Implementing a Control System Framework for Automatic Generation of Manufacturing Cell Controllers

Oscar Ljungkrantz*  Knut Åkesson*  Johan Richardsson**  Kristin Andersson*

* Department of Signals and Systems
  Chalmers University of Technology
  Göteborg, Sweden

{ljungkra, knut, kickid}@chalmers.se  jrich103@volvocars.com

** Advanced Equipment Engineering
  Volvo Car Corporation
  Göteborg, Sweden

Abstract— Quickly adapting the manufacturing system to the production of new or modified products is critical for manufacturers in order to stay competitive. For flexible manufacturing systems this typically implies modifications of the control programs. In previous work a framework for automatic generation of cell controllers has been developed. In this paper an implementation of the framework is presented.

Important properties of the presented implementation are: the information from earlier design phases is reused; automatic code generation for faster development and reduced implementation errors; the supervisory control theory is used to generate control functions that are correct by construction; object oriented principles are used in order to allow the reuse of existing library functions. The implementation is generic in the sense that it may generate control programs for a number of target platforms, but in this paper the focus is on generating a control program for the Java platform. An industrial example of a reconfigurable manufacturing cell has been used in the implementation process and shows that the framework is feasible for large manufacturing systems.

I. INTRODUCTION

The life-cycles for many mass-produced products, including automotive products, are constantly shortening due to frequently changing market demands and increased competition. Hence the manufacturing systems need to handle extensions and reconfigurations more frequently. To be able to modify existing equipment and introduce new equipment in the manufacturing system in a fast way, the manufacturing control system also has to quickly be modified and made fully operational [1], [2].

One way to decrease the development time for the control system is to reuse the information from the requirements and design phase of the development process of the manufacturing system. This information can be used to automatically generate important parts of the control system.

Decreasing the time spent on modifying the control program at the shop floor is critical for cost reasons. The use of off-line verification is important because it decreases the time spent on modifying the control program at the shop floor. A related approach is to automatically generate a control function that will be guaranteed to fulfill given specifications, for instance which operations to perform and operation conditions that may not be violated.

Some parts of the control program may be reused in many manufacturing cells, if object oriented design is used that facilitate the development of generic library functions. Those parts of the control program are suitable to arrange into a software component library. Reusing such software components could reduce the development time and the errors, as components already verified to work properly are used.

This paper describes a framework and an implementation of the framework, for developing control programs for manufacturing cells. It utilizes the ideas and methods presented above by having the following four properties:

Property 1: Reusing information from the requirement and design phase of the development process,

Property 2: Using automatic code generation for faster development and reduced implementation errors,

Property 3: Using supervisory control theory [3] to automatically generate control functions that by construction fulfills given specifications,

Property 4: Using object-oriented component libraries for higher reusability and reliability including parts that are not suitable for automatic code generation.

A Programmable Logic Controller (PLC) is an industrial computer that executes given control programs at the shop floor. The main idea of the presented framework is to transform data from the development process of a manufacturing cell into an operational PLC program. To do this, some of the data is used to build an object oriented structure of the control program, and some is used as input to a tool for synthesis that calculates an overall control function of the cell that satisfies the specified requirements. The control program structure, the control function and available PLC software components from a component library are then used to automatically generate a PLC program.

The program structure of the control program is based on previous work by Richardsson [4] and Andersson [5]. Examples of other frameworks and architectures can be seen in [6], [7] and [8]. The main contributions of the work presented in this paper are the following:

- The implementation itself, as glue that makes a whole out of different parts.
The paper is organized as follows: Section II introduces the framework for control program generation. An example that will be used throughout the paper is introduced in Section III. In Section IV the entire framework and the implementation are presented. Conclusions are presented in Section V.

II. OVERVIEW OF THE FRAMEWORK FOR CONTROL PROGRAM GENERATION

This section briefly presents the proposed framework, data and methods for making an operational control program for a manufacturing cell. The framework is designed so that it fulfills the four properties seen in the introduction. The workflow is outlined in Fig. 1.

