2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Factory-Level Control Consolidation Using Event-Condition-Action: Case Study on the Reconfigurable Factory Testbed

Lindsay V. Allen, Jing Zhang, James Moyne, and Dawn M. Tilbury

Abstract— **During its lifetime, a manufacturing system often has added functionalities or significant changes to its control system that can result in a broken control hierarchy, causing problems with debugging and reconfiguration. For the up-and-coming field of Reconfigurable Manufacturing Systems, poor reconfigurability is especially problematic. A procedure for restructuring a standard control hierarchy is presented, including how to decide where to move particular logic and preserve Event-Condition-Action (ECA) based structure, when originally present. This procedure is applied to an ECA-based case study — the Reconfigurable Factory Testbed at the University of Michigan — to demonstrate its use. Additional lessons learned about ECA's role in restructuring are also presented through the case study.**

I. Introduction

A good hierarchical control system can provide many desirable properties for a manufacturing system, including clearly defining the control scope of components and isolating different levels of control, which can aid in reconfigurability. Breaks in the control hierarchy can occur, however, resulting from the original design process or modification of an existing system. If a manufacturing system's control system has components from different vendors or subsystems developed by different teams, a broken hierarchy can result. This difficulty can be avoided through careful planning early in the design process, such as defining the interface through which the components or subsystems will communicate and clearly outlining each of their roles and responsibilities. When an existing system is upgraded or modified, the control hierarchy can be preserved if the modification is invisible to the rest of the system. Otherwise, temporarily breaking the control hierarchy may be necessary if the modification affects the hierarchy. For example, if the control system has one overarching controller that handles both enterprise and factory level control and the enterprise control is to be improved by adding a controller dedicated solely to the enterprise level, then temporarily there would be an enterprise controller and an enterprise-factory controller, breaking the hierarchy.

Although manufacturing systems lacking a clear control hierarchy may still be made to run properly, their jumbled structures can cause difficulty in debugging, leading to excessive down-time, and poor reconfigurability. It may be difficult to coordinate disparate resources to work together to achieve factory goals. Restructuring the system can address these issues. A variety of methods and paradigms for designing a new control system have been proposed, such as the hierarchical/heterarchical blended structure in [7] and bionic, fractal, and holonic manufacturing concepts reviewed in [14]. Upgrades and retrofits to replace obsolete parts, improve efficiency or completely replace the control system are all discussed in industry journals, see [6], [8], and [15] for example. Less discussed, however, is how to restructure an existing control hierarchy to fix or improve it. Such restructuring may be a better approach than replacing the entire control system in some cases, and may be especially advantageous when the system must be down for other maintenance or upgrade reasons.

This paper focuses on the evolution and restructuring of control hierarchies, with emphasis on Event-Condition-Action (ECA) based control hierarchies. An ECA system consists of a set of rules. The rules are triggered by events, check whether certain conditions are satisfied, and, based on these conditions, perform certain actions [16]. A case study system that evolved into having an ECA-based control with unclear hierarchy is described in Section II. Some guidelines related to performing a consolidation of a control hierarchy, including communication restructuring, are discussed in Section III. How these guidelines applied to the case study system's control hierarchy are the basis of Section IV. The communication restructuring is applied to the case study system in Section V. Lessons learned from the implementation of the consolidation thus far are shared in Section VI. Section VII provides conclusions and a description of future work on the case study system's consolidation. To assist the reader, after the References section there is an appendix describing acronyms commonly used throughout this paper.

II. Case Study System

The Reconfigurable Factory Testbed is a near-industrial grade, reconfigurable manufacturing system at the University of Michigan Engineering Research Center. It is used for research, teaching, and technology transfer purposes [11].

A. Basics of the Testbed

The testbed consists of hardware and software resources that communicate via networks, as shown in Figure 1, and are coordinated by a factory-wide ECA-based management system called the Control Workflow Manager (CWM) [16].

The CWM is part of the Software Infrastructure which also includes middleware, software modules, and programs called trackers which interact with the RFID system. A Human Machine Interface is used to place orders for the testbed. A

This work was supported in part by the National Science Foundation–ERC for Reconfigurable Manufacturing Systems under Grant EEC95-92125.

