Decentralized Execution of Sequential Function Charts

Ramón Piedrafita, Member, IEEE. José Luis Villarroel

Abstract—Programmable Logic Controllers (PLCs) are widespread in the control of industrial production systems. For their specification and the implementation of PLC programs, the language Sequential Function Charts (SFC) are increasingly used in industry. The reaction time of a PLC is a fundamental matter in discrete event control systems. Most industrial PLCs run their programs in an interpreted and centralized manner. The PLC reads the inputs, runs the SFC interpreter (also called coordinator in this paper) and writes the outputs. In this work we propose and analyze a new technique for the execution of SFC programs, the decentralized implementation, where several interpreters execute the SFC program of the PLC in concurrence If the SFC program is developed in several macrosteps, each coordinator will be in charge of a part of the set of macrosteps. The coordinators share a unique CPU, executing the SFC in concurrence. This technique is known as concurrent coordinators and was initially proposed to implement Petri nets in [4].

Also, we propose a new technique that allows us to compare centralized and decentralized implementations, and within the decentralized ones to choose the optimal degree of decentralization. In other words, to choose the optimal number of interpreters, with the aim of reducing the SFC execution time.

The implementation of this paper is as follows. In Section II, we review the implementation techniques for SFCs. The decentralized implementation of SFCs is described in Section III. Section IV illustrates the results of the evaluation for the different implementation techniques, while Section V introduces a new technique with the aim of optimizing the degree of decentralization and lastly, section VI is devoted to the main conclusions.

I. INTRODUCTION

Programmable Logic Controllers (PLCs) are widespread in the control of industrial production systems, and Sequential Function Chart (SFC) is one of his main programming languages [1]. By using SFC, the control logic can be specified in an intuitive way. Sequential, parallel and nested processes can be represented graphically, and subfunctions given in any of the other PLCs languages can be embedded. Similar to Grafcet [2], the PLC program is organized into a set of steps and transitions connected by directed links. Associated to each step is a set of actions, and with each transition a transition condition.

The SFCs are binary Petri nets with an interpretation for the control of industrial systems [3]

• Immediate actions are associated to the deactivate and activate steps (e.g., control signal changes, code execution)
• Level control signals are associated to active steps.
• Predicates are associated to transitions, that are additional preconditions for the firing of enabled transitions. Predicates are functions of system inputs or internal variables.

Currently most industrial PLCs run their programs in an interpreted and centralized manner. The PLC reads the inputs, runs the SFC interpreter (also called coordinator in this paper) and writes the outputs.

In this work we propose and analyze a new technique for the execution of SFC programs, the decentralized implementation, where several interpreters execute the SFC program of the PLC in concurrence If the SFC program is developed in several macrosteps, each coordinator will be in charge of a part of the set of macrosteps. The coordinators share a unique CPU, executing the SFC in concurrence. This technique is known as concurrent coordinators and was initially proposed to implement Petri nets in [4].

Also, we propose a new technique that allows us to compare centralized and decentralized implementations, and within the decentralized ones to choose the optimal degree of decentralization. In other words, to choose the optimal number of interpreters, with the aim of reducing the SFC execution time.

The organization of this paper is as follows. In Section II, we review the implementation techniques for SFCs. The decentralized implementation of SFCs is described in Section III. Section IV illustrates the results of the evaluation for the different implementation techniques, while Section V introduces a new technique with the aim of optimizing the degree of decentralization and lastly, section VI is devoted to the main conclusions.

II. IMPLEMENTATION OF SEQUENTIAL FUNCTION CHARTS

In the execution of a SFC it is necessary to determine which transitions can fire, and fire them making the state of the SFC evolve. It will also carry out the actions programmed in the steps. The algorithm to determine which transitions are enabled and can fire is important because it introduces some overload in the controller execution and the reaction time is affected.

An analysis of the implementation algorithms of SFCs has been carried out in [5]. In this work we have implemented several algorithms corresponding to different enabled transition search techniques:

• Brute Force (BF).
• Deferred transit evolution model (DTEVM).
• Immediate transit evolution model (ITEVM).
• Static Representing Places (SRP).
• Enabled Transitions (ET).

In the Brute force algorithm all the transitions are tested for firing. Brute Force algorithms do not try to improve the search of enabled transitions. Works such as [6][7][8] belong to this implementation class.