The Control Information consist of mainly two parts, one describing the manufacturing cell itself (robots, conveyors etc.) and one describing specification and conditions for the operations that are to be performed in the cell. The control information is supposed to be extracted from the mechanical design and other sub processes of the manufacturing cell development process. For instance, it could be converted from an existing information database at the manufacturing company. Some of this information is automatically converted into finite automata [9]. These automata are used to calculate/synthesize a supervisor according to the supervisory control theory [3]. This means that the operations in the different machines in the cell will be coordinated so that the work in the cell is performed according to the specifications and that no operation conditions are violated, for example that two or more machines are not in the same work zone simultaneously.

The control function and the description of the cell are used to generate a Control System Model. This control system model contains most of the data necessary for generating the control program but implies no specific PLC programming language. The control system model is object oriented with a hierarchical structure corresponding to the structure of the physical equipment of the cell. Hence the cell object contains machine objects, one per machine in the cell, that in turn may contain sensor and actuator objects etc., see Fig. 2.

The control function of the cell is also divided into different levels: COPs at the cell level and EOPs at the machines level. The cell object holds information about how to coordinate the operations of the different machines. This information is stored as COPs, Coordinated Operations which are directly extracted from the supervisor described above. The machine objects (except for machines that have their own control system such as many robots) in turn hold information about how to execute each operation. This information is stored as EOPs, Execution of Operations, one per operation, which are extracted from the control information.

From the control system model different PLC programs can be implemented according to IEC 61131 standard [10], [11] the IEC 61499 standard [12], [13] or a conventional high level programming language such as Java. The parts representing the control function (EOPs, COPs etc.) can be automatically generated from the control system model. The other parts of the control program can also be automatically generated if the control information is extensive but normally much of this information is stored in a component library. The approach chosen in this paper is that existing and suitable library components are instantiated while the remaining parts of the program are automatically created from the control system model.

In the work presented in this paper, Java programs are generated to show the feasibility of the framework. In the future more traditional PLC languages could be added as well. However, Java processors suitable for industrial automation are available, for instance [14].

A. Limitations

In this work only the control of single manufacturing cells are considered, not the whole manufacturing control system. A cell consists of multiple, concurrently executing machines, but each machine may only perform one operation at a time. However, if some machine can perform many operations at a time this machine can usually be considered as many sub-machines, one for each simultaneous operation. Certain
machines, for instance robots, often have their own control systems. The framework presented here assumes that those control systems take care of the execution of the different operations, see [15], and the cell control program tells those machines when to perform each operation. The implementation focuses on generation and execution of control functions and does not include, although the concept indeed does, start and stop of the cell, human interaction, communication (with the real or simulated process) details etc.

The different parts and the implementation of the framework will be explained in more detail in Section IV. In the next section an example of a manufacturing cell is introduced that will be used to explain the concepts in Section IV.

III. EXAMPLE OF A MANUFACTURING CELL

In this section an example of a flexible manufacturing cell will be introduced. This example will be used to explain the implementation through out this paper. The example has also been used to test the methods and algorithms of the implementation.

The example cell shown in Fig. 3 is a manufacturing cell at Volvo Car Corporation in Göteborg, Sweden, in which parts are welded to the floor of a car. The cell consists of nine machines that are controlled by one PLC: four robots, two fixtures, two turntables and one conveyor. The opposite sides of the cell perform the same work on the two sides of the car floor. When a new car floor enters the cell the robots R3325 and R3326 pick plates from the turntables and put them into the fixtures. The fixtures fixate the plates and move forward to the floor of the car. The robots R3325 and R3326 change tools to weld guns and then all four robots execute two different welding programs, first they weld previously loaded parts and after that they weld the floor and the plates in the fixtures together. When this welding is done the fixtures release the plates and move back to their home positions. The robots then execute a third welding program that welds parts of the plates that were previously blocked by the fixtures. After that the robots move away, the conveyor moves the floor out of the cell and the work cycle is completed.

