FORMAL SYNTHESIS, SIMULATION AND AUTOMATIC CODE GENERATION OF SUPERVISORY CONTROL FOR A MANUFACTURING CELL

Yuri Garcia Silva, ygarcia@das.ufsc.br
Max Hering de Queiroz, max@das.ufsc.br
DAS – Universidade Federal de Santa Catarina, CEP 88040-900 – Florianópolis SC, Brazil

Abstract. *This paper presents an application of the supervisory control theory (SCT), initiated by Ramadge and Wohnam in the 80s, to coordinate a flexible manufacturing cell (FMC) controlled by a programmable logic controller (PLC). We show how the SCT associated with simulation and automatic code generation can help the control designer to shift focus from synthesis and implementation to the modeling stage. In SCT we look for a minimally restrictive supervisor that monitors the actions of a real system and disables those operations which would lead to an undesirable condition for the system. Such approach offers a formal solution and its results achieve the most flexible control for the plant. In this application we used the local modular approach, proposed by Queiroz and Cury in 2000, which consists of dividing the mentioned supervisor into further ones to resolve specific problems in the system, thus reducing the computational cost to obtain the controller. In the FMC test bed, workpieces coming from an input buffer or a rework station are transported by a robotic manipulator and a rotating table through the drilling, welding and quality test stations. After testing, workpieces can be approved, reworked or discarded if they are rejected twice. We modeled the FMC behavior and its specifications using generators. From these models we synthesized the supervisory control with the TCT tool and next we used the emulator DESEM to simulate the execution of the supervisors on the modeled plant. This procedure allowed us to perceive the necessary changes to the specifications and plant model. When the simulation was working in the expected way we generated a PLC code in structured text for the supervisors using an automatic code generator - Ides2ST. In the implementation we applied a three level hierarchy, proposed by Queiroz and Cury in 2002, to make the interaction between the modular supervisors and the real system. Simulation made possible the debugging of the supervisor in the modeling stage, thus the errors detected during the implementation were restricted to the cell's operational sequences. The automatic code generation reduced the implementation time of the supervisor code in PLC and transformed the supervisor implementing problems into modeling ones.*

Keywords*: supervisory control; manufacturing systems; control system synthesis; programmable logic controller*

1. INTRODUCTION

Automated manufacturing systems are supported by supervisory control systems which are responsible for the monitoring and proper execution of the production line. They determine what actions should be avoided by the cell in order to keep the safety and sequencing of the system. The Supervisory Control Theory (SCT) (Ramadge and Wonham, 1989) offers a formal approach based on controlled automata to generate a monolithic supervisor to rule the system according to a map of control law. Summarily, the development of a supervisory controller considers a global map of possible physical actions to be taken by the real system – open loop behavior – and, according to desired behavioral specifications, excludes those sequences that stand against the wanted control logic. This approach achieves the most flexible control for the plant since it allows the happening of all actions that don't oppose the specified behavior.

When dealing with complex problems, the supervisory controller development faces high computational costs and sometimes can't calculate the optimal supervisor. This occurs because the states number of the global system model increases exponentially with the number of subsystems. Furthermore, if the supervisor was calculable, its implementation would be unfeasible since the PLC memory wouldn't support such a complex supervisor and the available minimization algorithms (Su and Wonham, 2004), (Vaz and Wonham, 1986) would demand high computational costs as well.

The local modular expansion of the SCT (Queiroz and Cury, 2000) proposes the development of several supervisors, every one responsible for a behavioral specification, that together co-operate for the entire system monitoring and control. This approach, instead of finding the supervisor based on a global composite system, specifies various local plants. Each of them is a composite system of the subsystems affected by the local constraints. The computational complexity of the supervisors synthesis process is reduced and their limited size makes possible the supervisors reduction.

In this paper we exemplify the SCT local modular approach application in the development of a controller for a flexible manufacturing cell test bed. Further we explain the implementation of the supervisors set in a PLC according to a hierarchical structure (Queiroz, 2004) that maintains the SCT original characteristics. Contrasted to the previous work (Queiroz and Cury, 2002), our proposal brings the support of a simulation tool linked to the modeling stage and the assistance of an automatic code generator (Klinge, 2007) in the implementation stage. They both add reliance and agility to the controller development and implementation.

This paper is organized as follows. In Section 2 we describe the physical characteristics of the FMC and the behavioral specifications we want the cell to assume. We introduce in Section 3 some theoretical concepts of the SCT and expose the system modeling and controller synthesis. Next, in Section 4, we comment the role of simulation in the work, present the hierarchical structure of implementation and describe how the automatic code generator works.