Like BF, in the algorithm Immediate Transit Evolution Model (ITEVM) all the SFC transitions are tested for firing [9]. However, the Deferred Transit Evolution Model

In this paper (and writes the outputs).
only performs the testing of the transitions descending from the active steps, thus improving the Brute Force operation.

In the algorithm Static Representing Places (SRP) [10] each transition is represented by one of its input places, the Representing Place. The remaining input places are called synchronization places. Only transitions whose Representing Place is marked are considered as candidates for firing.

In the algorithm Enabled Transitions only totally enabled transitions are tested. This kind of technique is studied in works such as [11][12].

In the following paragraphs, we present the SRP technique in more detail to illustrate how all techniques have been encoded. ET algorithm can be consulted in [11][12] and ITEVM and DTEVM can be consulted in [9].

A. Algorithm Execution Cycle

Program 1 presents the basic treatment cycle of the coordinator for SRP technique.

```
loop forever
  Executeactionswithfallingedge();
  Executeactiveactions();
  while elements in ARSL do
    Rstep = next_element (ARSL);
    Transitionsrepr= Rstep.transitionsrep;
    while T in Transitionsrepr do
      if enabled (T) and predicate(T) then
        // transition firing
        // update ARSL and ASSL
        Demark_input_steps (T, ARSL, ARSL);
        Transitionsfired.add(T);
        Break () ;
      end if;
    end while;
  end while;
  while T in Transitionsfired do
    // update ARSLnext and ASSLnext
    Mark_output_steps(T,ARSLnext,ASSLnext);
  end while;
  ARSL.update(ARSLnext);
  ASSL.update(ASSLnext);
  Clear(ARSLnext); Clear(ASSLnext);
end loop;
```

Program 1. SRP Treatment Loop

In the SRP technique, the ARSL list contains the active representing steps and the ASSL list the active synchronization steps. The output transitions of a marked representing step are verified for enabling. If a represented transition fires, the verification process ends because the rest of represented transitions become disabled (effective conflict). The lists ARSLnext and ASSLnext are built with the steps that become marked in a treatment cycle. Finally, ARSL and ASSL are updated with ARSLnext and ASSLnext respectively.

Centralized implementation techniques are based on a cyclic treatment (see Program 1). The main loop goes through the lists using an algorithm that depends on the executed technique. The coordinator cycle execution time depends on the size of these lists. In the case of SRP the size of the ARSL and ASSL lists depends on the current SFC marking. It determines the number of enabled transitions and the number of active steps. The size of the ARSLnext and ASSLnext depends on the number of transitions that fire in the cycle. Thus, the execution time depends on the evolution of the SFC marking and on the active SFC part, the SFC structure and the sequence of events. As algorithms use different lists, their execution times will be different.

III. DECENTRALIZED EXECUTION

We have implemented the techniques in the Java language using the Java Real-time extension [13] and following some ideas presented in [4][14][15]. The real time extension of Java provides the necessary aspects for the programming of real-time systems, e.g., pre-emptive planning based on static priorities, asynchronous transfer of control, real-time high resolution clocks, and the possibility of execution over the Java garbage collector. In our implementations, we used the Real Time Java Virtual Machine JamaicaVM v2.7 [16]. The target hardware was a personal computer with Pentium IV processor at 1.7GHz, running Red Hat Linux 2.4.

In the work [5] the execution of the SFCs was carried out in centralized form. A unique coordinator is in charge of executing the SFC program. The coordinator loads the SFC, creates the necessary data structures and next executes the control cycle, firing the transitions and making the SFC state evolve.

In the present work the execution of the SFCs is made up in decentralized form. Several coordinators execute the SFC program in concurrence. In the execution platform, the Real Time Java Virtual Machine JamaicaVM, each coordinator is implemented in a Periodic Real Time Thread of high priority. The execution is made in a single processor and the coordinator threads are scheduled following a static priorities policy without round-robin.

In the implementations developed here, the program loads the SFC structure from an XML file generated by an SFC editor. The implementation is independent of the SFC, and therefore is an interpreted implementation.
If the SFC program is structured in macrosteps, the execution of macrosteps can be distributed between several coordinators.

A common case in industrial control is as follows: one PLC is dedicated to carrying out the control of a set of machines, and one macrostep is dedicated to each machine. The SFC that controls the machine is nested inside this macrostep. If the machine has a sequential behavior, the program is a sequential SFC SEQ (see Fig. 1.a) (possibly with alternatives) and the concurrent control of several machines is made up by executing several sequential SFC, i.e., by executing a concurrent SFC PAR [5] (see Fig. 1.b).