Authors are with the University of Michigan, Ann Arbor, MI 48109-2125 USA (e-mail lzallen, jingzh, moyne, tilbury@umich.edu)

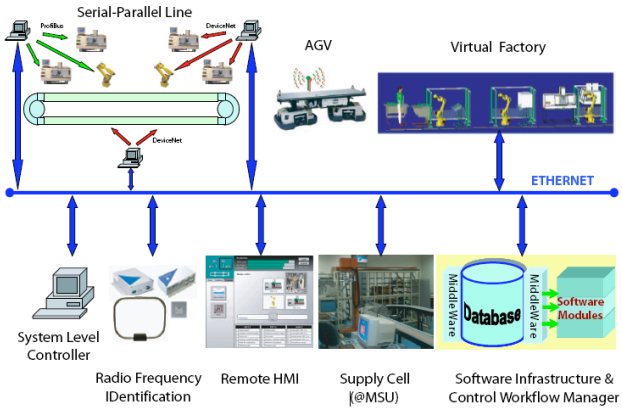

Fig. 1. Reconfigurable Factory Testbed Components

set of RFID antennas reads part and pallet numbers through their RFID tags and provides this information to the rest of the testbed system via the database. A system level controller (SLC) coordinates all of the hardware, including the serial-parallel line, virtual factory, and wheel processing station.

The hardware resources include milling machines, robots and a conveyor. These resources form a serial-parallel manufacturing line consisting of two cells, each of which has one robot and two milling machines, which are connected by a conveyor. The conveyor also connects the cells to the Supply Cell, where parts are loaded, and to the Virtual Factory, where assembly of finished parts is simulated. The wheel processing station emulates the production of wheels for the system, while the database tracks the wheel inventory and triggers an event to produce more when the inventory is low.

B. Event-Condition-Action Paradigm

The testbed's control system is Event-Condition-Action (ECA) based. An ECA rule is triggered by the arrival of an event; the conditions are checked by querying software modules, a database, or controllers (possibly several iterations of queries and responses); and the rule terminates in an action (or lack thereof) on the outside environment. An advantage of using ECA is that it can separate behavior from state — the ECA rules describe behavior and can receive from or query other sources for state information on which to base that behavior. Adopting ECA rules is also advantageous in providing reconfigurability of event-driven systems as demonstrated in implementing low-level logic controllers [1][2] and high-level management and control systems [16]. The ECA paradigm (or more generally the event/action paradigm) is widely adopted in many different areas, such as active database [3][13], logic control [1][2][16], and semantic web [4]. For a detailed survey on the event/action paradigm see [12].

C. Evolution of Control Hierarchy

The testbed's control system was developed by several different groups simultaneously, as often happens in industry. The SLC was created in a non-ECA form at the same time as, but by a different group than, the ECA-based CWM. Later on, the SLC was transformed to an ECA-based system to move the entire control system more towards ECA. As a result of this parallel development process, the resulting control hierarchy was not clear, the component interactions were not plainly separated, and system goals were not fully aligned.

The existing control hierarchy is shown in Figure 2, where it consists of separate control levels. In general, manufacturing control systems have four levels: enterprise, factory, cell, and machine. The testbed does not have enterprise level control, but does have the other three.

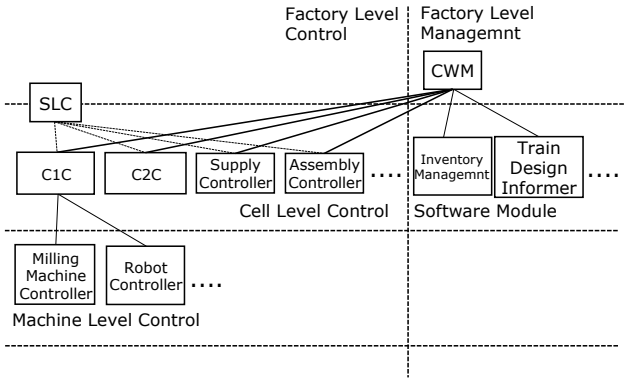

Fig. 2. ECA-Based Control Hierarchy of Testbed System

Having these three separate control levels separated by well-defined interfaces isolates the levels from one another so that problems can be more easily debugged and reconfiguration performed. The SLC performs aspects of both factory-level control and cell level control, which breaks the ECA-based control hierarchy and causes the debugging and reconfiguring problems associated with the lack of separate levels and well-defined interfaces.