2. THE FMC

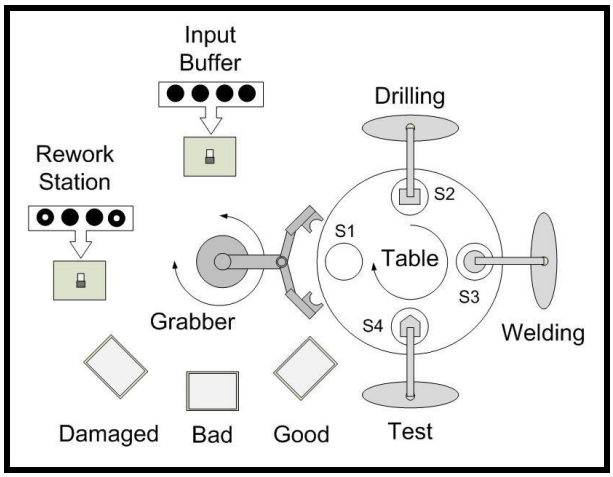

Figure 1. Flexible Manufacturing Cell test bed schematic diagram

The FMC test bed, Fig. 1, is composed of four operational stations that are linked by a rotating table and a grabber. These stations perform the drilling, welding, quality test and rework of workpieces. The FMC main goal is to drill and weld raw workpieces that come from an input buffer. Naturally, as in a real industry, workpieces sometimes are not manufactured correctly at first and they need to be refurbished. Therefore the FMC rework station is an input for manually restored workpieces. Three different workpieces storages are available in the system. The drilling, welding and test stations don't have any sensor to identify the arrival of a workpiece at their stations.

It's possible to implement a wide range of control laws to the FMC depending on how the designer wants it to behave. It could be desired, for example, a manufacturing sequence that allowed the use of a unique slot of the table. Since we want the cell to be as efficient as possible (regarding to the utilization of resources), provided that it respects the security restrictions, we adopted a manufacturing pattern that enables the FMC to use all table slots simultaneously. Given that the FMC main stations don't hold any workpieces arrival sensor, the proposed control must identify the workpieces position by the events history. In the next three paragraphs we explain the behavior proposed for the FMC and how it should act under certain conditions.

When a rough workpiece arrives at the input buffer the grabber drops it on the table in the position S1 since this slot is not being occupied by any other workpiece. The rotating table will conduct the raw workpiece to the drilling, welding and quality test stations and finally will return to the table initial position.

Whenever a workpiece arrives at the rework station the grabber will put it upon the table and the latter will make it reach the quality test station, and later the position S1, so that the drilling and welding processes will not be activated while the remanufactured workpiece is in the position S2 or S3.

Reaching the position S1 for the second time, the grabber takes the workpiece off the table. The quality test will determine the place where the grabber must put the workpieces. The ones manufactured properly are dropped at the "good" storage, no matter what input they came from. If those arisen from the input buffer are rejected, they are taken to the "damaged" storage in order to be reworked manually. Finally, the workpieces must be discarded if they are rejected when coming from the rework station because they failed the test twice, so they are put into the "bad" storage.

In order to obtain the described behavior for the FMC we have to pay attention to some problems that might happen when manufacturing several pieces at the same time. The table cannot turn if it's empty, while any of the stations is performing a task or in case the workpieces on the table haven't been drilled, welded – or skipped both actions –, tested or removed by the grabber. We still have to prevent the system from drilling, welding, testing and grabbing if there's no workpiece waiting to be served at that slot. Also the FMC can't carry out two tasks on the same workpiece and those ones coming from the rework station can't be drilled and welded.

3. SYNTHESIS OF SUPERVISORS

This stage counted on the support of IDES (Rudie, 2006) for the graphical modeling of the FMC subsystems and specifications, and TCT (Feng and Wonham, 2006) for the synthesis of supervisory controls.

3.1. Subsystems and specifications modeling

The modeling of discrete event systems (DES), in accord with (Ramadge and Wonham, 2008), is made by means of generators. A generator, which is an automaton, represents a DES specified by a regular language. It is a 5-tuple of form $G = (\Sigma, Q, \delta, q_0, Q_m)$. In this sense, Σ is the events set, Q is the states set, $q_0 \in Q$ is the initial state, $Q_m \subseteq Q$ is the subset of marker states and $\delta: \Sigma \times Q \rightarrow Q$ is the partial transition function defined in each state of Q for a subset of Σ. For control purpose the discrete events ($\sigma \in \Sigma$) are disjointed in two subsets $\Sigma = \Sigma_c \cup \Sigma_u$. The controllable events ($\sigma \in \Sigma_c$) may be enabled or disabled by an external agent and the opposite applies to the uncontrollable events ($\sigma \in \Sigma_u$). The DES generator recognizes the closed behavior and marked behavior languages. The closed behavior language, L(G), represents all possible event sequences in the generator starting from the initial state. The marked behavior language, $L_m(G)$, indicates all those sequences that reach a marked state starting from the initial state, meaning a task-completion language.

The generators proposed for the FMC are graphically visualized by state transition graphs. The graph vertices symbolize the generator states and the arcs, associated with the partial transition functions, represent the events. The initial state has an arrow pointing at it and the marker states are drawn as a double circle. Controllable events are indicated by a tick on their transitions arcs.

The modeling stage begins with the identification of the subsystems comprised by the FMC. Each subsystem generator is modeled separately using IDES, as shown in Fig. 2. The control problems of the FMC, referred in Section 2, relate to situations that might occur during the interaction of the subsystems and getting an abstract model wouldn't interfere in the controller performance. Therefore, the subsystems models basically involve the beginning and ending of operations.

