Local Modular Supervisory Implementation in Microcontroller

Yuri Kaszubowski Lopes, André Leal, Roberto Rosso Jr., Eduardo Harbs

9^{th} *International Conference of Modeling, Optimization and Simulation - MOSIM'12*
June 6-8, 2012 - Bordeaux - France
"Performance, interoperability and safety for sustainable development"

Local Modular Supervisory Implementation in Microcontroller

Yuri Kaszubowski LOPES, André B. LEAL, Roberto S. U. ROSSO Jr., Eduardo HARBS

Universidade do Estado de Santa Catarina
Rua Paulo Malschitzki, s/n
89219-710 Joinville,SC - Brazil
yurikazuba@gmail.com, leal@joinville.udesc.br, robertorossojr@ieee.org, eduardo.harbs@gmail.com

ABSTRACT: *This paper presents the implementation of supervisory control theory in microcontrollers. To ensure a good implementation problems previously discussed in the literature must be solved in the microcontroller environment. The memory of these devices is limited so the use of local modular supervisors is encouraged as well as the use of strategies for compact representation of these models in memory. Finally a code generation tool which implements the solutions presented in this work is elaborated aiming to facilitate the use of this research. The major advantage of microcontroller is that it represents a low cost solution for manufacturing automation when compared to a PLC. The use of the supervisory control theory provides formal methodology for automatic synthesis of the controller, obtaining an optimal control process, which is a minimally restrictive and non-blocking control.*

KEYWORDS: *Supervisory control, Discrete event systems, Automata, Microcontrollers.*

1 INTRODUCTION

Discrete Event Systems (DES) are a modelling abstraction for a large variety of systems (Fabian and Hellgren, 1998). The Supervisory Control Theory (SCT), presented by Ramadge and Wonham (1987) and Ramadge and Wonham (1989), is an approach which provides formal methodology for automatic synthesis of the DES controller. The main characteristic of the SCT is the generation of controllers performing a control action to achieve real goals.

According to Brandin (1996), there is lack of real industrial applications in the use of SCT. Queiroz (2004) reports that this lack is partly due to the problems caused by the huge number of states in the models in real systems, particularly in the monolithic approach, and partly attributable to the lack of practical results which demonstrate its implementation. Several papers bring different approaches considering the problem of computational complexity due to the exponential growth in the number of states in the composition of automata (Eyzell and Cury, 2001), (Zhong and Wonham, 1990), (Wonham and Ramadge, 1988), (Queiroz and Cury, 2000). Those papers show other methods besides the monolithic approach: the modular approach and the local modular approach. Other articles successfully apply the SCT (Silva et al., 2011; Diogo et al., 2011; Yalcin et al., 2005).

Beyond the exponential growth in the number of states in the composition of automata, the efficient action of these automata in memory is another important issue in the implementation of supervisory control in microcontrollers. In Barreta and Torrico (2008) the use of chained lists and an automatic code generation tool was presented. In Lopes et al. (2011) a strategy of a compact way to store the automata data in a vector for monolithic supervisors called **memory safe** was presented, and it started the development of an automatic code generation tool, called *Nadzoru*. Comparing these two approaches the **memory safe** represents a lower memory cost than the use of chained lists. However, chained lists are more efficient in terms of computational time. Here we apply the **memory safe** to implement the local modular control structure in a microcontroller aiming the lower memory space cost.

To implement the supervisory control some problems are detected, such as causality, choice and so on (Fabian and Hellgren, 1998). In this paper we present solutions to the problems in implementation of microcontrollers introduced by Fabian and Hellgren (1998). In Leal et al. (2009), Cruz et al. (2009), Cruz (2011) and Leal et al. (2012) solutions to these problems for PLC implementation were proposed. Here we apply and extend these solutions to the microcontroller environment.

Finally we extend the *Nadzoru* tool to support a code generator for all these methods. In fact, until this work *Nadzoru* only created the data structure based

on the monolithic supervisor and ran a single state machine in this data structure. The new tool version is capable to make monolithic and local modular supervisor microcontroller code, with a different set of solutions to each problem presented.

The SCT is addressed in Section 2, next the problems of PLC implementation addressed by Fabian and Hellgren (1998) are exposed in Section 3. The problems in the microcontroller environment is presented in Section 4. The microcontroller implementation and proposed solutions to the problems addressed are exposed in Section 5. Finally, Section 6 presents the conclusions and future works.

