Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 282708, 12 pages
doi:10.1155/2009/282708

Research Article

Microcontroller-Based Process Monitoring Using Petri-Nets

Marcos R. Frankowiak, Roger I. Grosvenor, and Paul W. Prickett

School of Engineering, Cardiff University, Queens Building, Newport Road, Cardiff CF24 3AA, UK

Correspondence should be addressed to Paul W. Prickett, prickett@cf.ac.uk

Received 29 July 2008; Revised 17 October 2008; Accepted 25 November 2008

Recommended by Wilfried Elmenreich

This paper considers the development of a Petri-net-based modelling tool as a mechanism for process and system monitoring. The use of Petri-nets, which has previously been largely based in the areas of systems modelling and simulation, is shown here to have great potential for deployment as a process monitoring and management application. Interfacing with real-world processes has been achieved in part by introducing a specific set of extensions to the original Petri-net concept. This work has resulted in the engineering of a tool that can be embedded within the process using a microcontroller platform. The potential for such systems to provide low cost, yet powerful process management tools, is becoming increasingly evident, particularly given the ever-improving capabilities of microcontrollers.

1. Introduction

The increasing level of complexity associated to many engineering systems requires the use of design methodologies that enable their representation, comprehension, and analysis. One such method, Petri-nets, was proposed in the early 1960's by Carl Adam Petri. Since then the basic concepts have been developed and explored [1, 2]. Although representing a mathematical formalism, the main feature of Petri-nets is their capability of describing a system's behaviour in a graphical manner making the approach suitable for modelling many engineering applications [3, 4]. Researchers have continuously suggested changes to the original concepts and developed improvements including some aimed at model validation thus supporting enhanced accuracy and reliability [5] and in order to better represent real-life situations [6]. It is not the aim of this paper to present a full consideration of this research, much of which, particularly work aimed at modelling manufacturing systems [7], has been recently classified and reviewed [8]. Other uses of the method include the development of fault detection and isolation methods [9] and its deployment in relation to process and condition monitoring [10]. The above taken together clearly makes the case for considering Petri-net models to be powerful and flexible tools with many potential applications.

The approach described here in is based upon developments to Petri-net theory and application to support the deployment of the designed system model within a microcontroller. This exploits characteristics such as concurrency, sequencing, and synchronisation that make Petri-nets a powerful tool for the representation and modelling of a variety of discrete event systems. Similar developments, such as in the use of the technique for the implementation of programming languages [11] and as a graphical programming method aligned with the acquisition and issuing of real-time signals [12] led to Petri-net based computer (PC) and PLC applications.

The potential of deploying Petri-nets in the creation of microcontroller software design was clearly identified in the early 1990's [13]. The author was able to harness the modelling attributes of Petri-nets, which allowed the incorporation of interrupts, into controller software at the design stage. The advantages and potential of deploying this approach were illustrated in this paper in the context of the operation of a bank of eight switched telephone lines. More extensive discussion of the benefits and associated implementation examples were provided in a later publication [14]. One limiting factor on the deployment of this approach was the computing power of the then available microcontrollers. The author identified the use of multiple processors implementations as a possible resolution of these

limitations. He also suggested ways of designing the software to overcome problems arising due to limitations of the available microcontroller memory. It is with the deployment of Petri-nets within into the much more powerful present day microcontroller platforms that the research undertaken within the Intelligent Process Monitoring and Management (IPMM) Centre has been focussed [15]. The added computational power and enhanced communication facilities thus made available have allowed for the utilisation of more complicated Petri-net models that can obtain their interrupt-related information from a wider range of sources. The work follows the development of a Petri-net-based machine tool monitoring and management system [16], and is based upon the increasing capabilities of microcontrollers to perform data acquisition and analysis functions [17, 18] which have supported machine tool-related applications [19, 20].

2. Petri-Net Representation

A Petri-net is a representation of a system or process in the form of a graphical model, intended to either explain or formally document the subject, often for human interpretation. Formally a Petri-net structure "C" may be taken to comprise of a finite set of places "P" and a finite set of transitions "T." Places equate to a representation of a system state, transitions represent the events occurring in the process execution. The sets of places P and transitions T are disjoint, as defined by

$$P \cap T = 0, \tag{1}$$

where $P(p_1, p_2, \ldots, p_n)$ is a finite set of places, $n \geq 0$ and $T(t_1, t_2, \ldots, t_k)$ is a finite set of transitions, $k \geq 0$.