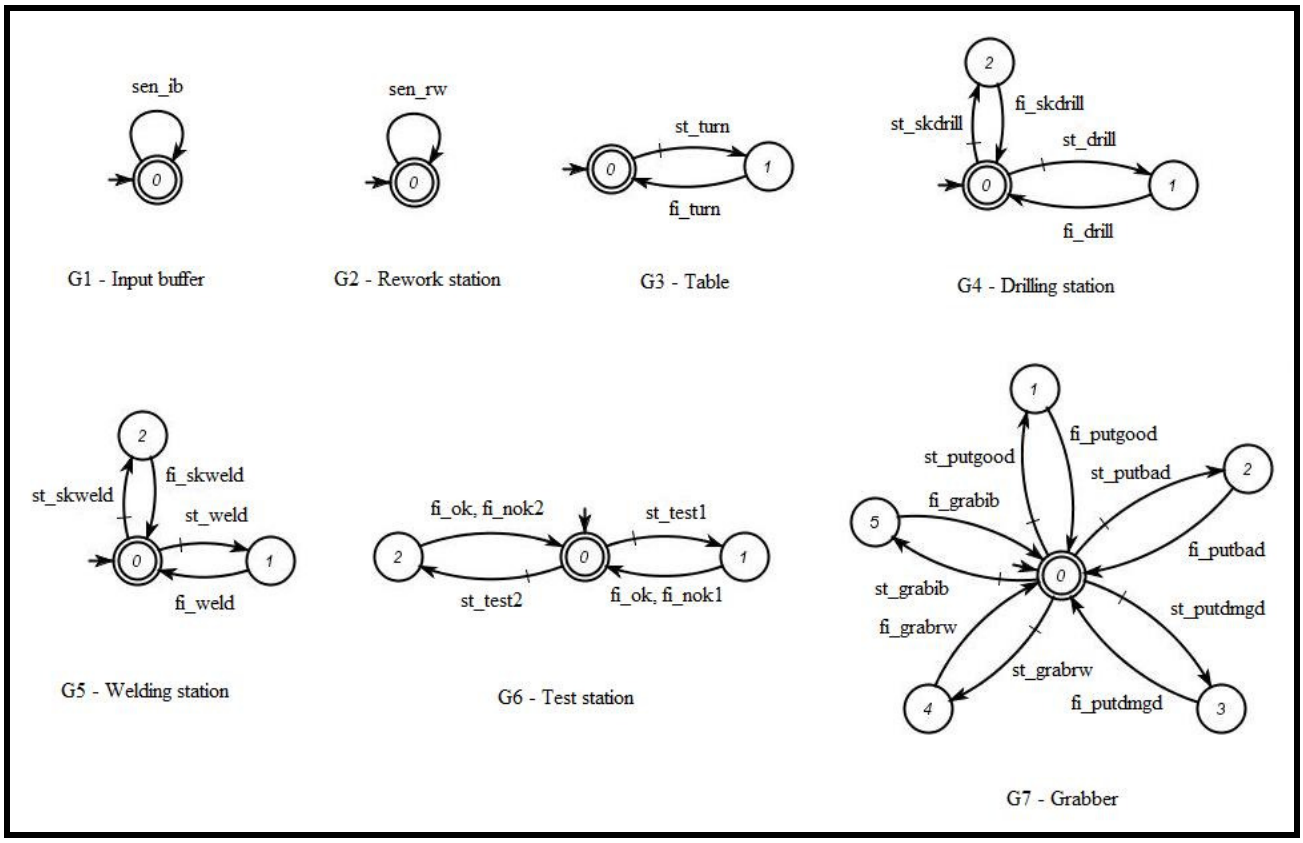

Figure 2. Generators for the FMC subsystems

The input buffer and rework station have sensors that signalize the arrival of workpieces through events sen_ib and sen_rw respectively. The table starts turning by the command st_turn and when it finalizes the signal fi_turn is sent. The drilling and welding station work similarly to the table, except for the possibility of skipping their process for workpieces arisen from the rework station by the events sk_drill and sk_weld. A different model was proposed for the test to warn the system how many times a workpiece has been rejected. So the test station performs, logically, two different tests. Test1 is applied to workpieces coming from the input buffer and has as response an approval, fi_ok, or the first rejection, fi_nok1. Test2 is for those workpieces arriving from the rework station and its result indicates an ap-