2 SUPERVISORY CONTROL THEORY

In a DES, physical events may be in an unknown interval, the processes are discrete, asynchronous and possibly non-deterministic. The DES can control, coordinate and assure an orderly flow of events (Ramadge and Wonham, 1987; Ramadge and Wonham, 1989). In the SCT a DES system is modeled in two different kinds: the plant subsystem models and the control specifications models. The plant subsystem models are all physical possibilities and the control specifications models are all the control rules.

In order to illustrate the SCT methodology we give an example. Consider a system consisting of a conveyor that brings parts of three types: c_1, c_2 and c_3 and a sensor which identifies these parts. The problem is that the conveyor can move only while the sensor is not active, when the sensor is activated the conveyor should stop. If the sensor changes its state from enable to disabled, the conveyor restarts the motion. Figure 1 presents the automata that can be used to model (a) the conveyor, (b) the sensor and (c) the control specification which implements the supervisory control rules which guide the plant behavior. The specification modeled by the automaton (c) prohibits the motion of the conveyor (event $e1\_lig$) while the sensor is enabled and prohibits to turn off (event $e1\_des$) the conveyor while the sensor is disabled.

The events $e1\_lig$ and $e1\_des$ are controllable events, which are the ones that can be enabled or disabled by external agents, in this case by the controller. The transition from state 1 to 2 happens when the sensor is enabled (events: s_c1, s_c2 or s_c3) and the transition from state 2 to 1 occurs when the sensor is disabled (event s_des). The events s_c1, s_c2, s_c3 and s_des are uncontrollable events, which are the ones that cannot be prevented from occurring (Queiroz and Cury, 2002a).

In the following we present the monolithic, the modular and the local modular approaches.

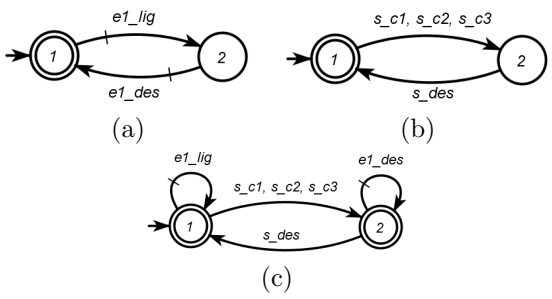

Figure 1: Automata for: (a)conveyor, (b)sensor (c)control specification.

2.1 Monolithic Approach

After make the plants subsystems and the control specifications models the first strategy is to create a unique monolithic supervisor. A supervisor, like plant subsystems models and control specifications models, is an automaton (see Sipser (2005) for an automaton definition). A monolithic supervisor $S^{monolithic}$ is represented by an automaton that generates the supremal controllable language given by $L_m(S^{monolithic}, G) = SupC(G, K)$. Where the free behavior of the physical system G is $G = G_1||...||G_n$ for n subsystems models. The target language $K = G||E$ which corresponds to the desired system behaviour of the system, and the control specification $E = E_1||...||E_m$ for m specification models. For details about the supremal controllable language calculation ($SupC$) and synchronous composition ($||$) operations see Ramadge and Wonham (1987).

2.2 Modular Approach

In Wonham and Ramadge (1988) the modular approach for the supervisory control which explores the modular property of each control specification was presented. This approach creates a supervisor for each control specification instead of a unique supervisor for all of them. Thus, in a problem with m control specifications models, m supervisors $S_i^{modular}$ such that, $L_m(S_i^{modular}, G) = SupC(G, K_i)$ for $i = 1, ..., m$, are created. Where the free behaviour of the physical system is the same of the monolithic supervisor, given by $G = G_1||...||G_n$ for n plants models and each target language is given for each control specification as $K_i = G||E_i$.

The benefit of this method is related to the combinatorial explosion of the automata synchronous composition. In the worst case a synchronous composition operation of p automata $A_1, ..., A_p$ results in an automaton $A_{1||...||p}$ where the number of states $\#A_{1||...||p}$ is given by $\#A_{1||..||p} = \#A_1 \times ... \times \#A_p$, in other words the resulting states in a synchronous composition is the multiplication of the number of

states of all composed automata. For the worst case, if all modular supervisors have two or more states, the following relationship is valid $\#S^{monolithic} \geq (\#S_1^{modular} + ... + \#S_i^{modular})$.