D. Testbed Controllers

At the factory control level of the testbed, there is a software control system (the CWM) and a logic controller (part of the SLC). At the cell control level, there are controllers for Cells 1 and 2, Supply Cell, Assembly, RFID, Conveyor and the other part of the SLC. The machine control level consists of all the machine and robot controllers.

The Cell 1 and 2 Controllers coordinate the milling machines and robots inside Cells 1 and 2, respectively. They receive instructions from the SLC/CWM and send their own instructions to the machine controllers and robot controllers. The Cell 1 Controller (C1C) is a research platform written in Modular Finite State Machine (MFSM) and running in Java [5]. The Cell 2 Controller (C2C) is a Siemens soft PLC running sequential function charts and ladder logic. The Cell Controllers' programming language should not matter to controllers at other levels so long as the interfaces between the levels are well-defined.

The SLC is implemented as a set of communicating ECA MFSMs, where an ECA MFSM is an MFSM that has one main module with ECA rules and other peripheral modules

that hold state information [2]. Part of the SLC's logic belongs to Factory Level Control, since it coordinates the Cell Level controllers, while the other part belongs to Cell Level Control, since it keeps track of the states of the cells and only passes along requests that it knows the cells can currently perform. As previously mentioned, the SLC breaks the leveled control hierarchy of the testbed, motivating its elimination which would make the CWM the sole factory level controller.

III. Discussion of Restructuring Guidelines

To restructure a broken control hierarchy, several guidelines should be followed. The control structure must be rearranged so that the levels are distinct and separated by clear interfaces; a procedure for doing so is described below. In addition, changing the control structure will also require making modifications to the communication flow and issues associated with that are also discussed.

A. Control Hierarchy

The first step in performing a control consolidation or restructuring is identifying the current control hierarchy and any deviations from the standard form, as done in Section II for the case study ECA-based system. These deviations can be addressed by moving elements of the logic of the non-conforming controller either up to the higher level or down to the lower level of control. Deciding whether to move particular logic up or down should not be arbitrary, but rather based on the following basic procedure. The procedure begins with specifying the control hierarchy by identifying the interfaces between distinct control levels, the basic capabilities provided at each level, and the information exchange across the interfaces between levels. The decision as to where to partition the hierarchy is largely implementation-specific and often governed by legacy implementation and capabilities provided by off-the-shelf components integrated into the control hierarchy (such as PLCs). Practically, partitions should be chosen that allow for the push-up and push-down of functionalities that can be supported by these legacy and off-the-shelf components.

For each controller that straddles two control levels, its functionalities should be identified and framed as methods with well-defined input/output interfaces. Viewing the controller as a collection of methods allows each method to be moved to its appropriate control level independently of the others. For each method, one should determine whether it performs factory, cell, or machine level control and move it to the appropriate level. This determination can be straightforward if the interface of each control level is well-defined, because then the method's I/O simply needs to be matched with an interface's I/O. If a method produces outputs that are consistent with information specified as being produced at the factory/cell interface by the factory component, then the method should be moved up. If the method's outputs, however, are consistent with information specified as being produced at that interface by the cell component, then the method should be moved down.

In this manner, all of the functionalities of the non-conforming controllers will be moved elsewhere, thus eliminating these controllers. Following this procedure will yield a control hierarchy that has the same interfaces as the original hierarchy but conforms to the standard level divisions. Also if the original system is ECA-based, this procedure can preserve that characteristic in the resulting system by combining the ECA rules of each method with the ECA rules of the controller they are being merged into. More information about how to combine the ECA rules is described in Section VI. In making these changes to the control hierarchy, however, any communication that previously went through the now-eliminated controllers will need to be redirected, a procedure described in the next section.

B. Communication Structure

Restructuring the control hierarchy requires modifying the communication structure to accommodate the changes. The modification of the communication structure can be largely implementation specific. The testbed control consolidation process (discussed in Section IV) revealed that, once a formalism for control consolidation is established, the reconfiguration of communication to support the consolidation can become one of the most difficult tasks. Following the guideline for moving controller logic, however, can assist in providing direction for the communication restructuring. In using this guideline, the controller to be eliminated was framed as several methods, each with their own inputs and outputs, which collectively should be all of the inputs and outputs for the entire controller. Thus, inputs and outputs associated with a particular method should move with that method to its new location — for example if a method M was moved to a cell level controller CLC, then all inputs and outputs of M should become inputs and outputs of CLC. There may be some inputs used by several methods, in which case a copy of those inputs must be routed to the new locations of each of the methods that use it.