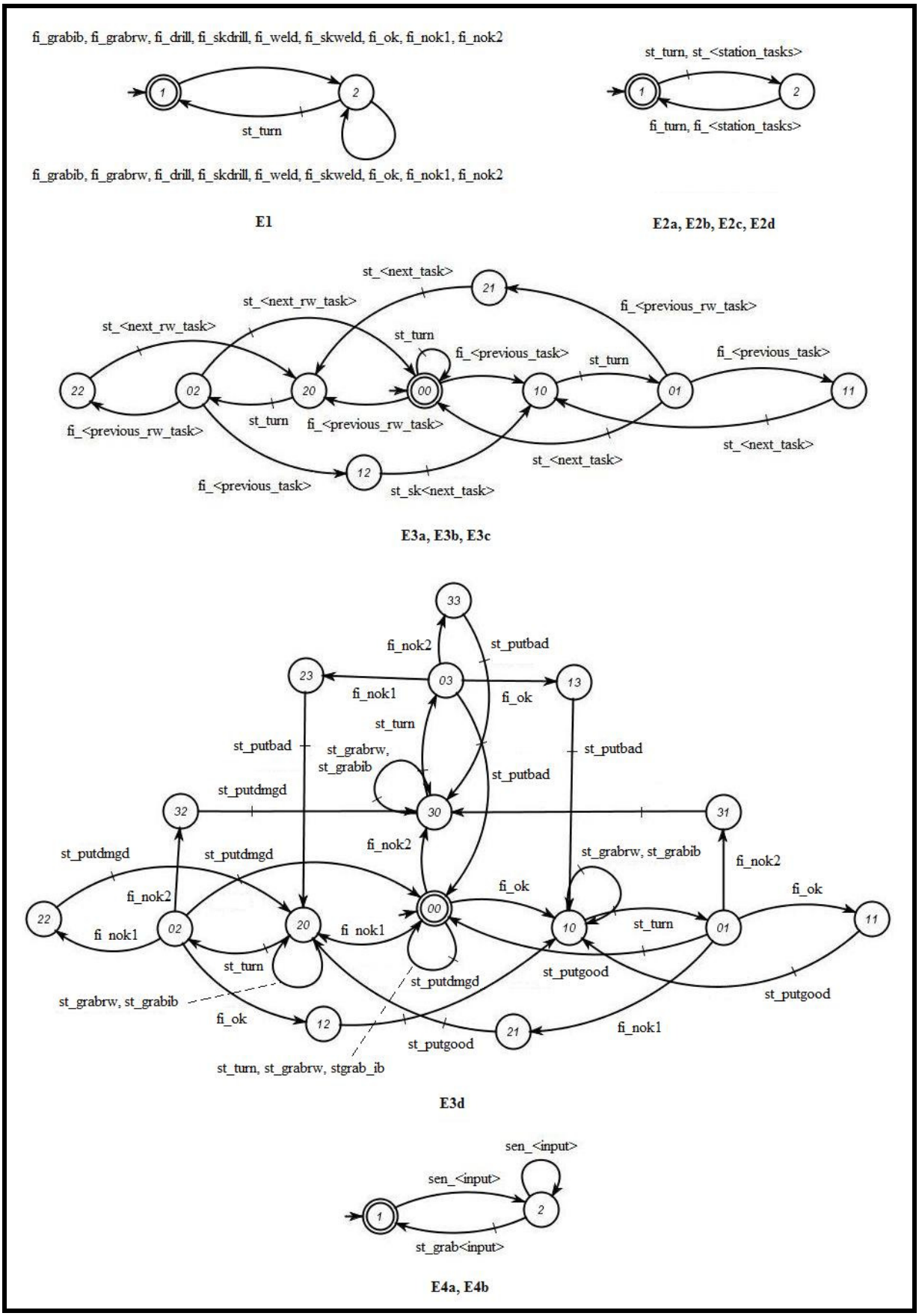

Figure 3. Behavioral specifications for the FMC

approval or the second rejection, fi_nok2. The grabber executes five different actions. It can grab workpieces from the input buffer, st_grabib, or the rework station, st_grabrw and place the workpieces into the three different storages – st_putgood, st_putbad and st_putdmgd.

Since we've got the FMC subsystem abstract models we need to define the FMC behavior, which is expressed, in Figure 3, through a set of generators as well. So for each one of the problems that might happen to the FMC, described in Section 2, we model a specification with IDES. The specifications affect the plants – a composite of subsystems in open loop – by excluding those undesirable sequences of actions.

Our first specification, E1, concerns about not allowing the table to turn if there's no workpiece on its slots. We do this by enabling the st_turn command only after the completion of actions that guarantee us at least one workpiece remains on the table – after the grabber puts a workpiece on the table from any of the inputs, after performing or skipping the drilling or welding and after testing. Notice that E1 contains a self-loop in state 2. This means the enabling of the previously mentioned actions more than once and so the possibility of executing these tasks concurrently.

To deal with the mutual exclusion that must exist between the table and the stations, we modeled four generators, E2a, E2b, E2c and E2d. By them we state that once the table is rotating, there must be no task execution by the respective station until a fi_turn is signalized and vice versa.

Specifications E3a, E3b, E3c and E3d take care of the synchronization between the stations. We separate this problem in four generators and each one focus on solving the coordination of two sequential stations. The generators states are referred by two numbers that indicates the presence of a workpiece in that slot, when different of 0, and a quality for the workpiece, numbers 1, 2 and 3. In specifications E3a, E3b and E3c, number 1 symbolizes workpieces coming from the input buffer and number 2 designates refurbished ones. The test can result in three responses and specification E3d states numbers 1, 2 and 3 represent an approval, a first rejection and a second rejection respectively.