IV. FRAMEWORK FOR CONTROL PROGRAM GENERATION

This section step by step presents a framework, data and methods for making a PLC program for controlling a manufacturing cell. It will be shown how the framework fulfills Property 1-4 listed in the introduction.

The whole cell will mainly be controlled by one PLC program that tells all the different machines when to execute which operation. The execution of the operations for machines, such as the fixtures, that do not have own control systems will also be controlled by this PLC program.

The principles for generating the PLC program are outlined in Fig. 4, which can be seen as a more detailed description of Fig. 1. All parts of the figure will be explained below. The solid lines represent what is the focus of this paper. The dotted parts are also included in the framework but they are not part of the prototype implementation presented in this paper. The upper part of the figure, using automata and synthesis, is described in detail in [16].

A. Implementation Basis

The main parts represented by solid lines in Fig. 1 have been implemented in this work.

Control information (Cell Description and EOPs) and the overall control function (COPs), for the specific cell to control, will be parts of the control system model. Hence it is important to define them stringent. The EOPs, COPs and the cell description are stored as XML, Extensible Markup Language [17], files. For the purpose of defining the structure of the XML files, XML schemas, see [18], for the cell description, EOP and COP files respectively, have been developed. Due to restricted space the XML files and schemas are omitted in this paper but the schemas and example XML files are available at [19].

The control system model has been implemented in Java. Having the EOPs, COPs and cell description stored as XML files, defined by XML schemas, their exist a variety of tools, see [18], for translating the files into programming language representations. JAXB, Java Architecture for XML Binding [20], is such a tool that we use for compiling the schemas into Java classes and converting (unmarshalling) the XML files into objects.

A Java PLC program has also been implemented. The structure and the algorithms of these classes have been made general to be used for different kinds of cells with different kinds of machines and components.

Thus we have one set of classes that represents just the structure of the XML files, one set of classes that represents the whole control system model and at this point one set of classes representing PLC programs in Java. These classes have been created once for all cells. For each manufacturing cell to control, the specified XML files are automatically converted into objects of the JAXB created classes. Those objects are used as input to a ControlSystemModelBuilder
The ROP also contains information about product type, type of operation, duration of the operation etc.

The EOPs describe how each operation shall be executed by the corresponding machine. In Table I an example of the EOP for moving the fixtures from home position to work position, with a plate fixed by the clamp, can be seen. The first row of the table is the supposed Initial State of the components. The machine controller checks if the states of the components match the desired state. If they do, it continues with the first action, otherwise an alarm is raised. In Action 1 the machine controller orders the fixation pin to go to state unlocked. When this is fulfilled the machine controller performs Action 2, moving the fixture itself, and so on. The clamp shall be closed during the whole operation holding the plate, which must be in place as indicated by the part sensors.

<table>
<thead>
<tr>
<th>Action</th>
<th>Fixture pos.</th>
<th>Fixation pin</th>
<th>Clamp</th>
<th>Part sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>home</td>
<td>locked</td>
<td>close</td>
<td>on</td>
</tr>
<tr>
<td>Action 1</td>
<td>home</td>
<td>unlocked</td>
<td>close</td>
<td>on</td>
</tr>
<tr>
<td>Action 2</td>
<td>work pos</td>
<td>unlocked</td>
<td>close</td>
<td>on</td>
</tr>
<tr>
<td>Action 3</td>
<td>work pos</td>
<td>locked</td>
<td>close</td>
<td>on</td>
</tr>
</tbody>
</table>

The IL specifies the conditions, states of the components in the cell, which have to be fulfilled before a machine can move an actuator without causing damage to the cell.

The cell description is divided into two parts: physical resources and virtual resources. The physical resources describes the mechanical, hierarchal, structure of the cell. The virtual resources contains information about the collision zones of the cell and variables used by the EOPs in the different machines. The virtual resources depends not only on the structure of the cell but also on the control design. Part of the cell description needed for the example cell can be seen in Fig. 5.

