Development of Re-configurable Distributed Controllers in 61499 based on Task Schedules Described by UML Diagrams or Gantt Charts

Seno Panjaitan, Tanvir Hussain and Georg Frey

Juniorprofessorship Agentbased Automation, University of Kaiserslautern, Kaiserslautern, Germany,
e-mail : (panjaitan, hussain, frey}@eit.uni-kl.de

Abstract—Achieving re-configurability and flexibility of the control software was one of the main factors that stimulated the emergence of the standard IEC 61499. The current paper is concerned with the re-configurability aspect in terms of control process within the scope of this standard as a task scheduling approach is presented as a probable solution. Task scheduling is proposed as a means of describing the process sequences of a sub-system in an effective and reconfigurable way. When orders of the processes need to be changed for a different mode of operation or even in case of total depreciation of the previous processing sequences, the changes can be actuated through task schedules. This allows reconfigurability of the sub-system to be achieved during runtime without stalling the controlled operations. The task schedules can also be associated with a maximum time limit for particular task executions and this may help in monitoring the controlled hardware’s actions. For the description of the schedules timing and sequence diagram of UML as well as Gantt charts are investigated. The implementation of the proposed approach on a resource-restricted hardware imposes some additional design constraints.

Index Terms—IEC 61499, task schedule, re-configurability, monitoring, UML, Gantt charts.

I. INTRODUCTION

The IEC 61499 standard defines how function blocks (FB) can be used to design and implement the software for distributed industrial process, measurement and control system (IPMCS) [1]. Underpinned by the basic concepts of object oriented software it is appearing to be the most vital element in revolutionizing the development of industrial realm software. This standard has, on one hand, paved the way of using formal and standard techniques like Unified Modelling Language (UML) in the development process of complex industrial Distributed Control Systems (DCSs). On the other hand, it opened up the possibility of achieving flexibility and re-configurability of the system as well as encapsulation so as to enable reusability and hence quality intensiveness.

Encapsulation is one of the salient features that IEC 61499 inherits from the object oriented programming paradigm. Reusability along with encapsulation enables IEC 61499 to ensure a hierarchical design and re-configurability [2]. Moreover, encapsulation is also important in realizing flexible holonic or agent-based manufacturing systems [3], [4] or for introducing plug-and-play in the context of industrial environment [2].

Generally, re-configurability is the ability to repeatedly revise and rearrange the design of a system in a cost effective way. Reconfiguration can be viewed from different levels of sophistication [5] and IEC 61499 also defines a classification of possible reconfiguration schemes that can be implemented in a multitude of controller devices [1]. The reconfiguration that can be realizable using IEC 61499 is based upon encapsulation and hierarchical design. The distributed controllers should hold concise implementations corresponding to Function Blocks (FBs) to be used as well as a table of reconfigurations which takes the information from the hierarchy of Device, Resource and FB management tables. The reconfiguration table should be kept current and available to higher level agents for managing the transitions from one configuration to another.

This paper attempts to present a reconfiguration approach in terms of task schedules. Schedulers on each of the distributed controllers allocate and manage the tasks of constituent FBs. Each of the FBs is allocated to a particular actuating task and the schedule of the task can vary in accordance with the state or mode of operation of the controlled system. The interface of the scheduler even allows, through a Human-Machine Interface (HMI) to activate changes during exigencies. It is, therefore reasonable to maintain a repository of possible pre-planned task schedules which may suffice the demand of probable situations and a higher-level intelligent agent may pick up and impose the appropriate schedule according to the situation on its arrival.

As long as task schedules are concerned the sequences of tasks are often explicitly dependant upon the completion of its pre-cursors. When the control is used for a real time system hard or soft time constraints apply on the tasks. Hence, in the proposed approach it also includes timeliness monitoring into the scheduler. It is not only useful for time-critical processes which have hard or soft time bounds, but also for other processes where longer time expenses may imply malfunctions.