The previous_tasks in specifications E3a, E3b and E3c are those the second station is waiting to be performed by the first station in the ordinary manufacturing process and the next_tasks those the second station should execute. The same applies to the previous_rw_tasks and next_rw_task except they regard to the refurbished workpieces manufacturing cycle. For instance, previous_task in E3a is fi_grabib, next_task is st_drill, previous_rw_task is fi_grabrw and next_rw_task is st_drill. Specification E3b coordinates the drilling and welding stations and E3c the welding and test stations.

Specification E3d, which is responsible for the interaction of the test and grabber stations, follows the same design pattern presented for E3a, E3b and E3c, however it differs in size. This happens because the test station offers three possibilities of response – fi_ok, fi_nok1 and fi_nok2 – whilst the previously mentioned models two – raw and refurbished workpieces manufacturing cycle.

These four specifications assure there will be no overflow of workpieces in slot S1; the table rotation before a workpiece is drilled, welded or tested; the welding and drilling of refurbished workpieces; the drilling, welding, testing and grabbing if there's no workpiece at the respective station; the double drilling, welding or testing of the same workpiece. In addition they signalize the grabber where to put the forthcoming workpieces.

We can't predict when workpieces will arrive from both inputs, however we establish by our last specifications, E4a and E4b, the grabber must take them only after the inputs sensors have been activated. The self-loop in state 2 enable the sensors activation at any instant of the manufacturing process.

In local modular approach, a specification restricts the actions of a local plant which is composed by all the subsystems that share at least one event with this specification. In specification E3d, for example, the local plant is the composition of the table, test and grabber subsystems since they share at least one event with E3d. The FMC local plants are described as follows and the operator || symbolizes the generators synchronous composition. $G_{loc1} = G_3 \parallel G_4 \parallel G_5 \parallel G_6 \parallel G_7$, $G_{loc2a} = G_3 \parallel G_4$, $G_{loc2b} = G_3 \parallel G_5$, $G_{loc2c} = G_3 \parallel G_6$, $G_{loc2d} = G_3 \parallel G_7$, $G_{loc3a} = G_3 \parallel G_4 \parallel G_7$, $G_{loc3b} = G_3 \parallel G_4 \parallel G_5$, $G_{loc3c} = G_3 \parallel G_5 \parallel G_6$, $G_{loc3d} = G_3 \parallel G_6 \parallel G_7$, $G_{loc4a} = G_1 \parallel G_7$, $G_{loc1} = G_2 \parallel G_7$.

Each local plant behaves according to what is imposed by its respective specification. This language, which represents a local specification, is determined by the synchronous composition of the local plant and the specification that produced the former. So in this work we have, for $x \in \{1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, 4a, 4b\}$, the local specifications $E_{locx} = G_{locx} \parallel E_x$, with the number of states shown in Tab. 1.

3.2. Synthesis of local modular supervisors

Once the modular specifications and respective local plants have been modeled, the set of optimal supervisors is computationally achieved with the TCT tool. In the synthesis, we want to find a set of supervisors to control every one of the local plants G_{locx} – closed loop behavior. These supervisors are represented by generators S_x which disable, according to the control actions defined for their states, certain events in the local plants. The completion of tasks of G_{locx} under supervision of S_x happens when both generators reach a marked state.

Generically, a supervisor S is nonblocking if $\overline{L_m(S/G)} = \overline{L}(S/G)$, where $\overline{L}$ represents the prefix closure of a language L. This means that every state in the supervisor that can be reached from the initial state permits a chain of events that leads to marker states. The necessary and sufficient condition for the existence of a nonblocking supervisor S that

satisfies a given specification $K \subseteq L_m(G)$ ($L_m(S/G) = K$) is the controllability of K. K is said to be controllable if $\bar{K}$ $\Sigma_u \cap L(G) \subseteq \bar{K}$. The class of controllable languages contained in K has a supremal element $SupC$(K, G).

So, in the local modular approach we compute, for x=1,…,m, the nonblocking supervisors S_{locx} directly on their respective local specifications E_{locx} so that $L_m(S_{locx}/G_{locx}) = SupC(E_{locx}, G_{locx})$. The local modularity – a sufficient and necessary non-conflict test – is verified by checking if the synchronous composition of all local supervisors is reachable and coreachable.

We obtained for the FMC 11 local supervisors S_{locx}, $x \in \{1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, 4a, 4b\}$ through the $SupC(E_{locx}, G_{locx})$ procedure. The supervisors were verified as non-conflicting after checking the generator $S = S_{loc1} \parallel S_{loc2a} \parallel S_{loc2b} \parallel S_{loc2c} \parallel S_{loc2d} \parallel S_{loc3a} \parallel S_{loc3b} \parallel S_{loc3c} \parallel S_{loc3d} \parallel S_{loc4a} \parallel S_{loc4b}$ (19180 states) is trim (reachable and coreachable). This indicates the set of supervisors has the same control attitude a monolithic supervisor would. The synthesis computations took less than one second and were processed in a personal computer (AMD Turion™ 64 X2 Dual-Core 1.9 GHz, 2048 MB RAM, Windows Vista® Home Premium 32-bit version).