The cell has a number of machines which may or may not have their own control system. If they do not have their own control system we need to specify the Equipment, such as sensors and actuators, in the machine. Equipment, which can be in one of a number of specified states, can be made up by equipment and of Elements. An element represents the lowest physical part of the equipment at which the communication between the control system and the real actuators and sensors take place. As an example consider the clamp group \( A_3 \) in the fixture FIX151. This clamp group actually consists of two clamps which cylinders are controlled simultaneously by one valve. Each clamp has two sensors, one indicating that the clamp is closed and one indicating that the clamp is open. Hence \( A_3 \) is an equipment entity consisting of one element, the valve, and two lower level equipment entities representing the two clamps. Those
Each operation in the COPs has a set of preconditions, which are operations in the current or other machines that has to be performed before this operation can be started. To facilitate concurrent execution of operations in different machines, when allowed, the coordinated operations are organized into different COPs, one per machine and product type. Especially if the cell can work on several products simultaneously this partitioning is useful [23]. A schematic example of a COP can be seen in Fig. 6. The order of the operations, and the preconditions for each operation, for the fixture FIX151 can be seen. For instance the fixture can only move forward to the work position (Op33) when the product is in place (Op21 finished) and the clamps are closed (Op32).

![Schematic picture of the COP for the fixture FIX151 in the example manufacturing cell](image)

**D. Control System Model**

As described in Section II, the control system model contains a representation of the control function (mainly EOPs and COPs) and the cell description. The purpose of the control system model is to have a middle layer that incorporates all the information about the cell and its control and the control system architecture, without stating the implementation details. In this way many different control system implementations, PLC programs, can be implemented from the same control system model, making it possible to quickly adapt the code generation to different companies that have different PLC hardware. Thus the control system model contains EOPs, COPs, the complete hierarchical structure of the cell, equipment states, machine variables etc. plus components specific to the chosen control system structure such as coordinators for controlling the cell and machine controllers for controlling the machines. However nothing is stated about how the different parts shall communicate with each other, or other PLC program specific details.

**E. PLC Program**

As mentioned in Section II, multiple target platforms are supported by the framework, however only the Java platform is supported in this implementation. The Java PLC program generator has been implemented mainly to develop and evaluate the presented framework. In this section the Java implementation will be discussed but many of the principles are suitable for other implementation languages as well, for example IEC 61131 Function Block Diagrams and
IEC 61499. Some comments are also given on how to utilize this approach in building an IEC 61499 implementation.

The PLC program structure is completely based on the program structure described in [4], [24]. Some parts concerning start, stop, human interaction etc. are omitted. Other parts, such as zone and variable handling, which have been extended compared to what is presented in [4], [24], will be discussed in more detail later in this section.

**Main objects:** The main objects of the PLC program are presented in Fig. 7. Each Machine in the cell is controlled by a Coordinator which tells the different machines when to perform which operation according to the COPs. As mentioned above the cell control function is divided into different COPs, one per machine and product type. In a similar way the coordinator uses different Machine Coordinators, one per machine, to handle the active COP for each machine. The machine typically has one COP per product and additional start cycle COPs etc., but only one COP per machine can be active at a time. When a COP is registered to the coordinator it creates a machine coordinator for the relevant machine, if it is not already created, and sets the COP to that machine coordinator.

**Mailbox communication:** All communication between the machine controllers and the machine objects is done by sending messages via the Mailbox. The purpose of the mailbox is that objects such as machines can easily be instantiated and added without having to rewrite too much. This approach is proposed in [24] and is suitable for the Java implementation. For other implementations, however, other means of communications may be more appropriate, for instance when implementing IEC 61499 programs direct data and event connections may be an alternative.

**Machine object:** Each machine object consists of subcomponents, the main parts can be seen in Fig. 8. The control of the machine is handled by the Machine Controller. The machine controller performs the operations by changing the states of the machine step by step by sending messages to objects representing the components Actuators, Sensors and Variables in the machine. When the machine coordinator tells the machine to perform an operation, the machine controller in the machine loads the corresponding EOP that defines the order of the states for that operation. The machine controller communicates with the components via a mailbox, for the same reasons as for the cell mailbox.