To use the modular approach the supervisors cannot be conflicting. In this case, the control action of the modular supervisors results in the same control action as in a monolithic supervisor. In order to verify if two or more supervisors are conflicting do $S^{sync(modular)} = S_1||...||S_p$ and check if $S^{sync(modular)}$ is accessible and co-accessible. Non accessible or non co-accessible supervisors synchronization denotes a conflict. If two or more supervisors are conflicting they need to be calculated together. Therefore, the resulting supervisor $S_{1,2}$ for two conflicting supervisors S_1 and S_2 is $S_{1,2} = SupC(G, K_{1,2})$, where $K_{1,2} = E_1||E_2||G$.

2.3 Local Modular Approach

The local modular approach (Queiroz and Cury, 2000), expands the use of the modular property of each control specification to the modular structure of the plant. In this approach, similar to the modular one, one supervisor is created for each control specification. However, here only the subsystems which are affected by the control specification are composed in the calculus of each modular supervisor, i.e. each supervisor S_i^{loc} controls an specific local plant G_i^{loc} composed by the subsystems which have at least one event in common with E_i. Figure 2 shows an example $G_1^{loc} = G_1||G_2$, $G_2^{loc} = G_2||G_3$ and $G_3^{loc} = G_4$.

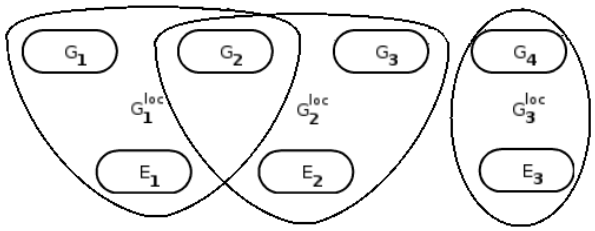

Figure 2: Local modular supervisory use of both modular property.

The advantage of the local modular approach if compared with the modular approach is the reduction in the computational effort of the syntheses process and the size of supervisors. This is possible because not all subsystems models are used in the calculus of each supervisor. This results in small supervisors which enable a lower memory usage implementation, maintenance and readability of source code.

3 PLC IMPLEMENTATION PROBLEMS

In Fabian and Hellgren (1998) the follow main problems-classes based on PLC implementation of supervisory control are exposed:

Signals and events: the straightforward way to implement a state machine in a PLC is to associate events with rising and falling edges of PLC signals and represent each state and each event with internal Boolean variables. Thus, the transitions can be represented by a Boolean AND between the state variable and the event variable, as illustrated in Figure 3.

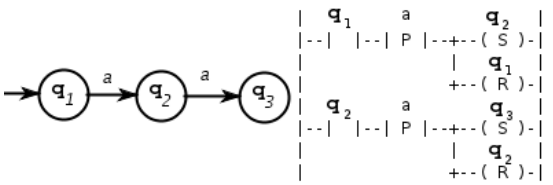

Figure 3: Avalanche effect in a PLC implementation.

Intuitively the uncontrollable (controllable) events may be associated with signal changes in the PLC inputs (outputs), which are updated at the beginning (end) of each PLC scan cycle. However, care must be taken when making these associations in order to not introduce the *avalanche effect* in the implementation. As illustrated in Figure 3, this effect makes the software jump over an arbitrary number of states within the same PLC scan cycle and may occur particularly if a specific event is used to trigger many successive state transitions.

The second problem in this class is the *simultaneity* problem. Due to the cyclic execution of PLC, the input readings are performed periodically and if two or more signals changes in the PLC input occur between two scan cycles it is not possible to identify the order of their occurrence. Figure 4 exemplifies this problem, the real sequence is *ababa*, but because of the interval between two scan cycles the code cannot detect which event happened first, *a* or *b*.

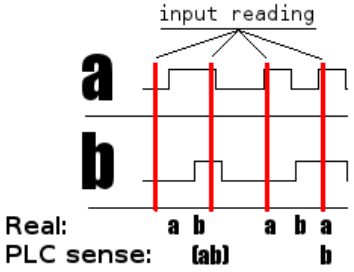

Figure 4: Simultaneity problem.

In addition, if an event changes its state twice between two scan cycles, the event cannot be identified. Thus, there are two possible sense of the PLC: *abb* or *bab*.

Causality: the SCT assumes that the plant spontaneously generates all events, and the supervisor only disables events which should not occur in the plant. However, the plant changes values as a response to

signal-changes commanded by the controller (PLC). Then, for implementation purposes, it is necessary to adapt the practice to the theory. This means, if the plant only generates uncontrollable events, which can not be disabled, and the controller only disables controllable events, something needs to generate the controllable events.