3.3. Supervisors reduction

Some of the computed local supervisors are described through a wide map of control sequences which involved a large number of states. Besides the lack of understandability, to implement such complex supervisors is not possible due to the huge amount of memory they require from the system. The simulation tool described in next section and used in this work is only functional for size-limited supervisors as well. As a solution for this problem we reduced (Su and Wonham, 2004) the supervisors using TCT and as result we obtained smaller supervisors RS_{locx} which represented memory economy and a more elucidated control map. The computation time for the reduction ranged between less than 1s and 2s, with exception of the worst case that took around 13 hours to be processed. All of them were made with the previously described computer. The reduced supervisors are similar to the original specification models and then comprehensible. The numbers of states for each generator involved in the controller synthesis are seen in Tab. 1.

Table 1. Number of states of generators in supervisors synthesis

x	E_x	G_{locx}	E_{locx}	S_{locx}	RS_{locx}
1	2	324	648	648	2
2a, 2b, 2c	2	6	4	4	2
2d	2	12	7	7	2
3a	9	36	324	252	9
3b, 3c	9	18	162	90	9
3d	16	36	576	288	16
4a, 4b	2	6	12	12	2

4. SIMULATION AND IMPLEMENTATION

4.1. Simulation

During this work we used a simulation tool before applying the generated supervisors to the physical system. The main purpose was to avoid the confusion caused by the late change of design patterns. Thus the emulator helped us to visualize the proposed logic and to make the necessary changes to the project beforehand.

DESEM is a discrete event system emulator developed by Universidade Federal de Santa Catarina that makes possible the simulation of the generated supervisors on the plant. With this tool we can follow the actions taken by the supervisor progressively and so we are able to identify whether the model is working as proposed by the control designer. We could check, for example, if an event that should be enabled at certain circumstance is not being forbidden from happening.

The simulation in DESEM is made by adding blocks which represent each part of the physical system. The blocks are loaded with their respective automata. Naturally, if no supervisor is keeping track of the plant the system can perform any event randomly. By adding supervisors to DESEM, the plant respects the constraints imposed by them.
The simulation works as a feedback for the supervisory control synthesis. When any odd behavior is detected, the designer has to think of a different specification model to satisfy the system logic. The modeling and simulation stages are overcome only when the response of the simulation works as wanted. Thus the modeling demands the greatest effort from the designer since all the supervisors problems are solved at this phase.

The use of simulation anticipates the test of the proposed supervisors and excludes the need of the cell subroutines in advance. So, a possible delay on the subroutines development doesn't interfere in the supervisors design process because the feedback provided by DESEM is enough to correct the mistakes. The independence between the development of the supervisors and subroutines suggests a time economy to reach the final controller if they both are made by different group of designers.

In this work, we had to review some specification models related to the refurbished workpieces manufacturing cycle that made the FMC simulation in DESEM diverge from the expected one.

4.2. Hierarchical structure

The SCT supervisors when applied on a plant represented by a set of generators act disabling events that are physically possible to be executed by the plant but cannot happen at that specific state. The theory implies that events are happening spontaneously and the supervisors block controllable events temporarily until the plant reaches a different state where that event can occur.

However, the operational sequences of our physical system aren't triggered automatically and the events of a plant, controllable or not, must be associated with their proper operational sequence inasmuch as they don't represent a direct input and output of the PLC. To implement the synthesized supervisors to the PLC and make them work as suggested by the SCT we applied a three level hierarchy so that there's an interface (Queiroz and Cury, 2002) between the modular supervisors and the real system.

As seen in Fig. 3, the modular supervisors are at the top of the hierarchy. The subsequent level contains the product-system that represents the abstract model of the local plants. Finally at the bottom level there are the operational sequences that act on the physical system. The supervisor receives from the local plant the indication of events occurrence and updates the list of controllable events that must be disabled. The product-system is informed by the supervisor of the events that are forbidden to happen at that state and sends commands to execute subroutines that are not disabled. Every time the plant sends a command to the operational sequences or gets a response from them the supervisors must be updated. The operational sequences activate the output signals and read the input signals from the real system, besides sending responses to the product-system.

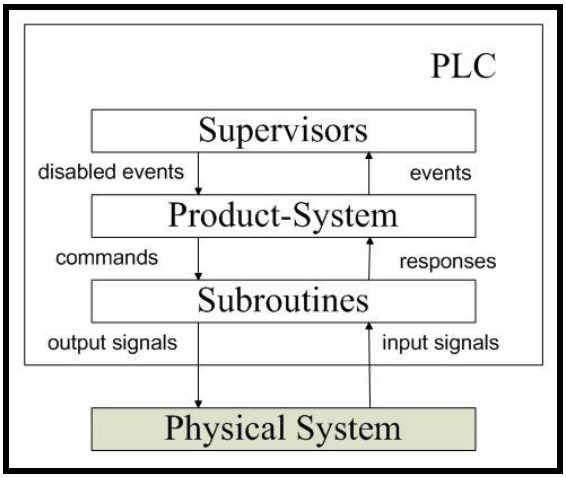

Figure 4. Hierarchical structure of implementation

4.3. Automatic code generation

Once the supervisor simulation has been approved, we used an automatic code generation tool, called Ides2St, to obtain a structured text (ST) code. This tool translates files designed through IDES. If the supervisory control was synthesized using a software other than IDES, such as Grail (Reiser, Cunha and Cury, 2006) or TCT, the files can be imported to IDES and then converted into ST. The Ides2St translation is founded on the implementation hierarchy previously described and in order to generate the code Ides2St requires as input the local plants and supervisors automata.