The relationship between places and transitions may be defined in terms of input "I" and output "O" functions. Peterson [1] described $I(t_j)$ as a mapping of the input places of a transition t_j, while $O(t_j)$ maps the output places of a transition t_j. The relationship between places and transitions, in terms of input and output functions, can be described as follows:

$$\begin{array}{ll} p_i \text{ is an input place of } t_j, & \text{if } p_i \in I(t_j), \\ p_i \text{ is an output place of } t_j, & \text{if } p_i \in O(t_j). \end{array} \tag{2}$$

To support the approach developed later in this work, an extension of these relationships is assumed:

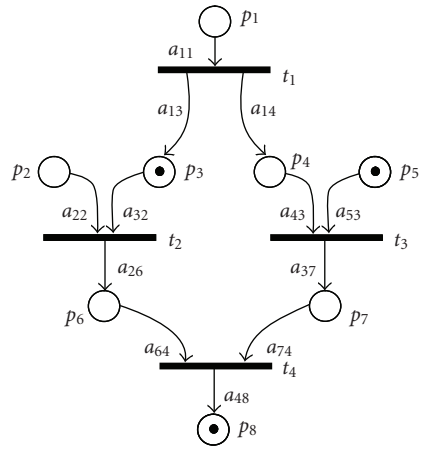
$$\begin{array}{ll} t_j \text{ is an input transition of } p_i, & \text{if } t_j \in I(p_i), \\ t_j \text{ is an output transition of } p_i, & \text{if } t_j \in O(p_i), \end{array} \tag{3}$$

These relationships allow the definition of the Petri-net structure C in the form of the four tuple; $C = (P, T, I, O)$. The Petri-net represents a system in terms of a sequence of events and states; the firing of transitions modifies such states. To enable the execution of these transitions primitive elements know as tokens are introduced. By adding (generating) and removing (destroying) tokens within places the enabling of transitions is achieved. The dynamic of the net structure, and of the system or process being modelled, may thus be enabled by the generation of tokens following specified events and the distribution of such tokens in places. This is known as the Petri-net mapping and is defined by the marking vector "μ." The marking vector and the set of places are closely related; the marking "M" of a Petri-net can be described as $M = (P, T, I, O, \mu)$. The initial marking vector is described as μ^0 and represents the very first marking map of the system. The execution of a Petri-net can then be defined in terms of a sequence of firing transitions $(t_{j0}, t_{j1}, t_{j2}, \ldots)$ and a sequence of marking vectors $(\mu^0, \mu^1, \mu^2, \ldots)$.

Given the initial μ^0 and the sequence of transitions that represent the Petri-net execution, it is possible to determine the sequence of marking vectors. Similarly, given the marking sequence, it is possible to establish the sequence of transitions of the Petri-net execution. The capability of reaching a marking μ^k from a previous marking by means of a sequence of firing transitions is defined as "reachability" and represents a useful property for the analysis of Petri-net models. This relationship can be used to determine the firing sequence vector, since the marking vector is known. Although the main strength of Petri-nets is graphical representation, this sort of mathematical formalism becomes important to support analysis methods to validate a model. Since graphical representations provide a more accessible and easier to understand model of systems, most Petri-net modelling procedures are illustrated using a Petri-net graph. The equivalence between the structure outlined above and a graph is presented in Figure 1.

This graphical representation of the Petri-net as a bipartite directed multigraph depicts the sets of places P, transitions T, and a representation of the input and output functions in the form of a number of arcs "A." These arcs represent the relation between the places and transitions; $A = (a_1, a_2, a_3, \ldots)$. A token, which is represented as a dot within a place in the graphical interpretation, resides in a place and illustrates the current state of the system at any given time; the assignment of tokens to places is the "marking" of the net. Thus, by adding tokens, the ability to execute the Petri-net is achieved. As events are represented by transitions in the Petri-net, the execution of the event occurs when the correct number of tokens has been assigned to the places of the transition's input function. Events are executed by the firing of transitions, whereby the tokens from the input places are removed and token(s) are assigned to each of the output place(s) of the transition. This can only be achieved if the transition is enabled, whereby for each input place p of transition t a marking of at least the weight of the arc from p to t is satisfied.

2.1. IPMM Extensions of Petri-Nets. Over several years, IPMM researchers have developed Petri-nets to introduce a revised management technique for the progression of tokens through the entire model, which can include multiple individual nets. The IPMM strategy was motivated by the need for process modelling in machine tool condition monitoring [16]. In this modified approach, tokens move top-down through the net, originating at a token generator "G" and being disposed of in a token bin "B" at the bottom. The token generator facilitates interactions with processes



$C = (P, T, I, O)$
$P = (p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8)$
$T = (t_1 + t_2 + t_3 + t_4)$

$I(t_1) = (p_1)$	$O(t_1) = (p_3, p_4)$
$I(t_2) = (p_2, p_3)$	$O(t_2) = (p_6)$
$I(t_3) = (p_4, p_5)$	$O(t_3) = (p_7)$
$I(t_4) = (p_6, p_7)$	$O(t_4) = (p_8)$

Figure 1: Graphical and textual representations of Petri-nets.

and events outside the definition of the net, which is intended to equate to events prior to the process under observation. Likewise, the token bin represents the end of the process. By definition there is one token generator for each net but there may be one or more token bins.

By defining the token generator and bin, IPMM modelled nets facilitate a degree of process encapsulation that allows for their existence as part of a larger process without the need to explicitly state the full process definition. The system interpreter needs only to understand that a token will be introduced to the system at the generator and leave at the bin. This ensures an explicit and obvious path through all modelled nets to manage overall net execution. In particular, the requirement for all net paths to terminate at a token bin ensures that modelled processes have a well-defined finish point, and as they can be counted at entry and exit of the net, tokens are less likely to become unnoticeably stalled within the net.