IV. PERFORMANCE ANALYSIS

The decentralization factor is defined as being the number of coordinators which execute the SFC in concurrency. For example, if the SFC is made up of 50 processes with a decentralization factor of 5, 10 processes will be executed in each one of the 5 coordinators.

\[ T_{p,fd,tr} \] is the execution time to execute \( p \) sequential processes with an \( fd \) decentralization factor when a total of \( tr \) transitions fire. The centralized execution has a decentralization factor 1 \((T_{p,1,2000})\). Maximum decentralization will come about when each coordinator only executes a sequential SFC \((T_{p,p,2000})\) and \( fd=p \). In this case, the number of coordinators are equal than the number of sequential processes.

\[ T_{p,tr} = (fd-1)*T_{p,1,2000} + T_{p,tr} \]

The total execution time of the decentralized execution shall be the sum of the execution time of the coordinators amongst which the execution of the SFC is distributed, plus the time necessary for the context switch and management of threads by the virtual machine. \( T_{cs} \) is the time for context switch and management of threads between two threads in the Java Jamaica virtual machine.

A. Tests with immediate firing of transitions

In the first tests developed, the predicate or condition associated to each transition is considered true, meaning that the transitions will fire immediately. The tests force the situation of maximum overload of algorithms, since in all cycles the maximum number of enabled transitions is fired.

Fig. 2. shows the results of the performance tests. The X axis represents the number of processes of the SFC and the Y axis represents the total execution time (in nanoseconds) of 2000 transition firings.

Fig. 2.a shows the test results for the centralized execution \( (T_{p,1,2000}) \) of the PAR model with the algorithms ET, SRP, DTEVM and ITEVM. PAR SFCs have \( p \) sequential processes without interactions, and in each cycle there are \( p \) enabled transitions which can fire simultaneously and \( p \) active steps. In this test, the best algorithm is SRP for SFCs from 1 to 18 processes and ET for SFCs from 19 to 100 processes. DTEVM presents poor performance, due to the active steps search in all the cycles and the dual passage in the enabling test [5]. However, when the size of the SFC increases, it presents better performance than SRP due to the simplicity of the algorithm, since it works with fewer lists.

Fig. 2.b shows the test results for the decentralized execution of the PAR model with the algorithm SRP. Five cases are composed: the centralized implementation \( (T_{p,1,2000}) \), the decentralized execution with two coordinators \((RPdesc2)\), with four \((RPdesc4)\), five \((RPdesc5)\), and ten coordinators \((RPdesc10)\).

Finally, in Fig 2.c, the centralized and decentralized executions for the ET algorithm are shown.

In a decentralized execution with ten coordinators, ten context switches are produced in each control cycle, and the virtual machine must manage ten threads. In our test platform, the time measured for context switch and thread management \((T_{cs})\) is 0.115 ms. This time is included in Fig. 2.b and c.

If we observe Fig. 2.b, in which the execution is carried out with the SRP algorithm, in SFCs with few processes, centralized implementation is better than decentralized. For example, for a decentralized execution with two coordinators from 10 or more processes, the decentralized execution time is less than the centralized one.

If the execution is carried out with 10 coordinators from 20 or more processes, the decentralized execution time is less than the centralized one, this being due to the overload resulting from the context switches.

For a number of processes greater than 20, behavior improves as the number of coordinators increases, although execution time does not drop if the number of coordinates is greatly increased, due to the overload of the context switches. Fig. 2.c shows similar results for the ET algorithm.

As can be seen, in the SRP execution of one SFC of fifty processes, the time for firing two thousand transitions in a centralized execution is 900 ms. However, in a decentralized execution with ten coordinators, the time downs to 434 ms.

If one SFC of 50 processes is executed decentralized with ten coordinators, each coordinator executes five processes, therefore, the coordinator can only fire five transitions each cycle, and to fire two thousand transitions 400 context switches are produced. For the global sum of two thousand transition firings, each coordinator contributes with two hundred firings.

\[ T_{30,10,2000} = 10 * T_{1,1,2000} + 400 * T_{cs} \]

Another example, in the SRP execution of one SFC of ten processes, the time for firing two thousand transitions in a centralized execution is 430 ms. In a decentralized execution with ten coordinators, the necessary time is 650 ms.
If one SFC of 10 processes is executed decentralized with ten coordinators, each coordinator executes one process, therefore, the coordinator only can fire one transition each cycle, and to fire two thousand transitions 2000 context switches are produced. For the global sum of two thousand transitions firings, each coordinator contributes with two hundred firings.