Ides2St assigns a memory variable of the PLC to every state of the finite state machines (FSM) representing the generators and the events. During the initialization of the program all the supervisors and local plants are set to their initial states as in Fig. 5(a). This should not be confused with the physical system startup which is adjusted before entering the logic control procedure. It uses the auxiliary variable ilc_inited to guarantee the logical startup runs once, for the reason that a PLC program is scanned sequentially.

According to Ides2St, when an event takes place its variable is set to 1 and remains like this until the supervisor is updated. To avoid the undesirable situation of computing two subsequent transitions without updating the supervisors, Ides2St uses an auxiliary variable called evt_blk. It doesn't mean that those subsequent events will be ignored by the program, but postponed. Their variables are set to 1 but they just run in a convenient state at the supervisors.

The program code related to the supervisor level is divided in two parts. The first one, Fig. 5(b), specifies the supervisors as FSMs. It describes them according to the current state and the event transition that leads to the next state.

Each supervisor is composed by a group of IFs instructions that set the bit of the current state to 0 and the target to 1 when a transition happens. The second part, Fig. 5(c), refers to the disenabling of events which is done by setting on a variable named De_<event>. The De_<event> is set to 1 when the variables of the supervisors' states which disable that event are on as well. In other words, the supervisor prevents an event from happening at a certain state if De_<event> is on.