Graphical Petri-net models, as with state diagrams and flow charts, quickly become large in size which can cause significant problems in their comprehension. IPMM nets implement a modular approach by decomposing the overall net into a series of subnets, linked by a control-net. Formally, every IPMM Petri-net modelled system has a single control-net, together with zero or more subnets. The use of subnets is another important concept where in a system can be divided into specific parts, easing the modelling task and improving the potential for analysis. By employing such notation, a set of elements of the Petri-net might be abstracted to a single element to simplify the main net representation. A complicated system based on several hierarchical levels can be represented by nested subnets.

Dividing a process into a series of subprocesses improves both development and comprehension ease without introducing a significant processing overhead. This follows a well-established practice in most nontrivial programming languages of implementing subroutines or subprocedures to perform a task before returning to the main execution of the program. Where the process defined in a subnet is called by different parts of the control-net, a saving of development time and a reducing in the potential for error is experienced. For example, in typical practical applications the execution of an emergency stop procedure could be called from any point of the process. Without subnets this would require either multiple definitions within the net, or a more complicated process definition to ensure it was reachable from every transition. The subnetting allows for a simple call to the subnet in the event of an emergency stop, promoting simplicity, procedure reuse and the inherent verification benefits associated with net standardisation.

Enhancements have also been made to the traditional transition to incorporate a series of both digital and analogue inputs from the modelled process. These are typically induced Boolean variables (direct digital states from switches or conditional range analysis from analogue inputs), which can also be negated to provide additional functionality. Input signals can thus be considered as external sensory or process information entering the model. In practical terms, this could include switches, gauges or, other type of sensors. For simplicity in definition, it is assumed the output value is discrete and easily measurable, and that the number of potential signals is limited to a few values per transition. This process is made possible and practical by the use of a microcontroller to continuously acquire and analyse appropriately conditioned signals and to make decisions regarding thresholds, and so forth. as described later in this paper. The addition of inputs to the basic transitions changes the way in which transitions fire. In addition to each input place of the transition needing to have the prescribed number of tokens as previously outlined, the input signal requirements also need to be satisfied.

From the above it can be stated that Petri-nets can support a modelling method that has mathematical formalism. Up to this point their main strength has been in providing graphical representations. One of the characteristics of the method is the capability of modelling systems as sequences of discrete events and states. To enable the method to better function as a management and monitoring tool for real-life systems the implementation of microcontroller-based nets described below was enacted.

3. The Modelling Approach

To deploy this method for system or process monitoring each place in a net must be taken as being representative of the process state being monitored. Assuming that is a finite number of input signals "k," there will be a set of input conditions, "X" defined by $X = (x_1, x_2, x_3, \ldots, x_k)$ that represent the conditions required to enable the entire set of Petri-net transitions. Given that the entire set of places of the Petri-net are defined as previously by $P(p_1, p_2, p_3, \ldots, p_m)$, then, the firing event of a transition t_j is controlled by two functions:

$$\begin{aligned} S(t_j) &= f(x_{j1}, x_{j2}, \ldots, x_{jw}), \quad w \geq 0,\ w \leq k, \\ Q(t_j) &= f\{I(t_j), \mu\}, \end{aligned} \tag{4}$$

where $I(t_j)$ is the input function of the transition t_j and μ is the Petri-net marking vector outlined in the previous section. The function S represents the logical relation of all the input signal conditions associated to a specific transition and can only assume 0 (false) or 1(true). Thus,

$$S(t_j) = x_{j1} \bullet x_{j2} \bullet x_{j3} \bullet \cdots \bullet x_{jw}. \tag{5}$$

Each x_j represents a condition of a specific input signal in the domain of the specific transition. Therefore, x_j will also assume the logic representation 0 or 1. Considering a transition t_j, which requires an input signal condition x_{jf} to be satisfied, then x_{jf} can be described as

$$x_{jf} = f(y_i, c_{ji}), \tag{6}$$

where y_i is a specific input signal of the entire set $Y(y_1, y_2, \ldots, y_i, \ldots, y_z)$ of the monitored process signals and c_{ji} is the desired status of this signal in the domain of transition t_j. Therefore,

$$\begin{aligned} x_{jf} &= 0 \quad \text{if the status of } y_i \neq c_{ji}, \\ x_{jf} &= 1 \quad \text{if the status of } y_i = c_{ji}. \end{aligned} \tag{7}$$

Consequently,

$$\begin{aligned} S(t_j) &= 1 \quad \text{if all } x_{jf} = 1, \\ S(t_j) &= 0 \quad \text{if any } x_{jf} = 0. \end{aligned} \tag{8}$$

The second condition required to enable a transition t_j is represented by $Q(t_j)$. This is restricted to two options, enabled or not enabled. Hence, the function can also be considered as assuming two logic states: 0 and 1. This results in the following relationships, considering an existing marking μ:

$$\begin{aligned} Q(t_j) &= 1 \quad \text{if } \mu \text{ satisfies } I(t_j) \text{ so that } t_j \text{ is enabled}, \\ Q(t_j) &= 0 \quad \text{if } \mu \text{ does not satisfy } I(t_j) \text{ in order to enable } t_j. \end{aligned} \tag{9}$$