The rest of this paper is arranged as follows. Section II introduces the task scheduling strategy and its role in reconfiguration of a control system. Section III describes the development of the control software from the specification of the requirements to the identification of the FB elements. Section IV presents the methodologies for planning the task schedules. In Section V an implementation of the proposed scheduling is depicted and in Section VI modifications of the implementation for coping up with the system requirements are described. At the end Section VII draws a conclusion and lights up the prospects of the presented work.
II. TASK SCHEDULING STRATEGY

In process and manufacturing industry task scheduling refers to the allocation of tasks to appropriate machines so that particular constraints are met. In the context of this work task stands for a defined operation in the controlled plant for running a desired activity. To actuate a particular task a machine has to run a specific control algorithm. Therefore the execution of a task means the execution of a particular algorithm. Since instantiation or re-sequencing of the algorithms requires stalling of the operations and/or considerable amount of time it is preferable to have predefined algorithms on the controller. The approach of using task scheduling in IEC 61499 compliant DCS can serve as a means of imposing easily changeable execution sequences of some algorithm modules without facing the above mentioned difficulties. It can be helpful in dynamically shifting to a new mode of operation or in redefining the sequence of operations.

The scheduling approach of achieving re-configurability is based on the idea that if the controllers of the DCS are developed on the basis of a functional division, a scheduler FB of composite or basic composure can control the sequence of actions to be actuated by the FBs on each of which a particular functionality is mapped. As long as the division of controlled elements assigned to each of the controller entities does not alter, this approach can prove beneficial in achieving re-configurability through re-scheduling of the tasks.

In developing a system of distributed controllers so as to achieve competence with the scheduler, it is necessary to recognize a proper classification of the entities in terms of functionalities. This can be done using UML [6]. The scheduler is reusable and the planning of the task schedules can be done using sequence or timing diagram of UML or by using Gantt charts popular in project management. The following part of this chapter puts a light on the use of the above mentioned tools in planning task schedules while a more detailed description of those within the context of an example is presented in Section IV.

The sequence diagram of UML can be used at various phases of development to reveal interactions. Since, it is a useful tool in manifesting requirements, even with time constraints – where needed – it can be used for planning the task schedules. The timing diagram is also one of the UML diagrams and is isomorphic with sequence diagrams [7]. The essential difference between the two diagrams is that the latter emphasizes changes in values or states over time while the former emphasizes sequences of message exchange. Gantt charts are a tool of rather different domain and are suitable for the use in project management, production planning and interactive schedulable applications [8]. Its ability to represent timeliness, concurrency and subject interdependencies is good reason to try its use in planning task schedules.

III. DEVELOPMENT OF THE CONTROL SOFTWARE

For developing the control software a part of the didactic production system described in [9] as the controlled subject is considered. Fig. 1 shows a schematic representation of it. This part of the production system is called processing station as it processes cylindrical work-pieces of different types. The processing differs for different types of work-pieces and in different operating modes the station should operate differently following particular specifications.

During the rest of this paper the explanation considers that the concerned control software is developed so that it satisfies the following requirements.

1. During normal operation the station will perform different operations upon distinguishing the appearance of a black or non-black cylindrical work-piece on position 1 of the rotary table. A black work-piece will be drilled at position 2, tested at position 3, and moved further to position 4 on the rotary table so that it can be picked up by the next station. Non-black work-pieces will be moved to position 4 without drilling and testing. However they will also be moved through positions 2 and 3 because the table can only be rotated in one direction.

2. Before going into the normal operating cycle the station will undergo initialization. During this mode of operation the components will all move to their initial positions, i.e. the drill motor will go to a stand-still, the clamp, the drill-module and the test module will be pushed back.

3. In case that a fault occurs or an emergency arrives the station will move to a safe emergency stop mode.

4. The station should undergo a dead-head mode before going to a normal halt. In this mode the station will not start processing any further work-piece that is not already undergoing processing and after finishing the started work-pieces will move to initial stand-still position.

5. In case that the work-pieces from previous operations are left over on the machine it may be cleaned up as the operator drives it to a clean-up mode.