Choice: The supervisor obtained through the SCT is required to be minimally restrictive, so the plant has the greatest possible freedom behavior satisfying the specifications. This usually results in a supervisor that in some states has more than one controllable event enabled. Thus, as a single event must be chosen, we have the problem of the choice. Also, according to Malik (2002), the choice can lead to a blocking implementation of a non-blocking supervisor.

Figure 5 exemplifies this problem, considering that the current state is 1. If a simple implementation is made, such as a choice which selects always the same event in each state, in this example, if event a is always chosen in states 1 and 3, we can achieve the final state, but it does not execute event b, which can mean it will never make a specific kind of product and this may be not desirable. To sort this problem out, a choice rule can be alternate choices, such as the sequence *abab*.... However, with this sequence the machine state will be in a dead lock in states 1 and 3 and will never achieve the final state.

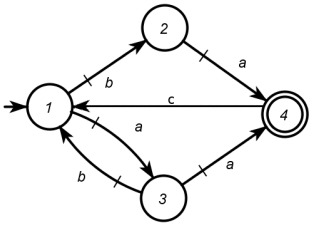

Figure 5: Choice's problem.

Inexact synchronization: During the PLC program execution the PLC does not observe the state of the plant. Thus, the signal-changes occurred in the PLC inputs during the program execution are associated with events just in the next PLC scan cycle. The inexact *synchronization problem* happens whenever the changed signal invalidates the choice made by the controller.

In order to illustrate this problem, consider the automata shown in Figure 6, where the event a is uncontrollable, event b is controllable and the current state is the initial state. In Figure 6(a) the strings ab and ba lead to the same state and thus, there is no difference if event b occurs after or before the detection of event a. In this case, the language generated by this automaton is said to be *delay insensitivity* (Fabian and Hellgren, 1998). However, in other cases this order is very important and the automaton does not have this property. Consider the program shown in Figure 6(c), which corresponds to the implementation of the automaton shown in Figure 6(b). If a rising edge in the PLC input signal associated with event a is detected, rung 1 performs the transition to q_3. Otherwise, if the event a is not recognized at the PLC reading stage, b is generated in rung 2 with the transition to q_2. However, while the PLC executes the program (before the PLC has applied the output corresponding to event b), the plant may generate a signal-change associated with event a, which invalidates the generation of event b.

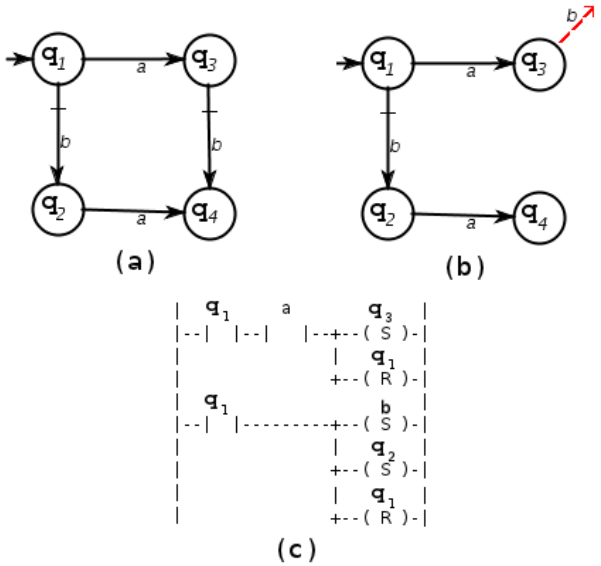

Figure 6: Inexact synchronization problem.

4 MICROCONTROLLER IMPLEMENTATION PROBLEMS

Signals and events: the *avalanche effect* is non existent in the microcontroller implementation because of the separation of the data structure that represents the automaton and the program logic implementation. But in a PLC implementation this problem occurs and the solutions can be found in Cruz (2011) and Leal et al. (2012). The *simultaneity* occurs in microcontroller environment like in the PLC one. Thus, the microcontroller implementation also must take care of the order in which uncontrollable events occur. However, due to inexistence of a scan cycle in microcontrollers, the solutions for this platform are broader than that for PLC and the freedom of development of a microcontroller allows us to overcome the simultaneity problem.

Causality: this problem must be solved in microcontroller environment as in PLC environment, the problem is strictly related to the SCT. In both environments there is a need to overcome the fact that the SCT expects which the plant generate spontaneously

all events. This in fact is not true in the implementation view. The implementation needs something which generates all events and the plant only generates the uncontrollable one. Thus, this problem persists in the microcontroller environment.