**Different messages when performing an EOP:** The machine coordinator communicates with the actuators and sensors by sending different types of messages:

- **Request State** asks which state the component is in. It is needed in the case where more than one alternative EOP are given for a specific task. In those cases the EOP that is to be executed is given by the initial states of the machine which are checked by sending request state messages to the components.

- **Check State** asks the component if it is in the desired state, otherwise the actuator or sensor is to alarm. The message type is needed when the first row of an EOP is to be executed to check that the states of the components match the desired states.

- **Order State** tells the component to go to the desired state and report when it is done. The component should alarm if the desired state is not received within a specified time. Actuators send an alarm if the wrong state is reached. The sensor cannot really go to a new state by itself, so in case of the sensor this operation rather means waiting for the state to change. The order state message is used when the state of a component changes in an action in the EOP. For instance, the fixture positioning actuator $A_1$ changes from home to work position state in action two in the EOP seen in Table I.

- A **Monitor State** message is also needed for sensors. It tells the sensor to raise an alarm if the state is changed before the state monitoring is turned off. It is turned on at the initial row of an EOP and is turned off before a new state is ordered. When the state has changed the state monitoring is turned on again and so on, and when the operation is finished it is turned off. For actuators this message type is not needed since the state of the actuators is monitored all the time, even when not performing an operation, and shall never change unless they are told to by a state order.

In [25] the different messages are elaborated on and an implementation of sensors and actuators in IEC 61499 is described.

**Component hierarchy:** Actuators can have an inner hierarchy consisting of actuators and sensors e.g. the clamp group $A_3$ in the fixtures, described in Section IV-B above.
To reduce the complexity of the EOPs, the machine controller only controls the top level actuators and sensor components, such as the clamp group $A_3$ and the actuator control object representing $A_3$ in turn controls the inner, lower level objects. The lower level components are checked by the upper level components by a request state method. Lower level actuators also have a state order method.

Variables: Some machines also have internal variables. An example of a variable is “new rack in cell” in the turntables which indicates a new rack of plates on the side of the turntable facing the cell and is needed for the robots to determine which plate to pick from the rack. The variable is set to true when the turntable is rotated and set to false when the first part is picked from the turntable by a robot. The robot cannot set the variable of the turntable by itself so this is done in a turntable EOP that is scheduled to be performed after the robot has picked the first part. In this implementation the variables are treated the same way as sensors and actuators, that is they respond to request state, check state and order state messages, sent via the mailbox. The reason for this choice is 1) to be consistent in the implementation and 2) to prepare for a distributed implementation.

Zone handling: The cell PLC program also consists of Zone objects. The zone objects represent areas in the manufacturing cells that are common for some machines but in which only one machine can be in at any time, to prevent collision. A zone must be booked by a machine before entering it and is unbooked when left by the machine. The COPs are synthesized so that more than one machine will never try to be in a zone at the same time in automatic mode. In manual mode the operators shall be prevented to enter an occupied zone, as specified in the IL. Hence, to be able to go from automatic to manual mode, the zones must be booked and unbooked also in the automatic mode. Therefore, in this implementation the states of the relevant zones are included in the EOPs. The state of the zones in the perspective of a single machine is “booked (by me)” or “unbooked (by me)”. Each zone is represented as an object connected to the cell mailbox, see Fig. 7, and knows which machine has booked the zone or if it is unbooked. When a zone is to be booked by a machine, the machine controller that reads the EOP tells the machine object to send a message, via the cell mailbox, to the corresponding zone. This way the zones are treated almost the same way as other components, which too is consistent and suitable for distributed control systems.