When rerouting communication, special attention should be paid to the communication protocols used by the controllers affected by the consolidation. If a method is moved to a controller with a different communication protocol (i.e. TCP/IP, OPC tags, XML etc.), then the inputs to that method may need to be converted from their original protocol to the protocol of the method's new controller. Likewise, the method's outputs may need to be converted before going to their final destination. If the controller to be eliminated has special communication subsystems to support it, these also need to be eliminated as part of the consolidation.

The communication involved in material handling presents particular difficulty. Although the control structure is made hierarchical through the consolidation, for practical purposes, the communication structure may not be. With material handling, there are often a number of hand-shaking type interactions among controllers and software modules at the same level, and if this communication had to be routed up to the parent control level and then back down, it could cause performance degradation. Thus, for performance reasons, one

may allow controllers and software modules at the same level to communicate directly for material handling purposes.

IV. Control Restructuring Case Study

The starting configuration for the case study system is shown in Figure 2, where the SLC breaks the desired structure by performing both factory and cell level control. The structure will be consolidated by eliminating the SLC and moving all of its logic either up to the CWM (factory level control) or down to the cell level controllers. The resulting control structure will be similar to that in Figure 2 but with the SLC removed.

Applying the procedure of Section III.A, the SLC can be viewed as a collection of methods, each of which performs a function or set of functions, of the SLC. As shown in Figure 3, the SLC is composed of a set of communicating ECA MFSMs, each of which are responsible for a distinct subsystem, making it easier to identify the SLC's equivalent methods.

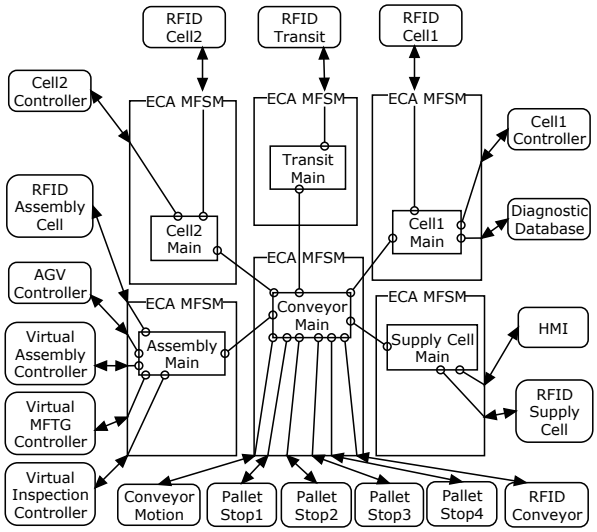

Fig. 3. SLC Structure

Using the push up/down criteria, the logic of each method of each ECA MFSM in the SLC can be appropriately moved to its new location in the control hierarchy. To demonstrate how the criteria works, it is applied to moving the methods in the Cell 1 ECA MFSM of the SLC (push-down) and moving the RFID subsystem (push-up).

A. Push-Down Example

The Cell 1 ECA MFSM of the SLC, called SLC-C1, keeps track of whether Cell 1 is in a fault state, whether the robot in Cell 1 is near the conveyor, whether there is a finished part in Cell 1 waiting to be unloaded, and which parts, if any, Cell 1 is currently processing. SLC-C1 uses this state information to filter requests from the software infrastructure (SWI), which includes all of the software such as the CWM and trackers. These requests are filtered such that only requests that Cell 1 can currently accommodate are passed along to the Cell 1 Controller (C1C). This function can be considered the first method, M1. Additionally, SLC-C1 tells the C1C which part to unload if Cell 1 currently has two finished parts waiting and an empty pallet available, which can be seen as the second method, M2. Beyond these methods, the SLC-C1 simply passes information between the SWI and the C1C, called method M3. All of the Cell 1 ECA MFSM's methods, including their I/O, are summarized in Table I.

