2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

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

** Advanced Equipment Engineering
Volvo Car Corporation
Göteborg, Sweden
jrich103@volvocars.com

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 of 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 the manufacturing system in a fast way, the manufacturing control system also has to be quickly 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. This may be accomplished using off-line verification. A related approach is to automatically generate a control function that will be guaranteed to fulfill given specifications, such as 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. 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 uses 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 formal methods to generate control functions that by construction fulfill given specifications,

Property 4: Using general and object-oriented component structure for higher reusability and reliability.

An example of a flexible manufacturing cell, shown in Fig. 1, has been used to test the methods and algorithms of the implementation. It will also be used to explain the implementation through out this paper.

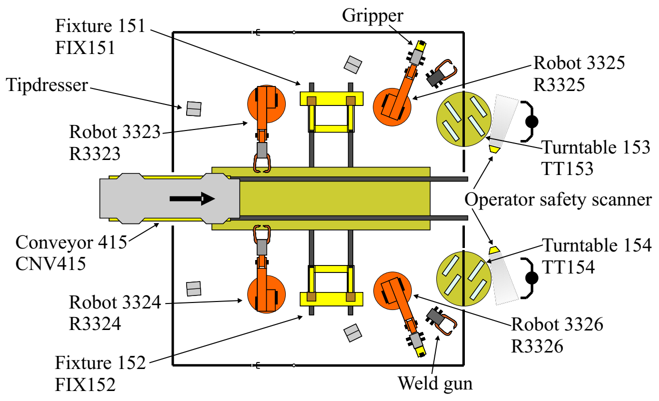

Fig. 1. An example of a manufacturing cell in which parts are welded to the floor of a car. The cell is an existing manufacturing cell at Volvo Car Corporation in Göteborg and consists of nine machines that are controlled by one PLC: four robots, two fixtures, two turntables and one conveyor. [4]

A Programmable Logic Controller (PLC) is an industrial computer that executes given control programs on 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 found 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.
- Definitions of the information, in terms of input files and formats, needed for the framework.
- Methods and algorithms for automatic generation of the PLC program.
- Extensions and modifications of the program structure.

The paper is organized as follows: Section II introduces the workflow for control program generation. In Section III the entire framework and the implementation are presented. Conclusions are presented in Section IV.

II. WORKFLOW FOR CONTROL PROGRAM GENERATION

This section introduces the proposed framework by describing the main workflow for making an operational control program for a manufacturing cell. The principles of the framework are outlined in Fig. 2.

Control Information from the development process of the manufacturing cell is used to create a *Control System Model* which is a representation of the equipment and the control function of the cell. Formal methods are used to synthesize a cell control function that fulfills the specification. The control system model is used, together with existing component library to create the PLC program.

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 of the cell and from robot simulations. Some of this information is automatically converted into finite automata. 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. 3.

The *control function* of the cell is divided into different levels: *COPs*, *Coordinated OPerations*, at the cell level and *EOPs*, *Execution of OPerations*, at the machines level. In the cell object the COPs hold information about how to coordinate the operations of the different machines. The COPs are directly extracted from the supervisor described above. In the machine objects the EOPs, one per operation, in turn hold information about how to execute each operation. The EOPs are extracted from the control information.

From the control system model different PLC programs can be implemented. The parts representing the control function can automatically be generated from the control system model. The other parts of the control program can also be 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 this work, Java programs and an *IEC 61499* [9] prototype are generated to show the feasibility of the framework. In the future customary *IEC 61131* [10] PLC languages could be added as well.

A. Restrictions

In this work only the control of single manufacturing cells are considered, not the whole manufacturing control system.

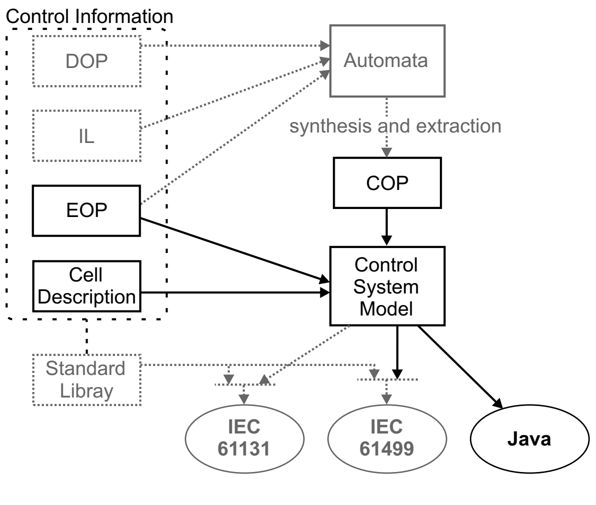

Fig. 2. The main principles of the presented framework for generating PLC programs for manufacturing cells. All parts of the figure are explained in Section III. The solid lines represent what is the focus of this paper. The dotted parts are also included in the framework but they are not part of the presented implementation.