Choice: like causality, choice is a problem related to the SCT. This problem exists because the SCT is required to be minimally restrictive, so the plant has the greatest possible freedom, which means, the controller has choices. In PLC some solutions suggest to make a random choice. The problem is how to guarantee a good random function. And this reflects in a good random method implementation in both environments.

Inexact synchronization: Due to the possibility of occurrence of an event in the plant that invalidate the calculated choice of the controller the *inexact synchronization* is still a problem in microcontroller environment. However, in the PLC environment we are limited to the cyclic input, but in microcontroller the freedom of design given an option to reduce this problem, a solution will be explained in next section.

5 MICROCONTROLLER IMPLEMENTATION SOLUTIONS

In this work the use of local modular supervisor is justified by the fact that this approach reduces the combinatorial explosion of the automata synchronous composition. The **memory safe** approach is used because it represents a compact way to minimize the memory space used in the microcontroller. In Lopes et al. (2011) this approaches was applied only to the monolithic approach. Here the use of both, local modular supervisor and the **memory safe** approaches reduces one of the major problems in microcontroller use for supervisory control: the memory space.

The idea of the strategy named **memory safe** is to store the data concerning the state transitions of each supervisor in a vector organized as follows (see Figure 7). Each state of the automaton is represented in a sub-vector, which, in turn, describes all output transitions from each state. The first byte for each sub-vector determines the amount of output transitions θ from this state. Then we have more $\theta \times 3$ bytes to represent each transition in this state (sub-vector): the first is the number used to denote the event, and the other two are the target state from the transition.

A priori, this method is limited to 256 events, $2^{16}-1$ states and 255 output transitions for each state in this basic form. However this method can be adapted to accept a large number of events, states and output transitions for each state. The memory occupation for each automaton is given by $C_{mem}^{memorysafe} =$

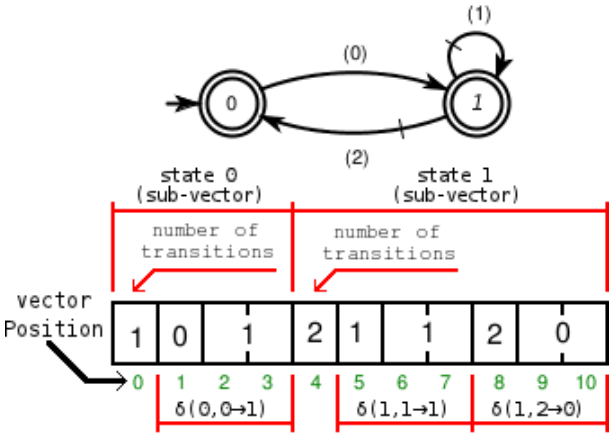

Figure 7: Memory safe strategy: a vector stores the data set of an automaton.

$T_{char} \times (s+3\times t+e)$, where s is the number of states in the automaton, t is the total number of transitions in the automaton and e is the number of different events in the automaton. The $s+3\times t$ component represents the vector used in Figure 7. The other component e represents an another vector with length e, for each automaton, which is used to keep the information if each event is controllable or not.

5.1 Implementing Solutions

In this work we look through the PLC implementation problems and review solutions for PLC environment. After that we analyse which of these problems persist in microcontroller environment. Now we adapt the PLC environment solutions to the microcontroller environment and extend these solutions with new options.

In order to solve the choice problem Leal et al. (2009) proposed the use of a random *bit* to select a controllable event. In Cruz (2011) this proposed was improved to use a random number. It is well-known that microprocessors are deterministic and cannot produce true random numbers, generating only pseudo-random numbers. To overcome this problem we proposed the use of external values. Therefore, random quality is guaranteed by an external value. Here we extend this solution by two types of random functions for microcontroller: (1) use an input value from an analogical-digital (AD) converter which reads a thermistor value as a random-number; (2) a pseudo-random function $f(x) = x * (-35) + 53$, where x is the last result for the $f(x)$ and the initial value from x, called seed, is created by the AD input. In the first case we had two problems, the low variety of the random number and the delay to get values. In the second case the use of a fixed seed is not satisfactory because at every restart the system makes the same sequence. Thus, we propose to combine these two

methods in order to use a random function f(x) in which x is a seed that dynamically receives a value from the AD input. This method is very quick and provides high variety of distict sequences.