TABLE I
Methods for Cell 1 ECA MFSM of SLC

Name	Function	Input (from)	Output (to)	Moved
M1	On pallet arrival, check if C1C available; if so pass on request, if not tell SWI rejected	PartDataReady (SWI), LoadPart1/2 (SWI) RobotAway (C1C)	LoadPartAck (SWI), PartNotTaken (SWI), Part1/2Taken (SWI), Start1/2 (C1C)	Down to C1C
M2	Tells C1C when an empty pallet is available and which part to unload	LoadPart0 (SWI), unload1/2 (C1C)	unload_part1/2 (C1C)	Down to C1C
M3	Passes messages between C1C and SWI	done1/2 (C1C)	Part1/2Finished (SWI)	Down to C1C

The first method, M1, involves declining to process a part or rejecting an empty pallet as unnecessary and should be handled by the C1C because it only involves Cell 1, and not coordination among various cells or subsystems. To confirm this conclusion, the interfaces that would result from pushing M1 up and down can be compared to see which better separates the factory and cell level control functions. If method M1 were moved up to the CWM in the SWI, then the I/O between M1 and SWI would become internal and the factory-cell interface between CWM and C1C would be: CWM tells C1C to start processing a part that has arrived (Start1/2) and C1C responding only to say that the robot has moved (RobotAway), not what was done with pallet and/or part. If, however, method M1 is moved down to C1C, then that factory-cell interface would be: CWM tells C1C when a part has arrived (PartDataReady), which part it is and requests its processing (LoadPart1/2), and the C1C acknowledges a part arrival and processing request (LoadPartAck), and responds with what is done with the part (PartNotTaken or Part1/2Taken). Clearly, the interface resulting from pushing M1 down to C1C better separates the factory and cell level control functions.

The second method, M2, keeps track of which finished part should be unloaded first from Cell 1 and is also clearly a Cell 1 task and should be moved down. It is less clear whether M3 should be pushed up or down based on its function and I/O, but practical implementation issues can be considered as well. Because it has been decided already that M1 and M2 should be pushed down, and the three methods are currently integrated together, it is simpler to keep M3 with M1 and

M2. Thus M3 will also be moved down to C1C.

B. Push-Up Example

The RFID, as a subsystem in the testbed, is responsible for part tracking. It is a stand-alone subsystem and its control infrastructure is not part of the factory-wide software control infrastructure. The incorporation of the RFID subsystem into the factory-wide software control infrastructure is an example of a push-up of logic in the testbed consolidation.

The current functionalities of the RFID include tracking the parts and pallets and providing the tracking information to the lower level controllers through OPC. The RFID system does not perform much control work. Rather it is an information collector and provider that assists the controllers, which is why the RFID does not appear in Figure 2. Instead, its role is illustrated in Figure 4, where it can be seen that the RFID system communicates with both logical controllers and the software control infrastructure to provide tracking information.

Before the consolidation, a typical control flow involving the tracking information from the RFID was as follows (where the cell level controllers communicated via their respective parts of the SLC):

- Conveyor Controller sends "pallet stopped" event to Cell 1 Controller (C1C);
- C1C queries the RFID whether the pallet contains part 1, part 2, or no part (is empty);
- C1C controls the manufacturing process based on the query results;
- C1C sends a "process done" or "part damaged" event to Conveyor Controller;
- Conveyor Controller releases the pallet.

The CWM is not involved in the above control flow. With the CWM participating in the above manufacturing process, the new control flow is as follows:

1) Conveyor Controller sends "pallet stopped" event to CWM;
2) CWM fires an ECA rule whose trigger is "pallet stopped" and whose condition and action are calls to proper software modules;
3) The software module called by this condition is a "production module" which queries RFID about the presence of parts, determines the resources needed to process the part, and the order in which the resources are invoked. The software module called by this action is a "communication module" which coordinates invoking of the resources;
4) C1C sends "process done" or "part damaged" event to the CWM;
5) CWM fires another ECA rule whose trigger is "process done/part damaged" event and whose condition and action are calls to software modules.
6) The module called by this condition is the "production module" which is able to check if the pallet can be released. The module called by this action is the "communication module" which can instruct Conveyor Controller to "release pallet".

In general, other control flows involving the RFID are similar to this example. Therefore, in order to enable the CWM to control these flows, there need to be new rules processing events from Cell 1/2 Controllers (e.g., step 5) and events from Conveyor Controller (e.g., step 2).