Using library components: The parts representing the application specific control function (EOPs, COPs etc.) are automatically generated from the control system model. For the other parts of the control program, representing the components of the cell, component details etc., different solutions are possible. For instance, they can also be automatically generated but that requires really extensive control information. Another alternative is to automatically generate some parts and to manually add missing parts. However, the manual part may introduce errors. Both the above alternatives may also be inefficient if similar code instead could be reused from previous projects. Thus, a third alternative is recommended in this paper. We propose to use existing and suitable library components and to automatically create the rest of the program from the control system model. This fulfills Property 4. Those library components must comply with the structure presented above. For instance, a machine object taken from a library must be able to read EOPs and to be able to communicate with the coordinator and a sensor object must be able to communicate with the machine controller via the message types request state, check state, order state and monitor state.

General algorithms: The library components indeed affects the amount of control information that needs to be specified. For instance if a whole or a part of a machine corresponds to an existing library component the subcomponents do not have to be specified in the cell description. However, one of the goals for the Java PLC program has been to develop general algorithms that can be used for as many different cell, machines and components as possible. The algorithms can also be used as a help when building a component library. The developed algorithms are general enough to handle different components of the example cell, as specified in the cell description [19]. For making a complete Java PLC program, communication (with the real process) details and some alarm and time handling etc. also has to be defined. An example of a general algorithm that can be used for many different kind of components is the general actuator request state algorithm, which can be seen below:

```java
procedure REQUESTSTATE
    if lowerLevelActuators ≠ null then
        currAct ← getFirstActuator()
        currState ← currAct.requestState()
        while hasMoreLowerLevelActuators do
            currAct ← getNextActuator()
            if currState ≠ currAct.requestState() then
                broken equipment, alarm!
                return null
            end if
        end while
    else if lowerLevelSensors ≠ null then
        currSens ← getFirstSensor()
        while hasMoreLowerLevelSensors do
            currSens ← getNextSensor()
            currSensState ← currSens.requestState()
            if knownState(currSensState) then
                if NOT stateFound then
                    stateFound ← true
                    currState ← currSensState
                end if
                if currState ≠ currSensState then
                    broken equipment, alarm!
                    return null
                end if
            else if currState ≠ currSensState then
                broken equipment, alarm!
                return null
            end if
            if NOT stateFound then
                broken or moving equipment!
                return null
        end if
    end if
end procedure
```

The code that requests the states of the sensors may seem a bit complex, but imagine an actuator that can be in one of two states A and B and that has two sensors, one in each position. The two sensors have the possible states A and “not
A” and B and “not B” respectively, where only A and B is known by the actuator. A and B shall not be given at the same time, then the equipment is broken. “not A” and “not B” at the same time indicates a broken or moving actuator. The presented algorithm works for many types of actuators and sensors but not for all types. For instance some safety sensors are coupled so that a signal is given if two out of three sensors are high. If such a sensor is used in the cell, a library component for this sensor has to be available.

V. CONCLUSIONS AND FUTURE WORK

In this paper an implementation of a framework for generating control programs for a manufacturing cell has been presented. The framework is suitable for the example manufacturing cell and the implementation has been made general and may be applied to other application areas as well. The framework reuses information from the development process of the manufacturing system to automatically create a control program. Supervisory control theory is used to guarantee that the control program fulfills the specification. The implementation structure is object-oriented and hierarchical to make it easy to read and maintain and to use component libraries for parts that are not convenient to automatically create from specifications.

These properties are intended to make it faster to develop new operational control programs for manufacturing cells. Although this work only focuses on control programs for manufacturing cells, and not the whole manufacturing system, it is an important part that might speed up the development of the entire control system. Being able to faster develop and change the control programs of manufacturing systems should make it faster to reach full productivity, thus leading to higher productivity at the manufacturing company.

Those aspects have been considered before, by for instance Richardson [4] and Andersson [5]. This work contributes an implementation that aims to test and unite this research. It also extends previous work by defining information and providing methods and algorithms for automatic generation of manufacturing cell PLC programs.

In the future we plan to add an IEC 61499 program generator from the control system model, thus adopting this new standard for distributed control systems. So far we have only executed the generated control program against a simulated plant, in the future we plan to also evaluate the framework in a physical plant.

ACKNOWLEDGEMENT

The authors would like to thank the ProViking programme [26] for financing this work.

REFERENCES