Another solution proposed to the choice problem is the use of a a global queue for all controllable events. When the choice problem occurs, the queue is consulted and the next enable controllable event in the queue order is selected. After all queue element are used the queue is restarted. For an example in a queue $Q = \{a, b, d\}$, if the state S_1 has the first choice problem between a and b the selected option is going to be a, suppose the second choice problem in the state S_2 between a and b again but now the next element in the queue is b so b is selected. If the third selection is between a and b again, so the next queue element is d, however d is not an option, because d is not enabled, then we jump the d in queue, we restart the queue at this point because all queue elements are used, and we try the next which is a, so a is selected. If we use in the example of Figure 5 the queue $Q = \{a, b\}$ starts in state 1 the state sequence is going $1, 3, 1, 3, 1, ...$ and this will never achieve a final state, in other words this solution leads to a blocking control action.

To solve this blocking control problem we propose the use of a local queue, which works similarly to a global queue, but we have a queue for every state responsible to the choice problem. In a local queue the choice made in a state does not affect the other states and this guarantees that all possible transitions can be used, such as a deep search in a tree. In the previous example the sequence is going to be $1, 3, 4, 1, 2, 4, 1, 3, 1, ...$ The problem with local queue is the memory space required because all states are likely to be the choice problem and need to keep a queue.

We also propose a hybrid approach between the local or global queue and the random selection. In this hybrid method the queue order is randomly generated when the queue restarts. We call these two approaches *random local queue* and *random global queue*. In *random global queue* we solved the "lock problem" using less memory space than in the solution adopted for the *random local queue*. The *random local queue* guarantees a non-deterministic exploration of the alternative paths of the supervisor and at the same time guarantees that all possible transitions may be used, but it still requires a large memory space. Other works addressed the synthesis and optimization of nonblocking supervisors based on Petri nets (Hu et al., 2012; Hu et al., 2011; Hu and Li, 2010; Hu and Li, 2009).

To solve the simultaneity problem we aim to reduce the time between each input reading. To do that we offer two solutions: (1) use a timing interrupt to read all inputs and; (2) use an external interruption to activate the read input method. In both cases the read events are stored in an input buffer. In the timing interrupt the interval between two external reads is configured according to the needs raised by the designer. In the external interruption this feeling is not required and the read of external stimuli is made just in time.

For the inexact synchronization problem we verify if the input buffer from the previous solution is free just before the application of a selected controllable event. If the buffer is not free, which means that a uncontrollable event had happen, we refresh the current state of the state machine with the uncontrollable events and recalculate the controllable event to be executed. Otherwise if the input buffer is free we apply the selected controllable event. It drastically reduces the problem because diminishes the time between apply the controllable event and read the plant state.

To solve the causality problem we can adopt the general control system structure proposed by Queiroz and Cury (2002b). This control structure is based on a three level hierarchy (see Figure 8) that executes the modular supervisors' concurrent action and interfaces the theoretical model with the real system. In this approach, the controllable events that are not disabled by the modular supervisors are generated in the layer named "Product Systems", which consists of the implementation of the asynchronous plant models. Thus, using this approach allow us to keep the hypothesis that the plant generates all events, and that the supervisor dynamically disables events that the plant might otherwise have generated. The other advantage of implementing the product system is that the supervisor can be reduced, however, it is still necessary to keep the product system in parallel with the supervisor (Cruz, 2011).

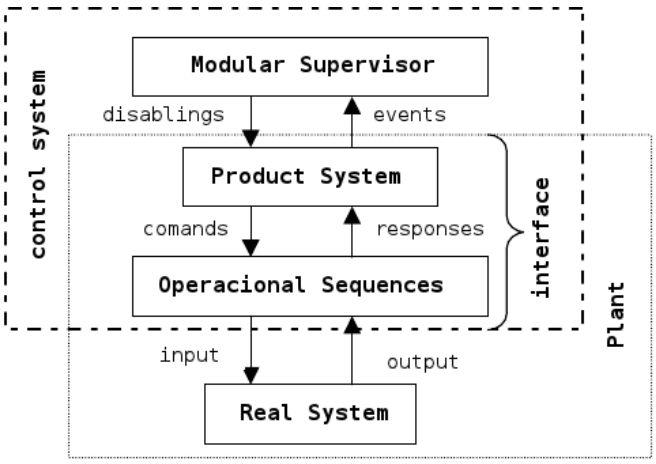

Figure 8: Control system basic structure (Queiroz and Cury, 2002b)

5.2 Execution Logic

The execution logic proposed for microcontrollers is to perform all the state transitions, and then dynamically update the disabling of controllable events, so no need to maintain a static list of events disabled in every state. Thus, we use only a list of events associated with each of the automata. At run time, the microcontrolller computes the disabled events at the current state.