TABLE II
NEW RULES FROM PUSH-UP OF RFID

Event	From	To	Condition	Action
"pallet stopped" at some cell	Conveyor Controller	CWM	Call "production module"	Call "communication module"
"process done" or "part damaged"	Cell Controller	CWM	Call "production module"	Call "communication module"

Table II summarizes the two rules processing these two types of events. Furthermore, two new software modules — "production module" and "communication module" — need to be built as shown in Table III. Both of them are called by the CWM during the firing of new ECA rules. The production module will be used by the CWM to query the RFID system (e.g. steps 3 and 6) and allocate resources. The communication module is called to coordinate the invocation of these resources, (e.g., the instructions for the Cell 1/2 Controllers and Conveyor Controller in the steps 3 and 6). In Table III are the inputs, outputs and functionalities of these two new modules.

TABLE III
NEW MODULES FROM PUSH-UP OF RFID

Module Name	Input	Output	Functions
Production	-OPC tags in OPC servers -Variables in database	-Resources needed in manufacturing process -An order in which resources are invoked	-Query RFID system about presence of parts and readiness of releasing pallets; -Determine resources needed by manufacturing procedure; -Determine order in which resources are to be invoked
Communication	-Resources needed -An order in which resources are invoked	Coordinated invocation of resources	Coordinate instruction and resource invocation for the Cell 1/2 Controllers and Conveyor Controller

C. Logic Decisions for All Parts of SLC

Applying the same procedure to the other parts of the SLC yields a push-up or push-down decision for each, as summarized in Table IV. Although each of these ECA MFSMs are split into methods and a push up/down decision made for each method, Table IV shows a single decision for all methods of each ECA MFSM because in this example case, all methods within each ECA MFSM were moved to the same location.

TABLE IV
DECISIONS FOR MOVING SLC LOGIC

ECA MFSM	Moved	Reasoning
Cell 1	Down to C1C	Only cell 1 involved in filtering its requests and keeping its unload order
Cell 2	Down to C2C	Only cell 2 involved in filtering its requests and keeping its unload order
Transit	Up to CWM	Passes RFID info to Conveyor – coordinates subsystems
Supply Cell	Up to CWM	Communicates with human machine interface to release pallet – coordinates subsystems with software
Assembly	Up to CWM	Coordinates cell level controllers – AGV and virtual factory
Conveyor	Up to CWM	Coordinates conveyor controller with Cell 1 and 2 controllers

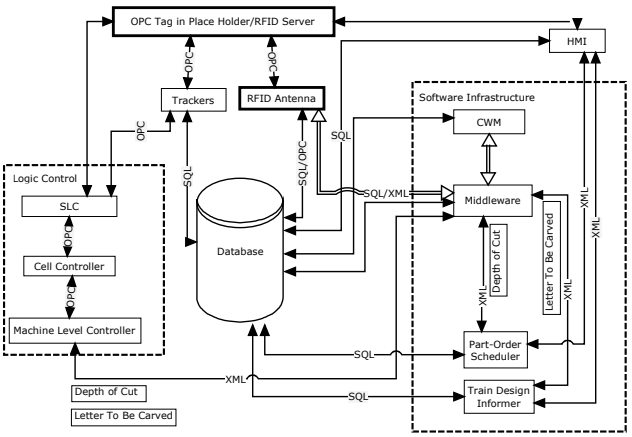

Fig. 4. Communication Flow of Testbed

V. COMMUNICATION RESTRUCTURING OF CASE STUDY

Moving the individual inputs/outputs to the same locations as their associated methods is a relatively straight-forward process. Eliminating communication subsystems associated with the controller(s) to be eliminated is a more difficult problem, one which can benefit from discussing the case study's removal of the trackers. In the testbed system, the tracking of parts and pallets is currently done by two subsystems: RFID and trackers. The RFID system has three antennas, one PLC, one server holding OPC tags and one client polling the server and sending out notification. A typical working cycle of the RFID starts with the detection of pallets and parts by the antennas. Upon detection, the server reads data from the PLC. When the server gets the data, the client will send notification to the SLC through the trackers.

Figure 4 shows the communication flow in the testbed, and illustrates that all of the components can be roughly segmented into three groups: software infrastructure, logic control, and data storage (which includes the OPC server and database). Data storage plays a central role in the communication flow because most of the information exchanged between the software infrastructure and logic control goes through it, with only one exception, namely the customized information for carving and cutting the part — "letter to be carved" and "depth of cut" — which circumvent the data storage.