This is only an informal description of the requirements, while a formal analysis of it reveals the functional elements. The functional elements should be realized through a FB encapsulating the algorithms for actuating the components of the station.
The UML class diagram of Fig. 2 shows the identified classes and those of the lowest inheritance hierarchy are converted to the concerning FBs. It is also possible to take advantage of the similarity through implementation of parent classes to FBs and then including the more specialized features through encapsulation of one or more additional FBs in a composite FB. Similar composition is also applied when the classes are rather specialized and bear quite intricate interrelation. For example, considering the intricate relationship have grouped the constituent FBs of the drill module together to form a composite FB and this is shown in Fig. 2 through aggregation of the constituent classes. The classes corresponding to the FBs inherit not only the class that represent the respective functionality oriented control but also the ScheduleExecutor class that offers the ability to handle scheduled tasks with time constraints.

The functional elements, each of which should be realized through a FB encapsulating the algorithms for actuating the components of the station, are listed in Table 1. The above mentioned FBs along with the scheduler FB and the Service Interface Function Blocks (SIFBs) for communicating with other resources and with the peripherals of the controller constitute the IEC 61499 compliant resource model for controlling this particular station.

<table>
<thead>
<tr>
<th>FB</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conveyor</td>
<td>Switching on/off the concerned motor</td>
</tr>
<tr>
<td>Rotary table</td>
<td>Switching on/off ensuring arrival at certain positions</td>
</tr>
<tr>
<td>Testing module</td>
<td>Push/pull the cylinder</td>
</tr>
<tr>
<td>Drill Module (Composite FB)</td>
<td>Push/pull the clamp cylinder</td>
</tr>
<tr>
<td>Drill cylinder</td>
<td>Push/pull the drill cylinder</td>
</tr>
<tr>
<td>Drill machine</td>
<td>Switch on/off drill motor</td>
</tr>
</tbody>
</table>

### IV. PLANNING THE TASK SCHEDULES

As mentioned previously task schedules can be planned using sequence diagrams, timing diagrams or Gantt charts. The following parts of this chapter discuss how these diagrams can be used in planning the task schedules.

#### A. Using sequence diagrams

Sequence diagrams are able to show interactions between various classes. In the context of FB based developments interactions are translated to event occurrences that trigger the FBs to perform according to the Execution Control Charts (ECC). In terms of stereotyped notes sequence diagrams can also manifest the time constraints concerning execution, stimulus occurrences, start or end time bounds etc. [7]. For different modes of operation or for situations which need rescheduling there is a need to use a new sequence diagram so as to represent a new scenario of operation.

The sequence diagram of Fig. 3 shows the interactions during normal mode of operation and when there is a black work-piece detected at the entrance of the station. The information regarding type and colour of the work-piece is known while this processing station undergoes its operation. Time labels are attached in the form of notes with various execution instances. The significance of these labels is that they hold the maximum allowed time limit for the execution of the process. In case the time is over, the scheduler assumes a malfunction of the corresponding controlled module in the station and gives the user a corresponding message. The SIFB entities are excluded from the sequence diagram for the sake of simplicity and spatial limitations. It is rather assumed that the class representations conceptually encapsulate the concerning actuator, sensor and time classes along with the controller class.

With the consideration that the scheduler and the concerned controller FBs are on the same computer which is deployed on a network-enabled controller named

![Fig. 3 Sequence diagram corresponding to normal operation with black work-pieces](image-url)
NETMASTER [10], none of the messages in the sequence diagram of Fig. 3 needed time tags manifesting constraints for sending and receiving messages.

The sequence diagrams generally emphasize the message exchanges between objects. In this work revealing the message exchanges is not of prime importance since it translates into event and data connections between the FBs which are kept fixed throughout the variation of the schedulable scenarios. Moreover, it is quite inadequate to plan the timing constraints of concurrently running operations with the help of the timing notations included in sequence diagrams.

### B. Using timing diagrams

Timing diagrams are rather common in designing electronic state machines. The horizontal axis represents the passage of time while on the vertical axes object states or attribute values are captured. Fig. 4 shows a form of timing diagram corresponding to the normal operation of the station for black work-pieces. A detailed task timing diagram is shown in Fig. 5. This diagram additionally shows the initiation, execution and dwell times of the individual tasks.