5.3 Comparative Results

Table 1 presents the number of states and transitions of all modular supervisors for the problem presented by Lopes et al. (2011). And the Table 2 shows the number of states and transitions of all local modular supervisors for the same problem.

Table 1: Number of states and transitions of the modular supervisors

Supervisors	States	Transitions
S_1	72	444
S_{2_6}	448	2892
S_3	132	844
S_4	132	844
S_5	180	924
S_7	72	438
TOTAL	1036	6386

Table 2: Number of states and transitions of the local modular supervisors

Supervisors	States	Transitions
S_1	4	10
S_{2_6}	121	360
S_3	33	112
S_4	33	112
S_5	30	74
S_7	12	29
TOTAL	233	697

Table 3 represents the total number of states and transitions in each case for the problem presented by Lopes et al. (2011). In the modular and local modular approach the result is the sum of all supervisors states and transitions. This table also shows the number of bytes required to represent these supervisors in three approaches: (1) Matrix, where memory is $C_{memory}^{matrix} = s \times s + e$; (2) chained list, presented by Barreta and Torrico (2008), where memory is $C_{memory}^{chained\_list} = 2 \times s + 5 \times s + e$ and; (3) the memory safe, introduced by Lopes et al. (2011), where memory is $C_{memory}^{memory\_safe} = s + 3 \times s + e$.

Table 3: Comparative Results for a case study

Supervisory	States	Transitions	Matrix	Chained Lists	Memory Safe
Monolithic	3420	11808	11696423	65903	38867
Modular	1036	6386	1073319	34025	20217
Local Modular	233	697	54312	3974	2347

5.4 Solutions Test

In the first step we tested the supervisors using a hardware that emulates a manufacturing plant. This hardware consists of LEDs (Light-Emitting Diode) and buttons. The buttons emulate the uncontrollable events from the plant and the LEDs show the control commands sent to the plant.

In the second step we tested the supervisors using a hardware that is connected to a computer software which emulates a manufacturing plant (a kind of virtual plant). This virtual plant sends by serial port in different rates the response of the plant to the controller commands. The virtual plant runs the plant subsystems models automata with a list of uncontrollable events to execute. These uncontrollable events are generated by the virtual plant if they are enabled.

The solutions to *simultaneity* and *inexact synchronization* involve making the processor fast enough to avoid the problem. Consequently, it is important to know how fast the controller reads its inputs and reacts to these stimuli. In addition it is important to know how fast the controlled system generate stimuli. The present work reduces the likelihood of the problems still occurring, solves the choice problem and approaches implementation methods in microcontroller environment.

In all previous tests the supervised system had a expected behaviour and none of the cited problems had taken place. By those tests we have validate the solution presented. In order to have a tool which implements this approach, we extended the *Nadzoru* code generator tool presented by Lopes et al. (2011). This tool is open source, free of charge and it is available at *http://gitorious.org/nadzoru*.

6 CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

The problems presented by Fabian and Hellgren (1998) in the PLC environment also exist in the microcontroller environment. The solutions presented by Leal et al. (2009), Cruz et al. (2009), Cruz (2011) and Leal et al. (2012) for PLC can be applied in the microcontroller environment. Our work extends these solutions making a set of one which helps to make a customized code for real needs for each kind of appli-

cation.

The use of local modular approach represents a low memory space required solution. To maximize this low cost the memory safe approach represents a compact way to store the supervisors data.

6.2 Future Works

In future works the authors aim to achieve a more significant reduction in the space memory used in the memory safe method. Another research opportunity is to verify the possible use of a reduced monolithic supervisor. It is also necessary to continue developing *nadzoru* to extend its features.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the Santa Catarina State University.

References

Barreta, R. D. and Torrico, C. R. (2008). Máquinas de mealy e moore para implementação de controle supervisório de sistemas a eventos discretos em microcontroladores, *XVII Congresso Brasileiro de Automática - CBA2008.* .

Brandin, B. A. (1996). The real-time supervisory control of an experimental manufacturing cell, *IEEE Transactions on Robotics and Automation, v. 12, n. 1, pp. 1-14* .

Cruz, D. L. L. d. (2011). *Metodologia para implementação de controle supervisório modular local em controladores lógicos programáveis*, Master's thesis, Santa Catarina State University.