Either $S(t_j)$ and $Q(t_j)$ can prevent t_j from being fired, however both are required to be true in order to enable t_j. This can be expressed as

$$\begin{aligned} S(t_j) \bullet Q(t_j) &= 0, \quad \text{then } t_j \text{ is not enabled}, \\ S(t_j) \bullet Q(t_j) &= 1, \quad \text{then } t_j \text{ is enabled}. \end{aligned} \tag{10}$$

Finally, it can be concluded that

$$\mu^k = \mu^{k-1} + O(t_j) - I(t_j), \quad \text{if } S(t_j) = 1. \tag{11}$$

It becomes clear that the marking of the Petri-net will depend only on its representation in terms of places and transitions, the relationship between them and the initial marking μ^0. A new marking within the Petri-net is only possible if all the required conditions are present together. This makes process monitoring possible by keeping records of the process marking and taking into account the process signals and their required status within the domain of each specific transition of the Petri-net that models the process.

The structures arising from the approach proposed by this research were classified as dynamic and static. Those that change their status during the execution of the Petri-net (i.e., places) were said to be dynamic. Petri-net transitions were considered as static structures. This is because within this approach the transitions define the process as a fixed sequence of events that do not change. Such classification becomes important for the implementation of the approach based on a microcontroller. Static elements can use the program memory, avoiding the use of the data memory, a valuable and limited resource in a microcontroller. This goes some way towards overcoming memory related limitations that may have restricted the deployment of similar approaches [13].

4. The Modelling Elements

In order to enable a process to be modelled as a sequence of events and states, the approach considers the two main elements: transitions and places. Each is provided with specific functionality to allow them to better represent particular modelling requirements. Transitions were defined as a structure that contains the information about the conditions that characterise the specific process event and the process states that will have their status changed following the process event. The characterisation of the process events as transitions effectively enabled the establishment of a Petri-net "skeleton." A collection of static structures can thus be used to describe the process Petri-net in terms of such events. These structures can be made to be autodescriptive to allow a totally independent execution. In considering the representation of these structures there was a need to provide in their description the identification, preconditions (inputs) and postconditions (outputs). To facilitate this approach the following transition structures were defined.

4.1. Ordinary Transitions. These are the basic modelling elements. They must have at least one Petri-net place and one or more digital signal(s) from the process as input conditions. When all the required input conditions are present the transition is fired. This removes tokens from the input places and adds tokens to the output places, thus establishing a new marking vector within the model. They were called "ordinary transitions" due to the fact that they represent the basic structure required in the modelling process. Figure 2 presents this structure diagram. The first

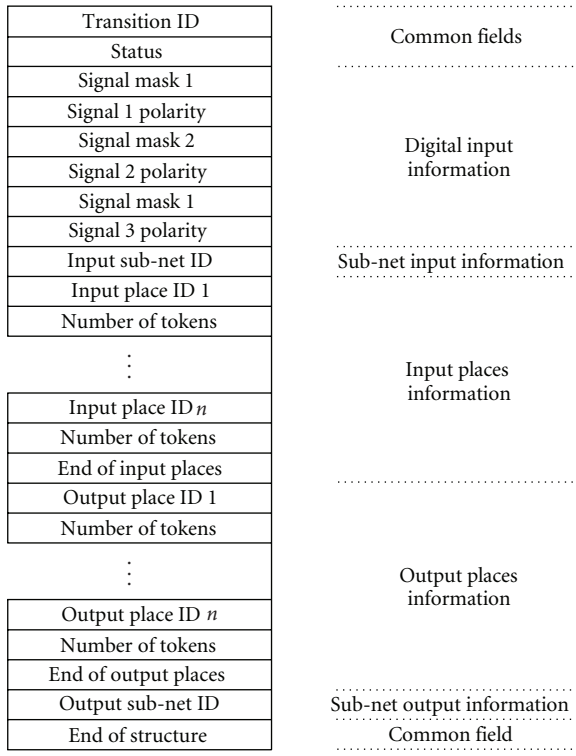


Figure 2: Ordinary transition data structure.

field in the structure is the transition identification, which is numerically represented $(1, 2, 3, \ldots, n)$. The second field provides specific information about the transition. Flags within this field tell the system what sort of transition it is. There is also a flag to indicate whether the firing of the transition should become a public event (i.e., is associated with the transmission of a message), or if it is only of interest within this Petri-net, to update the process marking. These two initial fields (Transition ID and Status) are common to all transition structures.

Ordinary transitions can only handle digital input signals. The following 6 (3×2) alternate fields in the structure provide information regarding these signals and their required status to enable the specific transition. The signal mask reserves 1 bit position for each signal (0 to 7). A bit level 1 at a specific position indicates that this signal must be taken into consideration at the given transition. The signal polarity field indicates the required signal status (0 or 1), that is, the input to the transition required to be logically true or false, and is aligned with the respective bit position in the signal mask field. For simplified implementation, the polarity assigned in the field must be inverted, meaning that it is expressed as 0 when the signal is required to be logically true. The existence of 3 groups of such field pairs indicates the ability of a transition to handle up to 24 digital inputs. This clearly indicates how the advances made in microcontroller capabilities can support the enactment of more complicated Petri-net models and hence support the deployment of these monitoring tools in more challenging applications. For implementation purposes, for those signals that are of no relevance to the particular transition, the corresponding bit position polarity was assigned as 1. This provided a simpler way to test an entire set of 8 inputs simultaneously, rather than individually.