The information contained in the timing diagram regarding deadlines, and probable execution time may help in making assumptions about the complete process time. Furthermore, for concurrently running tasks which are precursors to a single task or another group of concurrent tasks the representation of the timing diagram is rather helpful in calculating the waiting times or slack times for certain tasks.

It is also notable that timing diagrams can more efficiently reveal the timing profiles of a process with greater depth as compared to the sequence diagrams. While a single diagram suffices to reveal the timing requirements of a process of hierarchically lower level alongside the one of higher level, sequence diagram may need to provide a new scenario to reveal it.

### C. Using Gantt charts

Gantt charts are useful tools for analyzing and planning complex projects. Introduced by Henry L. Gantt it had been primarily used in computer aided project management. Though it was recognized as a tool for optimizing working times of subsystems or individual workers, later it was recognized that the gains through the above mentioned usage will be lost unless it is also used for overall optimization [8].

Gantt charts are helpful in planning prioritized tasks optimally so as to meet the time constraints as well as to plan optimal resource allocation in cases where resource constraints affect task schedules. The charts provide similar benefits as timing diagrams, revealing timing requirements at various depths and then showing the critical timing requirements so that waiting times for concurrent processes can be planned properly. Fig. 6 and Fig. 7 show Gantt charts for the normal operation of the processing stations respectively for the cases when the work-piece is black or not.
In manufacturing industries there are cases where resource management is needed to be optimized through proper scheduling. For example, a conveyor or crane or any other form of transport mechanism might be used by multiple stations and while one of them keeps the transport mechanism busy the others have to wait. In such cases Gantt charts can be used to shape up the schedules so as to ensure that the waiting times are minimal for each of the stations. Thus Gantt charts could be a good alternative when resource constraints are imposed in addition to the time constraints.

V. IMPLEMENTATION OF THE CONTROLLER

The scheduler FB is of prime importance in the implementation of the controller and hence this section begins with a description of it. The scheduler as such does not contain any specific control logic and has a generic structure used to schedule the task. As shown in Fig. 8 the scheduler FB has three event inputs concerning the three modes of operation, namely normal, setup and emergency stop mode and another event input for ensuring initialization. For providing a particular schedule to the scheduler there is a pair of data inputs defined. The pair consists of an array of parameters representing numbers concerning FBs that will be activated during the operation (task_auto, task_F and task_setup) and an array of times representing the allowed time limit for the concerned operation (DT_1, DT_1F, and DT_0). The binary form of the numbers of the first array allows including concurrent tasks in the schedule. Corresponding to each of the numbers in the number array there is a time value in the time array. When the operation is not time-constrained, the attached time is used for monitoring the process. The monitoring is such that if the operation is not finished before the time is over, an error or failure in any of the machines concerned with the operation is implied. There are two data input pairs attached with the input event Auto that corresponds to normal processing operation. These two pairs bear the schedules for two different processing schemes, namely, when there is a black work-piece, and when there is a work-piece of any other type. The Boolean variables at the bottom are for getting sensor information concerning the work-piece and rotor position.

Among the output events of the scheduler FB there are confirmations corresponding to initialization, emergency stop and completion of a given schedule. The output event Start triggers an event demultiplexer for sending out the mode of operation and a selector which triggers single or multiple functionality oriented FBs for completion of an operation.

As an instance of the functionality oriented FBs the one for controlling the drill module is depicted in Fig. 9. The event inputs Start and Estop trigger respectively starting of a scheduled task and emergency stop, i.e., stalling of any continuing tasks. Apart from the data inputs named mode and T_Constr the other inputs are for taking sensor signals in order to recognize the state of the controlled drill module. The output events correspond to confirmations upon completion of algorithms and the output values correspond to actuator signals which are communicated to the hardware via SIFBs. Certainly, the FB actuates an action and then waits till a desired state is reached, indicated by a sensor signal. Some of the tasks in a schedule consist of only one such actuation and sense action while some others may consist of a number of them. When a time-out occurs before completion of the assigned task the Estop output event is fired denoting an erratic situation and further processing is stopped till the operator settles it.