$T_{10,10,2000} = 10 \times T_{1,200} + 2000 \times T_{tr}$, and

$T_{1,200} = \frac{T_{10,2000}}{10}$

result

$T_{10,10,2000} = T_{1,2000} + 2000 \times T_{tr} = 420 + 2000 \times 0.115 = 650\text{ ms}$

When an SFC execution is decentralized amongst several coordinators, these will execute SFCs with few processes. As can be seen in Fig. 2.a, SRP has a lower execution time than ET for SFCs from 1 to 18 processes. In consequence, in decentralized coordinators execution will be quicker executed with SRP than with ET.

This is the reason why in the decentralized execution with five or more coordinators the SRP is better than ET, while in the centralized execution, the best algorithm is SRP for SFCs from 1 to 18 processes and ET for SFCs from 19 to 100 processes.

B. Tests subjected to events

In industrial control it is very common that, in many cycles, the events do not reach the SFC, and so no transition is fired. When they are fired, their quantity is variable. We can therefore differentiate between two operation regimes:

- Without events regime (static). No transitions are fired and the algorithm only runs the enabling test.
- With events regime (dynamic). Transitions are fired and the algorithm must run all the phases: enabling test, firing and updating of lists.

In order to analyze these situations, a second type of test has been designed: the tests subjected to events. In these tests the number of transitions which can be fired per cycle varies from zero to the maximum concurrency supported by the SFC (maximum number of transitions which can fire concurrently). For example, in PAR SFCs from 0 to p transitions can be fired, where p is the number of processes. Fig. 3 shows the results of these tests in a centralized execution of PAR SFCs. The vertical axis represents the cycle time of the algorithm in nanoseconds. The other two axes are the number of SFC processes and the number of transitions fired per cycle.

The figure shows that SRP presents good performance when few transitions are fired, but bad performance when the number of fired transitions per cycle increases. ET presents good performance when many transitions firing, otherwise it is worse than SRP. DTEVM presents poor performance in the static regime. However, it is better than SRP when the number of fired transitions is high. It can be concluded that the performance of techniques depends also on the dynamics imposed by the controlled system.
The aforementioned tests have been carried out in a centralized manner. The centralized execution has a decentralized factor 1. Fig. 4 presents the execution with the ET algorithm in the centralized case, decentralized with two coordinators and decentralized with ten coordinators.

\[ T_{40,2,40} = 11.36 \text{ ms}; \ T_{40,4,40} = 9.62 \text{ ms}; \ T_{40,6,40} = 9.17 \text{ ms}; \ T_{40,8,40} = 8.82 \text{ ms}; \ T_{40,10,40} = 8.94 \text{ ms} \]

For \( fd=8 \) the minimum execution time is achieved, and therefore the minimum period for the controller is:

\[ T_{\text{core}} \geq T_{40,8,40} = 8.82 \text{ ms} \]

As conclusion, the decentralized implementation can reduce drastically the controller reaction time.

V. ELECTION OF THE BEST DECENTRALIZED FACTOR

In this section a cost function to determine which decentralization factor is better for a specific SFC PAR is proposed. The decentralization gain function is defined as being the difference between the centralized time \( T_{p,1,\text{tr}} \) and the decentralized one \( T_{p,\text{fd},\text{tr}} \).

\[
F_{\text{dec}}(p, fd) = \sum_{tr=1}^{p} (T_{p,1,\text{tr}} - T_{p,\text{fd},\text{tr}})
\]

(2)

If we normalize by dividing by the number of transitions fired

\[
F_{\text{decnorm}}(p, fd) = T_{p,1,\text{tr}} - T_{p,\text{fd},\text{tr}} + \sum_{tr=1}^{p} \left( \frac{T_{p,1,\text{tr}} - T_{p,\text{fd},\text{tr}}}{T_{p,\text{tr}}} \right)
\]

(3)

Fig. 6 represents the \( F_{\text{decnorm}} \) function for the SRP algorithm (only the positive surface is represented). For SFCs with few processes the gain function is negative, which indicates that the best implementation is the centralized one. When \( F_{\text{decnorm}} \) is positive, it indicates that the decentralized implementation is better than the centralized one, obtaining the best decentralization factor for its maximum. \( F_{\text{decnorm}} \) is the cost function proposed.