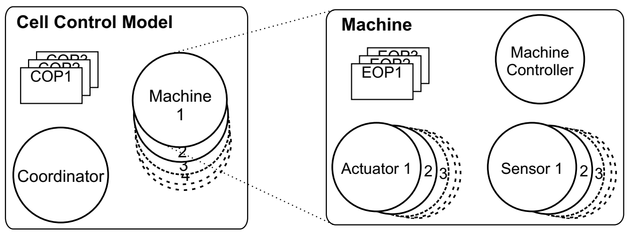

Fig. 3. The main objects of the control system model. The cell control model consists of machine objects and control information. The machine objects in turn consist of objects representing components, such as actuators and sensors, and information on how to execute each operation.

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 presented framework assumes that those control systems take care of the execution of the different operations, see [11], and the cell control program tells those machines when to perform each operation. The implementation focuses on automatic control of the cell and does not include, although the concept indeed does, start and stop of the cell, human interaction, alarm and time handling etc.

III. FRAMEWORK AND IMPLEMENTATION FOR CONTROL PROGRAM GENERATION

The framework, data and methods for making a PLC program for controlling a manufacturing cell are presented in this section. The framework, outlined in Fig. 2, will be explained step by step and 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 executions 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 main parts represented by solid lines in Fig. 2 have been implemented in this work, in Java. To facilitate automatic code generation the formats of the EOPs, *Cell Description* and the overall control function (COPs) have been clearly defined, using XML schemas [12]. Due to restricted space the XML schemas and XML files for the example cell are omitted in this paper but are available at [13]. Having this defined formats, the files for a specific cell can be automatically converted into objects of the control system model, which in turn can be converted into objects of the PLC program. This fulfills Property 2.

A. Control Information

The *Control Information* consists of *DOP* (Declaration of OPerations), EOPs, *IL* (InterLocks) and Cell Description, and is supposed to be extracted from the development process of the manufacturing system. In a study at Volvo Car Corporation [14] it was shown that most of the information for generating a manufacturing PLC program can be extracted from the mechanical design of the cell. Furthermore, robot simulation can be used to decide which collision zones to use, and when each machine needs access to the zones. Since all this information might be stored in different and possibly company specific systems, the control information specified in this paper is supposed to be extracted from those systems. This fulfills Property 1.

An operation is a set of actions in one machine which are chosen so that interaction with other machines is not needed during the execution of that operation. The DOP holds information about when an operation in the cell can be executed. Here natural predecessor relations between the different operations are stated, for instance that the robot has to have changed tool to the weld gun and that the floor and the plate has to be in place, before the robot can weld the plate to the floor of the car. The DOP 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 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 I
EOP (PART OF) FOR THE OPERATION OF MOVING THE FIXTURES FROM HOME POSITION TO WORK POSITION.

	internal components				
	A_1 Fixture pos.	A_2 Fixation pin	A_3 Clamp	S_1 Part sensors	S_2
Initial state	home	locked	closed	on	on
Action 1	home	unlocked	closed	on	on
Action 2	work pos	unlocked	closed	on	on
Action 3	work pos	locked	closed	on	on

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 describe the mechanical, hierarchical, structure of the cell. The virtual resources contain information about the collision zones of the cell and variables used by the EOPs in the different machines. Hence, the virtual resources depend not only on the structure of the cell but also on the control design.

Part of the physical resources needed for the example cell can be seen in Fig. 4. The cell has a number of machines. 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 clamp equipment contain no elements, since one clamp can not be actuated alone, but consist of two equipment entities, one representing each sensor.

```
-Physical Resources - Example cell
  -Machines
    Robot - R3323, hasOwnControlSystem: Yes
    -Fixture - FIX151, hasOwnControlSystem: No
      -Equipment
        -Sensor - S1, part sensor
          -States
            on
            off
          -Elements
            Element - FIX151S1
        +Sensor - S2, part sensor
        -Actuator - A3, Clamp Group
          -States
            open
            closed
          -Elements
            Element - FIX151A3, bistable valve
          -Equipment
            -Actuator - A3A, Cylinder clamp 1
              -States
                open
                closed
              -Equipment
                -Sensor - A3AS1, Sensor clamp 1 open
                  -States
                    open
                    not open
                  -Elements
                    Element - FIX151A3AS1
                +Sensor - A3AS2, Sensor clamp 1 closed
            +Actuator - A3B, Cylinder clamp 2
```

Fig. 4. Physical resources. Part of the cell description of the example cell.

B. *Coordinating the operations - building the COPs*

The cell control function is rather complex since it has to handle different concurrently executing machines, collision zones and other conditions that must not be violated. It may also be more complicated to test the control of the entire cell, compared to testing a single machine. For those reasons we automatically create the COPs, the coordinated operations, that by construction are guaranteed to fulfill the conditions.