Cruz, D. L. L. d., Leal, A. B., Rosso Jr., R. S. R. and Rozario, J. G. d. (2009). Proposta de implementação de controle supervisório em controladores lógico programáveis, *IX Simpósio Brasileiro de Automação Inteligente (SBAI 2009). Anais do IX Simpósio Brasileiro de Automação Inteligente, v. 9* .

Diogo, R. A., Santos, E. A., Vieira, A. D., de F.R. Loures, E. and Busettia, M. A. (2011). A computational control implementation environment for automated manufacturing systems, *International Journal Of Production Research* **49**.

Eyzell, J. M. and Cury, J. E. R. (2001). Exploiting symmetry in the synthesis of supervisors for discrete event systems, *IEEE Transction on Automatic Control, vol.46, pp. 1500-1505* .

Fabian, M. and Hellgren, A. (1998). Plc-based implementation of supervisory control for discrete event systems, *Proc. of the 37th IEEE Conference on Decision and Control, v. 3, pp. 3305-3310* .

Hu, H. and Li, Z. (2009). Efficient deadlock prevention policy in automated manufacturing systems using shared resources, *International Journal of Advanced Manufacturing Technology* **40**(5).

Hu, H. and Li, Z. (2010). Synthesis of liveness enforcing supervisor for automated manufacturing systems, *Journal of Intelligent Manufacturing* **21**(4).

Hu, H., Zhou, M., and Li, Z. (2011). Optimization of supervisor for deadlock resolution in automated manufacturing systems, *IEEE Transactions on Automation Science and Engineering* **8**(4).

Hu, H., Zhou, M. and Li, Z. (2012). Liveness and ratio-enforcing supervision of automated manufacturing systems using petri nets, *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* **42**(2).

Leal, A. B., Cruz, D. L. L. and Hounsell, M. S. (2009). Supervisory control implementation into programmable logic controllers, *14th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA* .

Leal, A. B., Cruz, D. L. L. and Hounsell, M. S. (2012). *Manufacturing System*, Vol. 1, INTECH, chapter PLC-based Implementation of Local Modular Supervisory Control for Manufacturing Systems.

Lopes, Y. K., Harbs, E., Leal, A. B. and Rosso Jr., R. S. R. (2011). Proposta de implementacao de controle supervisorio em microcontroladores, *X Simpósio Brasileiro de Automação Inteligente (SBAI). Anais do X Simpósio Brasileiro de Automação Inteligente, v. 10* .

Malik, P. (2002). Generating controllers from discrete-event models, *F. Cassez, C. Jard, F. Laroussinie, M. D. Ryan, Proc. of MOVEP* .

Queiroz, M. H. and Cury, J. E. R. (2000). Modular control of composed system, *Proccedings of the American Control Conference, Chicago* .

Queiroz, M. H. and Cury, J. E. R. (2002a). Controle supervisório modular de sistemas de manufatura, *Sba Controle & Automação* **13**(2).

Queiroz, M. H. and Cury, J. E. R. (2002b). Synthesis and implementation of local modular supervisory control for a manufacturing cell, *Proccedings of the Workshop on Discrete Event Systems, pp.103-110* .

Queiroz, M. H. d. (2004). *Controle Supervisório Modular e Multitarefa de Sistemas Compostos*, PhD thesis, Universidade Federal de Santa Catarina.

Ramadge, P. J. G. and Wonham, W. M. (1989). The control of discrete event systems, *Proceedings IEEE, Special Issue on Discrete Event Dynamic Systems* **77**: 1202–1218.

Ramadge, P. J. and Wonham, W. M. (1987). Supervisory control of a class of discrete event process, *SIAM J. Control and Optimization* **25**(1): 206–230.

Silva, D. B., Vieira, A. D., Loures, E. F. R., Busetti, M. A. and Santos, E. A. P. (2011). Dealing with routing in an automated manufacturing cell: a supervisory control theory application, *International Journal Of Production Research* **49**(16).

Sipser, M. (2005). *Introduction to the theory of computation*, 2 edn, Course Technology.

Wonham, W. M. and Ramadge, P. J. (1988). Modular supervisory control of discrete event system, *Mathematics of control, signals and systems, vol.1, pp. 13-30* .

Yalcin, A., Khemuka, A. and Deshpande, P. (2005). Modelling inter-task dependencies and control of workflow managements systems based on supervisory control theory, *International Journal Of Production Research* **43**(20).

Zhong, H. and Wonham, W. M. (1990). On the consistency of hierarchical supervisionin discrete-event systems, *IEEE Transction on Automatic Control, vol.35* .