In order to be able to calculate the function \( F_{\text{decnorm}}(p, \text{fd}) \) it is necessary to know the surface \( T_{p,1,\text{tr}} \) (the cycle time of the centralized execution), and the time of context switch, and manage threads of the execution platform \( T_{\text{cs}} \). The values of the decentralized surface \( T_{p,\text{fd},\text{tr}} \) can be calculated by applying the formula (2).

For a SFC of \( p \) processes the best decentralized factor \( \text{fd}_{\text{best}} \) is the \( \text{fd} \) that maximizes the function \( F_{\text{decnorm}}(p, \text{fd}) \)

With this better decentralized factor, the % of execution time gained with regards to the centralized implementation,
can be calculated with the following function.

\[
G_{\text{exctime}}(p, fd) = 100 \times \frac{\sum_{p=1}^{n}(T_{p,\text{exec}} - T_{p,\text{plc}})}{\sum_{p=1}^{n}T_{p,\text{plc}}} 
\]

This function is represented in Fig. 7. The maximum (60% of execution time gained) is reached for an SFC of 90 processes with an \(fd=9\) and the minimum (40% of execution time lost) for an SFC of 8 processes with an \(fd=5\).

![Graph showing execution time gained vs. processes and decentralized factor.](image)

Fig. 7. % Execution time gained.

### VI. CONCLUSION

In this work we propose and analyze a new technique for the execution of SFC programs, the decentralized implementation, where several interpreters execute the SFC program of the PLC in concurrence. The interpreters share a unique CPU, executing the SFC in concurrence.

Also, we propose a new technique that allows us to compare centralized and decentralized implementations, and within the decentralized ones to choose the optimal number of interpreters, with the aim of reducing the SFC execution time.

As can be seen in the experiments the decentralization can improve the performance of the implementation. It can drastically reduce the reaction time of the PLC when a large number of events come about in a cycle and the number of transitions fired is high. In this situation the decentralized implementation responds much more quickly than the centralized implementation, minimizing the cycle time and obtaining improved control quality.

In a periodic execution the controller must be able to execute the SFC when the maximum number of transitions fire within his period \(T_{\text{cont}}\). In this situation the decentralized is better. Therefore, the decentralized implementation can reduce drastically the controller reaction time.

We are currently working on a new implementation technique which simultaneously contains the advantages of the centralized and decentralized implementations.

### ACKNOWLEDGMENTS

This work was funded by the NERO project DPI2006-07928 of the Spanish Ministry of Science and Technology. We wish to thank Cristian Mahulea, for his help.

### REFERENCES

In the study of SRP algorithm the cycle execution time (CET) can be estimated by the following expression:

\[ \text{CET(SRP)} = T_1 \times \text{size(ARSL)} + T_2 \times \text{FTnumber} + T_3 \times \text{size(ARSLnext)} + T_4 \times \text{size(ASSLnext)} \]

Where FTnumber is the number of fired transitions; \( T_1 \) is the mean consulting time of descending transitions from a marked representing place; \( T_2 \) is the mean time for firing an enabled transition; \( T_3 \) is the mean update time of ARSL with a place of ARSLnext and; \( T_4 \) is the mean update time of ASSL with a place of ASSLnext.

These last three phases of algorithm contain operations of consultation and insertion in lists. Because the size of the list is the number of process of SFC[5]; then, \( T_2, T_3 \) and \( T_4 \) can be considered (in a first approximation) proportionally to the number of process of the SFC.

\[ \text{CET(SRP)} = \frac{T_p}{10} + \frac{K_p p^2 + K_p p^3}{100} \]

For 2000 firings \( \frac{2000}{p} \) cycles are necessary.

\[ \text{CET(SRP)} = \frac{T_p}{10} + \frac{K_p p^2 + K_p p^3}{100} \]

\[ 2000 T_1 + 2000 K_p p + 2000 K_p p \]

In a decentralized execution with ten coordinators, each coordinator executes \( p/10 \) processes, and fires \( p/10 \) transitions

\[ \text{CET}_{\text{DESCEN}}(\text{SRP}) = \left( \left( \frac{T_1}{10} + \frac{K_p p^2 + K_p p^3}{100} \right) + 10 \times T_{\text{exch}} \right) \]

\[ 10 \times \left( \left( \frac{T_1}{10} + \frac{K_p p^2 + K_p p^3}{100} \right) + 10 \times T_{\text{exch}} \right) \]

\[ 2000 T_1 + 200 K_p p + 200 K_p p + 10 \times T_{\text{exch}} \]