The information in the DOP, the EOPs and the IL can be extracted and processed into finite automata. These automata are used to synthesize a supervisor for the cell according to the supervisory control theory, SCT, of Ramadge and Wonham [3]. From this supervisor automaton the relevant information is extracted and presented as COPs. The SCT tool *Supremica*, see [13] and [15], is used to perform the synthesis. The procedure for creating the COPs is described in detail in [5]. Below the main principles are explained.

Five different types of automata are needed: operation models, machine models, relation specifications, zone models and safety specifications. The operations are extracted from the DOP and are each modeled using two events, one controllable event representing the start of the operation and one uncontrollable event representing the finish of the operation. The machines are also extracted from the DOP and are modeled as simple two-state automata stating that the machine may only perform one operation at a time. The relation specifications too are extracted from the DOP and describe the order in which the operations are specified to be performed. The zone models, which are extracted from the EOPs, state that only one machine can access each zone at a time. They have one state indicating that the zone is free and one state for each machine that may book the zone. Finally, the safety specifications are extracted from the IL and the EOPs, and model restrictions on the execution of each operation. As seen in Table I an operation consists of a number of actions. For an operation to be allowed to be executed all interlocks for all actions must be fulfilled. If an operation violates the interlocks for any of the actions in another operation, the two operations are not allowed to be executed at the same time.

All extracted automata (the operation models as plant automata and the rest as specification automata) are used to perform the synthesis, resulting in a supervisor. From the supervisor the COPs are extracted. The algorithm for the extraction, for each operation lists all states in the supervisor in which that operation could be started. From these lists, restriction expressions are built, which are simplified and presented as COPs. This fulfills Property 3.

Each operation in the COPs has a set of preconditions, which are operations in other machines that have to be performed before this operation can be started. The preconditions have either been specified in the DOP or added by the synthesis. 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 [5]. A schematic example of a COP can be seen in Fig. 5.

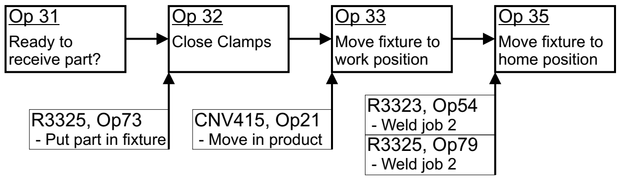

Fig. 5. Schematic picture of the COP for fixture FIX151 in the example cell. The operations must be performed in the order from left to right and some of the operations have preconditions which state that some operations in other machines have to be performed before this operation can be started. For instance the fixture can only move forward to the work position (Op33) when the product is in place (Op21 finished by the conveyor) and the clamps are closed (Op32).

C. *Control System Model*

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 made 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.

D. *PLC Program*

Multiple target platforms are supported by the framework, however only the Java platform is fully 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. The PLC program structure is based on the program structure described in [4]. Some parts, such as zone and variable handling have been extended, as presented later in this section.

Main objects: The main objects of the PLC program are presented in Fig. 6. 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. Likewise, 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 not yet created, and sets the COP to that machine coordinator.

Mailbox communication: All communication between the machine coordinators 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. Examples of message types are performEOP and EOPDone. All messages include the name of the receiver and the mailbox is implemented so that it delivers the messages to the receiver.

Machine object: Each machine object consists of subcomponents, see Fig. 6. 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 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 four different types of messages: *Request State* - asks which state the component is in; *Check State* - asks the component if it is in the desired state, otherwise an alarm is raised; *Order State* - tells the component to go to the desired state and report when it is done; *Monitor State* - an alarm is raised if the state is changed when the state monitoring is on. In [16] the different messages are elaborated on and an implementation of sensors and actuators in IEC 61499 is described.

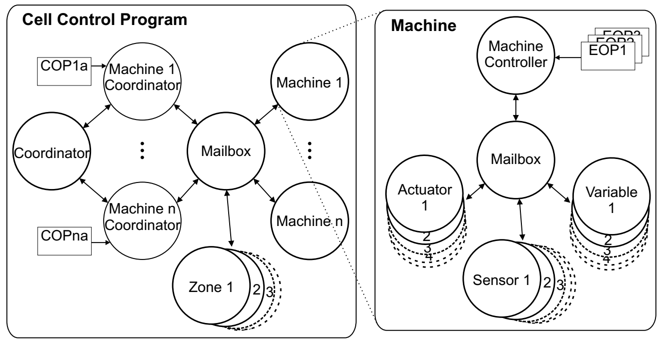

Fig. 6. The main objects of the manufacturing cell PLC program. To the right, the main sub-objects of the machine object are presented as well.

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 III-A 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 have internal variables, corresponding to relevant sub-states of the machine not represented by an actuator or a sensor. 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. 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. The reason for this choice is to be consistent in the implementation and to prepare for a distributed implementation.