Continuing with the structure definition in Figure 2, the next field is the input subnet (input subnet ID). By definition, if this field is assigned as zero, no subnet input is linked to the transition; otherwise, the appropriate subnet ID code is identified. The collection of input places is the next information in the ordinary transition structure. The input place ID identifies the Petri-net place that is assigned as an input condition of the transition. The number of tokens indicates the "multiplicity" of the arc linking the input place to the transition (a condition becomes true if the assigned number of tokens is found in the place). An ordinary transition may have several input places, each one with its individual multiplicity mark (limited to 255). An "end of input places" field indicates that there are no more input places to be considered in the domain of the specific transition.

Similarly, the next collection of fields within the structure is associated with output places. The output place ID identifies the Petri-net place that should be updated due to the transition firing. The number of tokens represents the multiplicity of the arc linking the transition to the output place, thus indicating the number of tokens the place should receive. The "end of output places" is the mark indicating that there are no more output places linked. The output subnet field identifies, if appropriate, the subnet element requiring notification of the specific transition firing. A value 0 in the output subnet ID field indicates that there is no such a requirement. The way in which the subnet ID is made public is a matter of system implementation. However, considering the nature of such feature and its purpose, a message broadcast method was adopted, allowing other Petri-nets to decide whether or not to make use of the information. Such a method is common in the implementation of distributed systems, provided with local processing capabilities. The final field of an ordinary transition structure is the end mark "end of structure," meaning that there are no more fields in this transition. This field is mandatory in all the transition structures used in this Petri-net approach.

4.2. Analogue Transitions. Analogue transitions were defined to allow the use of analogue signals, together with Petri-net places, as input conditions. Typically, the level of the analogue signal is compared against a predefined threshold or value. This type of transition was developed to support a hybrid digital-analogue monitoring approach. Aside from handling analogue inputs they act in the same way as ordinary transitions. A method that considers two parameters was proposed, in order to characterise such an event. The first of such parameters would represent a threshold and the second an analysis condition. In this way, it becomes possible for the signal being monitored to be acquired and analysed in real time by the microcontroller. Condition-based decisions relating to the enactment of the analogue transition, such

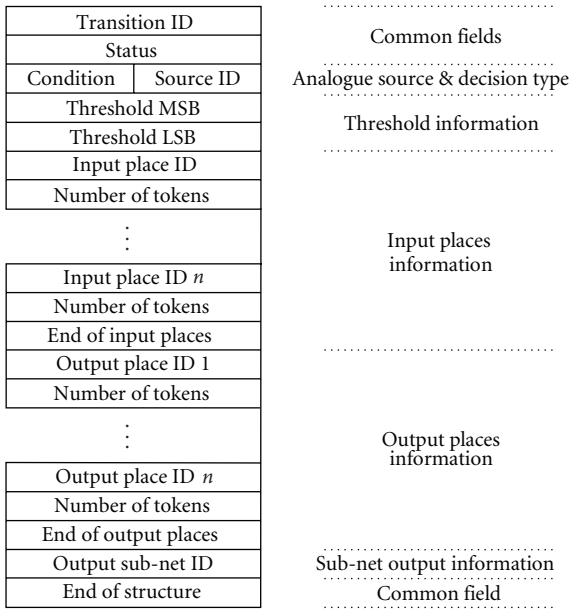


Figure 3: Analogue transition data structure.

as $(\langle, \rangle, =)$, can be made. Importantly it becomes possible capture any data that pertains to faulty enactments for subsequent communication and analysis using the developed IPMM architecture, as outlined in Section 5 below. Recent evolutions in microprocessor technology mean that the nature of the data acquisition and signal processing functions now available can support increasingly sophisticated decision-making tools. Current IPMM work is focussed on frequency and time-based approaches [20, 21]. The results arising from these methods can be incorporated into the monitoring function via analogue transitions. Figure 3 illustrates the data structure of an analogue transition.

The transition ID and status fields follow exactly the same description provided for the similarly named fields in the ordinary transition structure. The condition field is the identification of the comparison method requested. Further implementation details are summarised in Table 1.

The source ID identifies which specific process analogue source should be measured, in order to determine the firing condition. The analogue transition allows only one analogue source as input condition. The next two fields, the threshold most significant byte (MSB) and least significant byte (LSB) of the data structure need to be combined to produce the threshold value required in the comparison test. All the following remaining fields, from "input place ID" to "end of structure," have exactly the same meaning and representation described for the ordinary transition structure. Although linkage to an output subnet field was permitted by the structure, the equivalent input field (input subnet ID) was not considered to be necessary.