A communication flow shown as a double line with arrows illustrates the notification sent from the RFID to the CWM through the Middleware which is in charge of all the communications to and from the CWM. This newly added communication flow will replace the old flow from the RFID to the SLC through trackers.

In order to eliminate the SLC, the tracking system needs to be modified. The communication flow in the old tracking subsystem starts from the OPC servers through the trackers to the SLC. To remove the SLC, one must first determine the new recipient of the notification. In the testbed, the CWM is the only controller at a higher level than the SLC. Therefore, the CWM should handle notification from the tracking subsystem because this is a high level task. However, considering that not all the tracking information needs to be handled by high level logic which is enforced by the CWM, we add an ECA based software module named next-state-mapper on the communication path from the OPC servers to the CWM. The next-state-mapper can process tracking information and also filters it before it goes to the CWM. It receives the events of OPC tag changes. In response to the events, it updates some other OPC tags on the OPC servers and/or updates the database and/or notifies the CWM of the changes. Since the next-state-mapper does not necessarily notify the CWM every time there is an OPC tag change, it serves as an event filter and improves the performance of the system by only passing to the CWM the tracking information which really needs to be processed by high level logic. Therefore, the new flow path starts from the OPC servers through a datahub and the next-state-mapper to the CWM, where a datahub is a software application which collects the OPC tag changes on multiple OPC servers and records the changes in the database. As can be seen from Figure 4, in the old path, the information exchange format between OPC servers and trackers and the information exchange format between trackers and SLC are both OPC. When the new path is established, the information exchange format between OPC servers and datahub is OPC; the information exchange format between datahub and next-state-mapper is SQL queries; the information exchange format between next-state-mapper and CWM is XML.

VI. LESSONS LEARNED FROM CASE STUDY SYSTEM

In planning the consolidation of the case study system and the implementation thus far, several lessons were learned that can be transferred to other ECA consolidation projects. Having the control logic distributed by functionality within a controller makes shifting that logic much simpler. Because the SLC consisted of a group of communicating ECA MFSMs, rather than one all-encompassing ECA MFSM, preparing its methods for push up or push down only required re-routing their communication through the software infrastructure. In this manner, individual ECA MFSMs could be moved independently of one another, allowing the consolidation project to occur incrementally while maintaining

a functioning control system. If the SLC had been a single ECA MFSM, then separating the methods would have been more complicated and might have required that all of the SLC's methods be moved before the control system was functional again.

In pushing down SLC-C1 into the new Cell 1 controller, it was noted that having an ECA control structure made the consolidation process much easier, in agreement with the prediction in [2]. Although the original Cell 1 controller was in MFSM form, the portion being merged with the SLC-C1 to form the resource allocation control part of the new C1C was relatively easily converted into ECA MFSM form. With both the SLC-C1 and portion of the C1C to be moved into the resource allocation control section in ECA MFSM form, merging them involved simply combining their ECA rules and not having to change any of the peripheral modules. Because ECA rules only involve external input as a trigger and external output as a final action, merging two sets of rules from different ECA MFSMs involves two scenarios. First, a rule from one of the source ECA MFSMs is neither triggered by nor triggers any rules in the other ECA MFSM, in which case this rule is added directly to the new rule set. Second, a rule is triggered by the output of a rule from the other ECA MFSM, in which case the first rule is appended to the end of the second, with the previously external output action and input trigger removed, and this rule added to the new rule set. This procedure is executed for all rules in the two ECA MFSMs to be merged. Once the rules are merged into a new rule set, all that is left to merge the ECA MFSMs is to gather their peripheral modules in one file and write a new system description file.

In eliminating the SLC from the tracking information flow, a straightforward solution is to make the CWM take over the processing work of the tracking information previously done by the SLC. However, when we examined the performance of this solution, we found out that not all the tracking information needs to be handled by a high level controller like the CWM. Therefore, we introduced an ECA based software module on the path of the tracking information to the CWM. This software module handles tracking information and filters it before it goes to the CWM. Thus, the traffic to the CWM is reduced and the CWM only needs to handle tasks where the high level logic control is really needed.

