the port, and ones generated by the destination of the event.

Figure 3 contains the port defining the allowable communication between Module A and Module B. Module A is allowed to send the events ret, ext, on, and off. Module B will respond with the event dn.

Figure 4 shows Module A connected to Module B through Port A-B. The solid arrow indicates that Module A is the source for Port A-B and that Module B is the destination. The dashed arrow serves as a reminder that the communication is 2-way.

When two modules are connected together with a port they are forced to operate in a synchronized manner. By analyzing the FSMs of the two modules it is possible to verify, prior to implementing and attempting to run the controller, that the two modules will run without conflict. The details of this verification appear in the next section. By “marking” specific states in each module, and checking that there always exists a path back to those states, it will be possible to verify the that controller is deadlock free.

The major result of our previous work (Lucas *et al.*, 1999) is that the modules can be verified using only the information contained in the ports to which they are connected, without detailed knowledge of the other modules in the system. This allows the modules to be designed and verified independently of each other. It also allows one module to be easily replaced with any other which meets the requirements defined by the port.

SOFTWARE TOOLS

This section describes the software tools being developed to model, implement and verify FSM-based controllers. These software tools are implemented in Java. In addition to its platform independence, it is desirable to use Java because it will be easier to integrate these tools with other projects within the ERC/RmS which are also using Java, such as the reconfigurable machine design program developed by Moon and Kota (1998).

The software tools are constructed from a group of Java classes and methods. There are classes for FSMs, states, transitions, actions, ports, modules, and the control system. There are also several classes for support operations such as reading objects from data files and performing FSM manipulations.

The core FSM Object

The core of the system is the FSM object. An FSM object contains all of the information necessary to describe an event / action FSM for use in logic control.

The FSM object contains a hierarchy of other objects. The FSM contains a set of states objects. In addition to fields describing a particular state, each state object has a set of transitions to other states. Each transition has a triggering event and a set of action objects. The action object contains an event which is to be generated.

In a similar manner, both module objects and port objects contain an FSM object; a system object has a collection of module and port objects.

A Parallel Composition for Event / Action FSMs

The primary operation needed for the verification of FSM modules is a parallel composition which gives the desired behavior for FSMs with events and actions. This operation will provide the functionality to check a module against a port, and to combine two modules into a single module.

The synchronization method used in the traditional FSM parallel composition consists of a single rule: a transition triggered by an event which is common to both FSMs can only occur when transitions out of the current state of each machine exist for that event. If one of the FSMs is preventing an event from occurring by not having a transition for that event, that FSM is said to have disabled that event.

Associated with each event is the concept of controllability. It is not valid for an FSM to disable an event which is designated uncontrollable. When checking a module against a port the

events which are generated at the end of the port not connected to the module are defined as uncontrollable.

The above behavior is included in the parallel composition for event / action FSMs. However, additional rules are needed to account for the actions. Actions are considered when determining which events are common to both FSMs. Therefore, if an event is a trigger in one FSM and an action in the other, that event can only occur when it is driven by an action.

When a transition with actions occurs, the actions are placed on an action queue. These actions are then used to fire additional transitions in the machines until the queue empties. The transition in the resulting FSM goes from the states that the FSMs were in when the transition occurred to the states that the FSMs are in after the queue empties. All of the actions that occur during the simulation become actions of that transitions. The software detects racing conditions where an event could cause an infinite number of actions. It also detects the invalid condition in which one of the FSMs tries to disable an action.

The action queue is a last in first out (LIFO) buffer. The actions are placed on the queue so that if no other actions are generated during the simulation, the actions from the transition in the first FSM would occur in the order in which they appear, then the actions from the second FSM in the order in which they appear. If an action causes a transition to occur which causes more actions, these actions are added to the front of the queue and will be processed first. The decision to have the action queue behave as LIFO was arbitrary. The queue could also have been made first in first out (FIFO). A direction of future research is how the behavior of the controller is different based on this choice.

A module is considered to behave correctly if this parallel composition can be successfully performed between the FSM of the module and the FSMs of all of the ports to which the module connects, and the resulting FSM is non-blocking. This verification will check that there are no conflicts between the module and the ports. That is, no uncontrollable events or actions are disabled, there are no racing conditions, and the final controller will be deadlock free.

Example Parallel Composition

Table 1 shows how the 1.drill transition would be resolved when Module A and Module B are combined using the event / action form of the parallel composition. From state (1,1) (State of Module A, State of Module B), the event 1.drill is enabled because there is a transition for it in Module A, and Module B does not contain it as either an event or an action. When 1.drill happens, Module A places 2.on and 3.drilling on the action queue and the system moves to state (2,1). The system then processes the queued action 2.on. This causes the 2.on transition in Module B to fire, placing 4.on onto the action queue and moving the system to (2,5). The actions 4.on and 3.drilling are then processed. Because these actions go ports other than the common one, they do not change the state of the system. So as a result of 1.drill occurring in state (1,1) the system moves to state (2,5) and generates the actions 2.on, 4.on and 3.drilling.

Table 1. Simulation of One Transition

Source	Event	Dest	Action Queue	Action History
1,1	1.drill	2,1	2.on, 3.drilling	
2,1	2.on	2,5	4.on, 3.drilling	2.on
2,5	4.on	2,5	3.drilling	2.on, 4.on
2,5	3.drilling	2,5		2.on, 4.on, 3.drilling
1,1	1.drill	2,5		2.on, 4.on, 3.drilling

CONCLUSIONS AND FUTURE WORK

This paper described a set of software tools made for the verification of modular finite machines for use in logic control for machining systems. It described how FSM modules are used to construct a logic controller, and how these modules are verified. A modified parallel composition operator, necessary for the verification and combination of FSM modules, was also presented.

There are several projects that can be started once these basic tools are completed. One project currently underway is adding to the tool set the ability to convert a modular FSM based controller into ladder logic to demonstrate how this framework can be implemented on existing PLC's. Planned projects include providing these tools with a user-friendly GUI, and integrating them with the mechanical design tools being developed at the ERC/RmS to produce a complete package for the design of reconfigurable machining systems.

REFERENCES

Koren, Y., 1999, "Reconfigurable Machining Systems; Vision with Examples," ERC/RMS #19, University of Michigan.

Lucas, M. R., Endsley, E. W., and Tilbury, D. M., 1999, "Coordinated Logic Control for Reconfigurable Machine Tools," in: *Proceedings of the American Control Conference*, pp. 2107–2113.

Moon, Y.-M. and Kota, S., 1998, "Generalized Kinematic Modeling Method for Reconfigurable Machine Tools," in: *Proceedings of the ASME Design Engineering Technical Conferences*, Atlanta.

Park, E., Tilbury, D. M., and Khargonekar, P. P., 1999, "Modular Logic Controllers for Machining Systems: Formal Representation and Performance Analysis using Petri Nets," *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 6, pp. 1046–1061.