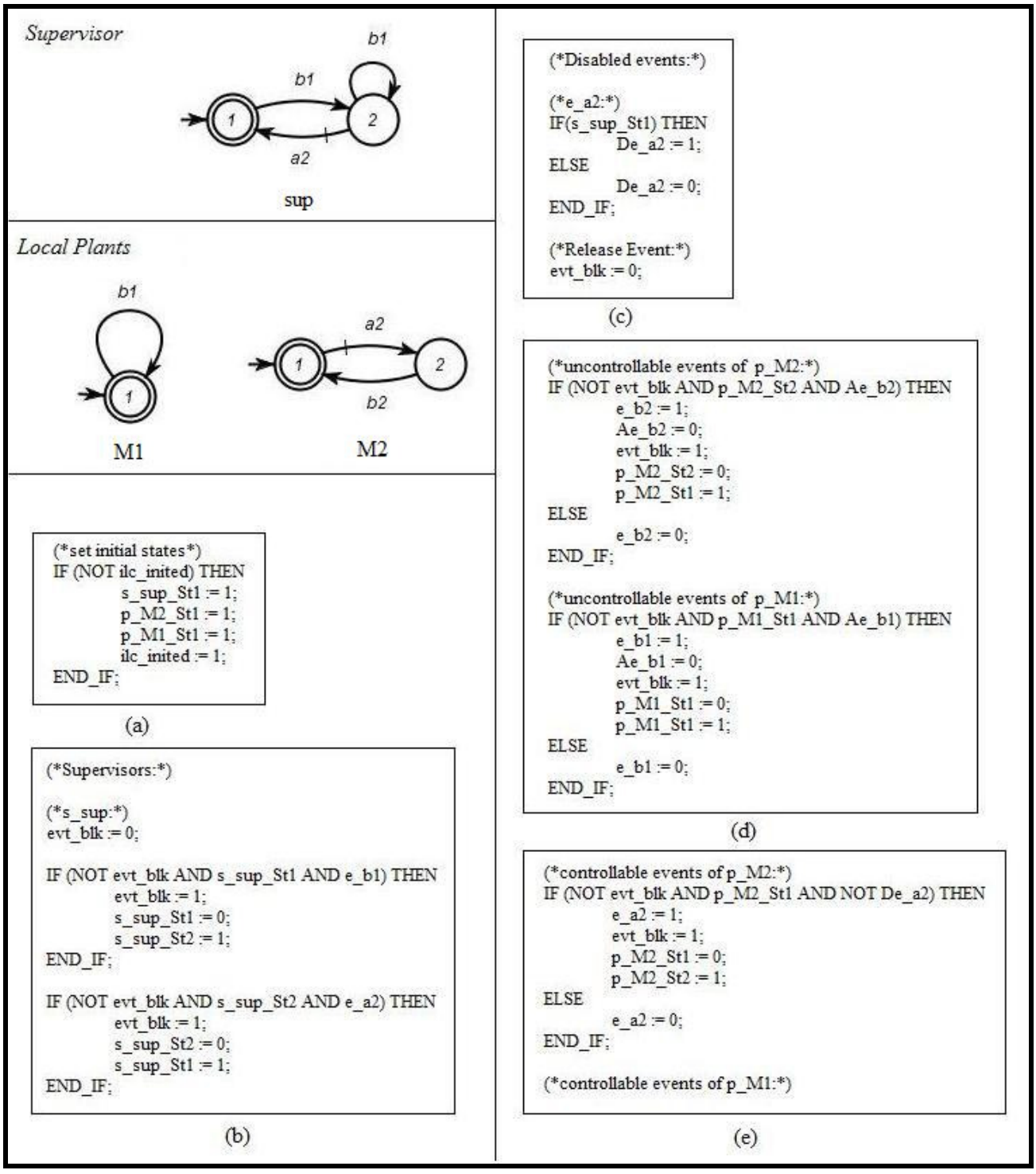

Figure 5. Example of Ides2St implementation code structure

The product-system is also implemented as FSMs, similar to supervisor, and its code is divided in two parts. The uncontrollable events code, Fig. 5(d), is implemented apart from the controllable one, Fig. 5(e), and the former must precede the latter in consequence of its unpredictability. Thus before any controllable event can be triggered the code will check if there is any uncontrollable event's bit set. If so, the local plant FSM is updated and sets the response event

variable e_<event> to update the supervisor. Were it not, and the local plant is in a state where a controllable event is not disabled, the variable e_<event> bit is set which means a command is sent to start a subroutine and to update the supervisor after this local plant does.

At the operational sequences level, Ides2St deals with the commands and responses forwarding and lets their memory address blank to be associated with the designer's subroutine start and ending. When a subroutine ends it sets the bit of the variable Ae_<event>. The subroutines that command the internal behavior of each subsystem were manually developed in Ladder and their explanation deviates from the scope of this work.

5. CONCLUSIONS

Certainly the implementation of the presented FMC problem by traditional methods of programming would cause confusion to the developer due to the wide range of tasks assigned to the test bed and this problem would normally be simplified by a restrictive solution, such as the manufacturing of one workpiece per cycle. The modularity test of the local supervisors showed us how complex the FMC problem is. If we tried to implement the monolithic solution, whose supervisor has 19180 states, we wouldn't succeed in reducing it since the computational cost demanded by the minimization algorithm is too high for such amount of states.

Applying the local modular approach in this work we focused on resolving local problems of the FMC described by each specification. Thus we obtained a group of smaller supervisors, that are equivalent to the monolithic supervisor, and we could minimize and implement them.

The simulation and the remodeling, until we could reach the desirable supervisors set, assured us that any malfunctioning in the FMC would not be related to the supervisors' model. In addition, since we used an automatic code generator, besides saving time, we avoided mistyping errors that might happen during the code's manual implementation. Therefore the supervisors code responsibility is excluded for any odd behavior performed by the FMC and it's passed on to the subroutines, mechanical and electrical systems.

At last, the approach taken in this work allowed the controller to be modeled concurrently and independently to the subroutines, seen that the simulation tool discarded the need of the subroutines to check the wanted behavior.

6. ACKNOWLEDGEMENTS

Financial support from CAPES is gratefully acknowledged by the first author. Also, the authors thank the undergraduate student Paulo Luis Franchini Casaretto for the help with the PLC code implementation.

7. REFERENCES

Feng, L. and Wonham, W.M.,2006, "TCT: A computation tool for supervisory control synthesis", Proceedings of the 8th International Workshop on Discrete Event Systems, Ann Harbor, USA, pp. 388-389.

Klinge, S.,2007, "Supervisory control of a manufacturing cell: modeling and implementation", Minor Thesis, Fakultät für Elektrotechnik und Informationstechnik, Otto-von-Guericke-Universität Magdeburg, 92 p.

Queiroz, M.H., 2004, "Controle supervisório modular e multitarefa de sistemas compostos", Doctoral Thesis, Dept. de Engenharia Elétrica, Universidade Federal de Santa Catarina, 150 p.

Queiroz, M.H. and Cury, J.E.R.,2002, "Synthesis and implementation of local modular supervisory control for a manufacturing cell", Proceedings of the 6th International Workshop on Discrete Event Systems, Zaragoza, Spain, pp. 377-382.

Queiroz, M.H. and Cury, J.E.R.,2000, "Modular supervisory control of large scale discrete event systems", Proceedings of the 5th International Workshop on Discrete Event Systems, Ghent, Belgium, pp. 103-110.

Ramadge, P.J.G. and Wonham, W.M.,1989, "The control of discrete event systems", Proceedings of the IEEE, Vol. 77, No. 1, pp. 81-98.

Reiser, C., da Cunha, A.E.C. and Cury, J.E.R.,2006, "The environment grail for supervisory control of discrete event systems", Proceedings of the 8th International Workshop on Discrete Event Systems, Ann Harbor, USA, pp. 390-391.

Rudie, K.,2006, " The integrated discrete-event systems tool", Proceedings of the 8th International Workshop on Discrete Event Systems, Ann Arbor, USA, pp. 394-395.

Su, R. and Wonham, W.M.,2004, "Supervisor reduction for discrete-event systems", Discrete Event Dynamic Systems, Vol.14, No. 1, pp. 31-53.

Vaz, AF and Wonham, WM,1986, "On supervisor reduction in discrete-event systems", International Journal of Control, Vol.44, No.2, pp. 475-491.

Wonham, W.M.,2008, "Supervisory control of discrete-event systems", Dept. of Electrical and Computer Engineering, University of Toronto, 458 p.