4.3. Delay and Output Transitions. Although original Petri-net theory considered that transitions were instantaneous events, the use of the method to model real applications showed the necessity to represent events that take time to execute; when attempting to monitor a process through its signals, there might be occasions where some sort of flexibility would be required. For example, the switching action of an electrical signal can introduce noise that may induce the misinterpretations of the signal's levels. Therefore, this modelling approach has provided a structure that enables the insertion of a time delay, named a "delay transition." Figure 4 defines the delay transition data structure used.

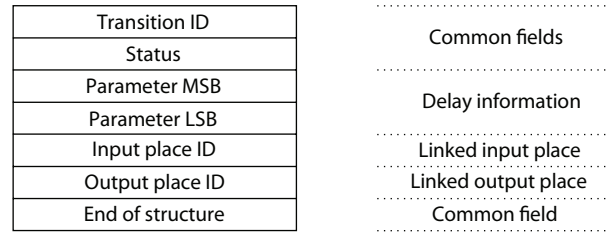


Figure 4: Delay transition data structure.

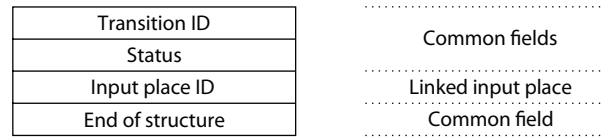


Figure 5: Output transition data structure.

The structure considers only one input and one output place. The firing is enabled by the existence of a single token in the input place, that is, it has arcs that do not support multiplicity. The two first fields in the structure are as previously defined (for ordinary and analogue transitions). The same applies to the last field (end of structure). The input place represents the unique condition required to enable the transition to fire. The output place field indicates which of the Petri-net places should receive a single token when the transition is fired. The delay value is specified via two-field parameters (parameters MSB and LSB). Together, these define the time delay in milliseconds (ms). Immediately after being enabled, the token of the input place is removed. After the delay expressed in the parameter field is elapsed the transition is fired and the output place updated.

The final structure defined was required to enable the monitoring hardware to issue local (hardware) alarms, following a modelled event. Such an element was defined as an "output transition" and its data structure is shown in Figure 5. In this simple structure definition the common fields apply as before; an output transition is enabled by only one input place (input place ID), which requires a single token. Although resulting in a token being removed from the input place, there is no output place to be updated. In line with formal Petri-net theory, it could be assumed that an output transition, when fired, sends a token to a subnet represented by the monitoring hardware/software implementation. The resulting action of such an event is then a matter of system implementation (hardware and software).

4.4. Places. In this modelling approach, the actual condition of any active process state is represented by "tokens" within

Table 1: Petri-net monitoring approach graphic modelling elements.

Field	Size	Representation	Description
Transition ID	8 bits	1 to 254	Petri-net transition identification.
Place ID	8 bits	0 to 254	Petri-net place identification (input & output).
Subnet ID	8 bits	0 to 255	Petri-net subnet identification (input & output).
Number of tokens	8 bits	0 to 255	Arc multiplicity—number of tokens required from an input place or added to an output place.
Status	8 bits	—	Defines transition structure and actions. Detailed in Figure 5.
Signal mask	8 bits	—	Selection of the digital signals considered in the transition domain.
Signal polarity	8 bits	—	Digital signals level, with reversed polarity—default binary 1.
Condition	4 bits	=: 0 000 binary >: 1000 binary <: 0 001 binary	Comparing condition of a nondigital parameter in an analogue data structure.
Source ID	4 bits	1 to 15	Nondigital input parameter identification, representing the signal input in an analogue transition
Threshold MSB + LSB	16 bits	0 to 65535	Value to be considered in the comparison process of an analogue transition.
Parameter MSB + LSB	16 bits	0 to 65535	Delay, in milliseconds, to be performed by a delay transition.
End of input places	8 bits	255	Input places delimiter.
End of output places	8 bits	255	Output places delimiter.
End of structure	8 bits	255	Defines the end of the structure.

the associated place. An empty place, that is, one without any tokens, indicates an inactive state. As the sequence of process events occurs places change condition due to the flow of tokens. Places thus require an identification to distinguish them from each other. Here a continuous numbering method was employed $(1, 2, 3, \ldots, n)$. Each place must provide a container (counter) that holds the number of tokens belonging to the place. The number of tokens will then vary (increase or decrease) during the system's execution. The maximum number is bounded by the container data type size. In considering the use of a microcontroller, the "byte" was selected as the place container data type, since instructions are optimised for the processor's natural data format. Thus, the number of tokens of a place was bounded to 255.

Place "0" has a special meaning for the system implementation. It represents the initial state in the Petri-net, that is, the place that should receive the first token after initialisation. It follows Peterson's [1] suggestion of a "start place" with a token and no tokens elsewhere. A second meaning of place "0" is for a Petri-net reset request. A reset condition is identified by an output place "0," which should result in the system restarting the Petri-net execution (initial start state). The monitoring functions are thus inevitably linked to places. In the approach developed here the structure of the Petri-net linking the places and defined by static transition information was stored in the microcontroller's program memory. The dynamic information relating to the status of the places however was placed in the program memory, making it more accessible and more able to support the monitoring function outlined below. In the initial installations practical considerations limited the number of available places to 254.