Zone handling: The cell PLC program also consists of *Zone* objects. The zone objects represent volumes 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, in automatic mode, more than one machine will never try to be in a zone at the same time. In manual mode the operators shall be prevented to enter an occupied zone, as specified by 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. 6, and knows which machine that 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.

General algorithms and component libraries: One of the goals for the Java PLC program has been to develop general algorithms that can be used for as many different cells, 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 all different components of the example cell. For making a complete PLC program, 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 actuator request state algorithm, which can be seen below:

```
procedure REQUESTSTATE
    if any lowerLevelActuators then
        currentState ←
                getFirstLowerLevelActuator().requestState()
        for all lowerLevelActuators do
            if (currentLowerLevelActuator.requestState()
                ≠ currentState) then
                broken equipment, alarm!
            end if
        end for
    else if any lowerLevelSensors then
        stateFound ← false
        for all lowerLevelSensors do
            currentSensorState ←
                currentLowerLevelSensor.requestState()
            if knownState(currentSensorState) then
                if NOT stateFound then
                    stateFound ← true
                    currentState ← currentSensorState
                else if currentState ≠ currentSensorState
                    then
                    broken equipment, alarm!
                end if
            end if
        end for
        if NOT stateFound then
            broken or moving equipment!
        end if
    end if
    // For actuators with no sensors (rare)
    // the state is the last ordered state
    return currentState
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, however only A and B are 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 indicate 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.

We propose to use existing and suitable library components for parts that can be reused between cells and to automatically create the rest of the program from the control system model. Those library components must also comply with the object-oriented structure presented above. This fulfills Property 4.

IV. 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 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 operational control programs for manufacturing cells. Being able to faster develop and change the control programs of manufacturing cells should make it faster to reach full productivity, thus leading to higher productivity at the manufacturing company.

This work contributes an implementation that aims to evaluate and develop the framework. It also extends previous work by defining information and providing methods and algorithms for automatic generation of PLC programs.

The Java PLC program for the example cell has been tested in detail on a text-based simulation of the cell. The Java program has also been used as a basis for an IEC 61499 program generator prototype which has been used to control a 3D simulation of the example cell. In this prototype the operations of the machines where simulated and controlled, but not detailed movement of the actuators and sensors involved. The 3D simulation can be seen at [13]. Both simulations have shown that the framework is suitable for industrial manufacturing systems. In the future we plan to further develop the IEC 61499 program generator.

ACKNOWLEDGEMENT

This research is financed by the ProViking research programme.

REFERENCES

[1] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.

[2] Valckenaers, P., V. Brussel, H., M. Kollingbaum, and O. Bochmann, "Multi-agent coordination and control using stigmergy applied to manufacturing control," ser. Lecture Notes in Computer Science, vol. 2086/2001. Springer, 2001, pp. 317–334.

[3] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[4] J. Richardsson, "Development and verification of control systems for flexible automation," Licentiate thesis, Dept. of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, 2005.

[5] K. Andersson, "Hierarchical control and restart of flexible manufacturing systems," Licentiate thesis, Dept. of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden, 2006.

[6] E. W. Endsley, E. E. Almeida, and D. M. Tilbury, "Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation," *Control Engineering Practice*, vol. 14, no. 10, pp. 1127–1142, 2006.

[7] K. C. Thramboulidis, "Model integrated mechatronics—towards a new paradigm in the development of manufacturing systems," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 1, 2005.

[8] R. R. Lindeke, J. Kapla, A. Stadtler, and D. Green, "Developing a virtual process controller—the missing link in the automated manufacturing process tree," in *Flexible Automation & Intelligent Manufacturing*, 2005.

[9] IEC, "IEC 61499-1: Function blocks—part 1: Architecture," International Electrotechnical Commission, Tech. Rep., 2005.

[10] R. W. Lewis, *Programming industrial control systems using IEC 1131-3 Revised edition*. The Institution of Electrical Engineers, 1998.

[11] T. Nordin, "Off-line programming at Volvo Cars—the technique," in *33rd International Symposium on Robotics*, 2002.

[12] "XML schema." [Online]. Available: http://www.w3.org/XML/Schema

[13] "Supremica." [Online]. Available: http://www.supremica.org

[14] A. von Euler-Chelpin, , P. Holmström, and J. Richardsson, "A neutral representation of process and resource information of an assembly cell—supporting control code development, process planning and resource life cycle management," in *2nd International Seminar on Digital Enterprise Technology*, Seattle, USA, 2004.

[15] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proceedings of the 8th Workshop on Discrete Event Systems*, Ann Arbor, MI, USA, 2006, pp. 384–385.

[16] G. Cengic, O. Ljungkrantz, and K. Åkesson, "A framework for component based distributed control software development using IEC 61499," in *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, 2006.