For achieving distributed control the controller programs have been deployed on network-enabled controllers named NETMASTER [10]. The NETMASTER contains a TINI microcontroller which, like most modern microcontrollers, supports high-level languages like Java and C [11]. This provides the advantage of programming applications for these microcontrollers without specialized expertise of proprietary or assembly-like low-level languages. Moreover, using appropriate APIs it is possible to access the peripherals.

For applying the controller Java-based Function Block Development Kit (FBDK) is used for the development and Function Block Run Time (FBRT) environment [12] is used for the deployment of the FBs on the NETMASTER controllers. For the time critical operations needed certain modifications as the conventional idea of separating the peripheral access and control algorithm resulted in non-deterministic and inappropriate execution times.

VI. MODIFICATIONS

The NETMASTER controllers (also usually most of the modern microcontrollers) are programmable with high level languages (i.e., Java, C++, etc.). For accessing peripherals of these controllers there are usually built-in APIs available. These APIs are built upon a native interface and they consume costly time especially for real-time applications. Moreover, on many of the platforms this extra time for achieving peripheral access is dependent upon the number of threads running or computing resources occupied, making the time-required for the peripheral access a non-deterministic parameter. It is needed to impose certain modifications of implementation as the previously developed one failed to meet the requirement of timeliness. Within the scope of the above mentioned example it is the rotary table where the timeliness requirements are quite strict. The rotary table should stop at various fixed positions as it operates and as the control application is able to only switch on and off the motor, as soon as it is detected that the table has arrived the position the control application should actuate the action of switching off the motor so that it stops correctly. The response should be actuated in a millisecond; otherwise the rotary table is misplaced so as to hamper the operation of the other modules.

One possible solution could be to encode the part of the program that will be deployed on the controller to contain rather simple and/or low-level code satisfying the timing requirements and then to bridge it with the other programs through adapter interfaces [12]. A quite different approach is taken in this work. The main idea is encapsulation of the functionalities of multiple FBs into one rather complex one. It was rather an attempt to conglomerate appropriate SIFB functionalities with an algorithmic FB so that the resulting FB can detect the state changes and directly actuate accordingly conserving time. From the viewpoint of a DCS this can be termed as encapsulation of some programmatic components in order to achieve better performance or to meet QoS requirements.

Certainly, this new FB does not define its action sequences through Time Sequence Diagrams (TSDs) as in the SIFBs; rather it has an ECC for defining its behaviour. The resulting code generated through FBDK showed faster response as compared to the previous implementations and the control of the rotary table was appropriate in meeting the time constraints.

VII. CONCLUSIONS AND OUTLOOK

In this paper an approach of achieving reconfigurability using task schedules has been elucidated. The proposed task scheduler monitors execution of tasks in predefined sequences. Use of scheduler for execution of algorithms encapsulated in FBs is expected to give better ability to reconfigure in terms of rapid change and rearrangement of the FBs and/or their connections in a system. Moreover, there are scopes to add intelligent nodes in the system which will react in response to situation when a running schedule is needed to be replaced by a new one from a library of schedules.

With the wide-spread attempt of introducing encapsulation of hardware and software components to enable “plug-and-play” in the realm of IPMCS, it can be foreseen that such an encapsulated component may reveal only its interfaces which give a leeway to influence the embedded task schedules. When needed, these interfaces can be used to optimize performance and thus enable achievement of better control and QoS. In this setting the internal structure of the control may remain private to the vendor of the corresponding module.

The first practical studies with the presented approach showed good results. As a next step in this work a method will be searched to automatically derive the scheduler from the respective UML diagrams. This will not only help in avoiding errors in the manual translation but also lead to a much faster development process. Shorter iterations in this process are definitely necessary since – using restricted hardware – the test on the real system reveals additional timing constraints.

VIII. REFERENCES