5. Implementations Aspects

The nature of the Petri-net monitoring approach outlined above means that implementations do not need to be restricted to a particular platform. The approach could potentially be based on different processors because the way in which each Petri-net is actually executed is mainly a question of software development. However, since one of the main objectives of this research was the proposition of a low-cost monitoring system that can be embedded within systems, work was focused on the use of microcontrollers.

The description of a monitoring task within the system is represented by the transition structure's characterisation of the process events. In terms of an implementation, such a set of data structures is defined as a data table. An element named "end of table" and represented by the numerical "0" identifies the condition that indicates the end of such table. Earlier research deploying Petri-nets in a monitoring function [16] had identified the potential uses of providing a "time-out feature," characterised by a transition failing to fire within a defined period of time. Considering the implementation aspects in a microcontroller environment, it was decided to associate such a feature with places, rather than transitions. Thus, in the approach proposed by this research, "time-out" records will be produced in response to a process state (selected Petri-net place) lasting longer than previously recorded (or defined). Within this proposed monitoring approach, certain Petri-net places assume greater relevance in the implementation of special functions for the assessment of processes. Such places need to be defined in the system implementation as a set, associated to the function they will perform (i.e., to monitor the status of a specific part of the process by means of generating "beginning" and

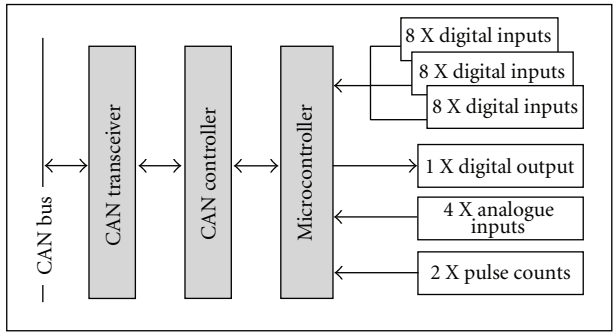


Figure 6: Monitoring module hardware block diagram.

"ending" records or triggering the acquisition of analogue signals). This is different to transitions, which are defined individually within an essentially static structure but can still be used to provide information relating to that structure. The nature of the process related information thus made available is shown in Table 2.

The hardware architecture supporting this implementation was fully detailed in a previous paper [13]. The Petri-net structure was implemented into a microcontroller-based monitoring module (MM). This was provided with data acquisition, communication, and processing capabilities. A block diagram illustrating the main MM components is shown in Figure 6. In order to provide the necessary flexibility and data integration capabilities, communication assumed a great importance at different levels within the system so in this case each MM is able to implement a CAN bus node. Such a serial link reduces considerably the hardware design and consequently cost, although increasing software engineering complexity. Both, the microcontroller and CAN controller, share a single 20 MHz oscillator. A CAN transceiver was employed to physically interface the CAN bus.

Monitoring modules were provided with 3 different sources of signal input: digital, analogue, and pulse. In order to combine simple hardware design and improved system capabilities, each MM allows up to 3 digital cards (with 8 inputs each) to be attached, all sharing one of the microcontroller's input ports. Each MM is therefore able to interface with up to 24 digital inputs. A 3 bit port was used to implement the card selection logic the operation of which was implemented in software. Each digital input was provided with an optocoupler, in order to interface to the process signals and ensure equipment protection.

Four analogue inputs were implemented to support special monitoring purposes. The hardware design supported the use of transducers with an output range of 0 to 10 volts. No further conditioning or signal filtering method was employed. Two-pulse inputs were connected to two of the microcontroller's port B pins and configured as external interrupts. Each pulse input was again interfaced using an optocoupler. In order to support the "transition output" implementation one of the microcontroller's pins (port A, bit 6) was configured as an output. Further interfacing to adapt the electrical levels or latching mechanisms to an external alarm-signalling device was provided, depending on the application specifics.

The MM software development to support the implementation of the Petri-net monitoring approach was undertaken within the microcontroller's development environment. Although mainly concerned in executing the process Petri-net, a number of other tasks, such as data acquisition, communication, and timing were required. The flow diagram shown in Figure 7 illustrates the main software tasks. The microcontroller's interrupt capability was explored in order to reduce software complexity and increase efficiency. The initialisation process sets variables, buffers, and configures hardware devices. The microcontroller's memory was divided between the system's variables and stack pointer, communication buffers, and the Petri-net implementation area as detailed in Table 3. Interrupts were deployed to synchronise the serial peripheral interface (SPI) interface data transmission/reception. A similar technique was employed in order to enable the CAN controller to notify CAN related events, such as transmission/reception and error indications.

Each MM was configured to support 254 transitions, 254 places, and 255 subnet IDs. The implementation considered the application's Petri-net description to be based on three attached text formatted files. The first of such files represents the Petri-net main structure (event descriptions), MM identification, and the identification of the places required to provide time-out events.