VII. Future Work and Conclusions

This paper presented some guidelines to follow when consolidating a control hierarchy to make it conform to the standard structure, and in particular, doing so for an ECA-based system. These guidelines were applied to a case study system, the Reconfigurable Factory Testbed, and some lessons presented from the implementation of the consolidation. Future work includes further formalizing this ECA consolidation procedure and completing the testbed's consolidation to address the problems caused by its broken hierarchy.

Acknowledgments

The authors would like to acknowledge the contributions of all the students on the RFT team, and in particular, those of Krishnakumar Ramamoorthy and Shyam Gala for their work on the RFID conversion process.

References

[1] E. Almeida, J. Luntz & D. Tilbury. Modular Finite State Machines Implemented As Event-Condition-Action Systems. *Proceedings of the 16th IFAC World Congress*, July 2005.

[2] E. E. Almeida, J. E. Luntz, & D. M. Tilbury. Event-condition-action systems for reconfigurable logic control. *IEEE Transatctions on Automation Science and Engineering*, 4: 167-181, 2007.

[3] S. Ceri, R. J. Cochrane & J. Widom. Practical Applications of Triggers and Constraints: Success and Lingering Issues. *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[4] J. Dietrich, A. Kozlenkov, M. Schroeder, & G. Wagner. Rule-Based Agents for the Semantic Web. *Journal on Electronic Commerce Research Applications*, 2003.

[5] E. W. Endsley, E. E. Almeida & D. M. Tilbury. Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation. *Control Engineering Practice*, 14: 1127-1142, 2006.

[6] E. Ferro. Nuclear Supplier: Upgrades Controls, Increasing Efficiency, Production. *Control Engineering*, 53: IP1-IP5, 2006.

[7] S. S. Heragu, R. J. Graves, B. Kim & A. St. Onge. Intelligent Agent Based Framework for Manufacturing Systems Control. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, 32: 560-573, 2002.

[8] J. Montague. Software 'bridge' improves CNC efficiency during retrofit. *Control Engineering*, 50: 18, 2003.

[9] J. Moyne, J. Korsakas, C. Milas, T. Hobrla, T. Hong, H. Kim, J. Priskorn, K. Sukerkar, H. Wijaya, N. Agarwal & D. Tilbury. A Software Infrastructure for Reconfigurable Manufacturing Systems. *Proceedings of the CIRP International Conference on Reconfigurable Manufacturing Systems*, 2003.

[10] J. Moyne, J. Korsakas & D. M. Tilbury. Reconfigurable Factory Testbed (RFT): A Distributed Testbed for Reconfigurable Manufacturing Systems. *Proceedings of the Japan–U.S.A. Symposium on Flexible Automation*, July, 2004.

[11] J. Moyne, D. Tilbury & H. Wijaya. An Event-Driven Resource-Based Approach to High-Level Reconfigurable Logic Control and its Application to a Reconfigurable Factory Testbed. *Proceedings of the CIRP International Conference on Reconfigurable Manufacturing Systems*, 2005.

[12] A. Paschke. The Reaction RuleML Classification of the Event / Action / State Processing and Reasoning Space. White paper, *Reaction RuleML Technical Group*, October 2006.

[13] N. Paton, editor. *Active Rules in Database Systems*. Springer-Verlag, 1999.

[14] A.Tharumarajahm, A. J. Wells, & L. Nemes. Comparison of the bionic, fractal, and holonic manufacturing system concepts. *International Journal of Integrated Computer Manufacturing*, 9: 217-226, 1996.

[15] Whirpool upgrades with servo motor/drive retrofit. *Control Engineering*, 52: 18, 2005.

[16] H. Wijaya, K. Sukerkar, S. Gala, N. Arora, J. Moyne, D. Tilbury & J. Luntz. Reconfigurable Factory-wide Resource-based System Integration for Control. *Proceedings of the IEEE International Conference on Electro/Information Technology*, May 2006.

Appendix of Acronyms

Acronym	Description
C1C	Cell 1 Controller (in testbed)
CWM	Control Workflow Manager (in testbed)
ECA	Event-Condition-Action (paradigm, see [16])
MFSM	Modular Finite State Machine (see [1])
OPC	Open Process Control, opcfoundation.org
RFID	Radio Frequency Identification
SLC	System Level Controller (in testbed)
XML	Extensible Markup Language, xml.org