The second file identifies the Petri-net's places representing the process states required to have their active status watched (and reported). This was defined as a 32-byte structure, in which 1 bit is used to represent each possible place. The last attached file correlates places with analogue or pulse inputs. It contains a table with 254 inputs, sequentially representing the Petri-net's places and indicating an input source, analogue or pulse ("0" meaning no source associated), used to trigger the data acquisition of process specific parameters.

Places were defined in the microcontroller's data memory as a continuous set of 256 bytes (only 254 effectively used). The place identification indexes the place location within this data structure. Each place location will hold its respective number of tokens, being updated by the Petri-net execution. The MM makes public a subnet event by broadcasting its identification (subnet ID). Internally, subnets will be assigned in a bit-mapped data structure (32 bits), with one bit representing each subnet ID. The Petri-net execution, when required, searches for subnet events and updates this bit-mapped structure.

One of the microcontroller's timers (TMR0) was configured to generate a 1 ms time base that was used to update the monitoring module date/time record. The microcontroller's interrupt functionality was employed in order to generate a precise and reliable timing method. The same time base was used in other tasks that require time measurement. Digital input updating was synchronised with the 1 ms time base generated by TMR0. Since there are a possible 3 digital cards, all using the same microcontroller's port (D), switching time had to be considered. The employed procedure reads the 8 digital inputs of the selected card and then identifies and

Table 2: Monitoring messages definitions.

Monitoring record	Purpose description
Process event	Message issued in response to a process event (fired transition). Only *ordinary* and *analogue* transitions can issue this sort of message.
Beginning of a process state	Enabled places issue such message when receiving the *first* token, indicating the "beginning" of the associated process state (state became active).
Ending of a process state	Enabled places issue such message when becoming *empty* (last token removed), indicating the "ending" of the associated process state (became inactive).
Process state timeout	Messages issued by selected places to indicate that a process *state has lasted longer* than expected.
Special record	Message issued at the end of a selected process state containing a record with a *feature* extracted from a process analogue signal. The signal is acquired as long as the process state remains active (e.g., a DC motor current *mean value*, to indicate the motor's operating condition).

Table 3: Monitoring module data memory distribution.

Buffer description	Buffer size (bytes)
System variable	256
Software stack	32
SPI transmit buffer	96
SPI receive buffer	32
Transmit message buffer (built)	16
Receive message buffer (rebuilt)	16
Event record buffer	64
Petri-net places buffer	256
Petri-net places timeout control buffer	512
Petri-net subnet buffer	32
Timeout devices buffer	150
Active places mapping buffer	32

moves on to the following one, which will be read in the next acquisition cycle, thus ensuring enough time to make the bus stable when the next update is carried out. By using such an approach, digital inputs are updated every 3 ms (333.33 updates/second).

The pulse inputs were configured to automatically generate interrupts whenever such an event is matched. Counters (one for each input) were incremented during the interrupt service. The counters are read and reset by the system's application every second providing a monitoring parameter in terms of pulses/second (P_m). Such a parameter might be further used by an analogue transition and as the basis for a special record as defined in Table 2. Analogue inputs were configured for 10 bit resolution. They are updated periodically, by polling the A/D converter in the system's application execution main loop shown in Figure 7. Based on the microcontroller's analogue channels specifications and software implementations, the sampling rate is approximately 2.5 K sample/s. Small variations occur due to different tasks being performed in different execution loops. The sampled analogue input data can be integrated over a period equivalent to 256 samples, resulting in an average value that may be used as an input parameter into an analogue transition.

Following the approach described above Petri-net places can be used to trigger the acquisition of process specific parameters, resulting in a "special record." In this implementation, such a parameter was defined as the mean value of the observed analogue or pulse inputs, based on observation time. Such values will be continuously added, as long as the process state remains active.

The monitoring Petri-net will run by executing the transitions defined in the Petri-net table (text format file). The data retrieved from this table is verified in the transition structure context, thus checking whether or not the transition is enabled. Transition firing actions include updating input and output places, requesting a subnet broadcast and event messaging. Transitions execute sequentially in the order in which they are defined in the Petri-net table, through handling one transition each time, per cycle. For those fired transitions that require a message, a record will be stored in the "event record buffer." Whenever places are updated (following a Petri-net execution), further verification is carried out to identify the "beginning" or "end" of an active state of selected places or to trigger the "mean value calculation" of an analogue or pulse input. When required, a message transmission will be requested by inserting a record in the application event record buffer.

Places enabled to have a time-out control and whose active state lasted longer than has been previously recorded will produce a time-out record. The default approach compares the actual cycle with previous one. However, such comparison parameters can be fixed by supplying a command, received through the data communication interface (CAN). Using this property fault isolation was implemented by identifying the place that provided the symptom (timeout) through the data communication interface. The Petri-net can be searched in order to find the transition(s) having the provided place as an input. In this way, verification can be made to detect whether the fault relates to the transition input signals or to other places that failed to enable this transition. Successive iterations will be carried out until a result is obtained or the entire set of transitions is investigated. Loops are avoided by marking places that have already been verified. Records are then placed in the "event record buffer," identifying the transition and the signal(s) that failed to enable the transition. Subnet, digital and analogue/pulse inputs are considered as possible sources of faults.

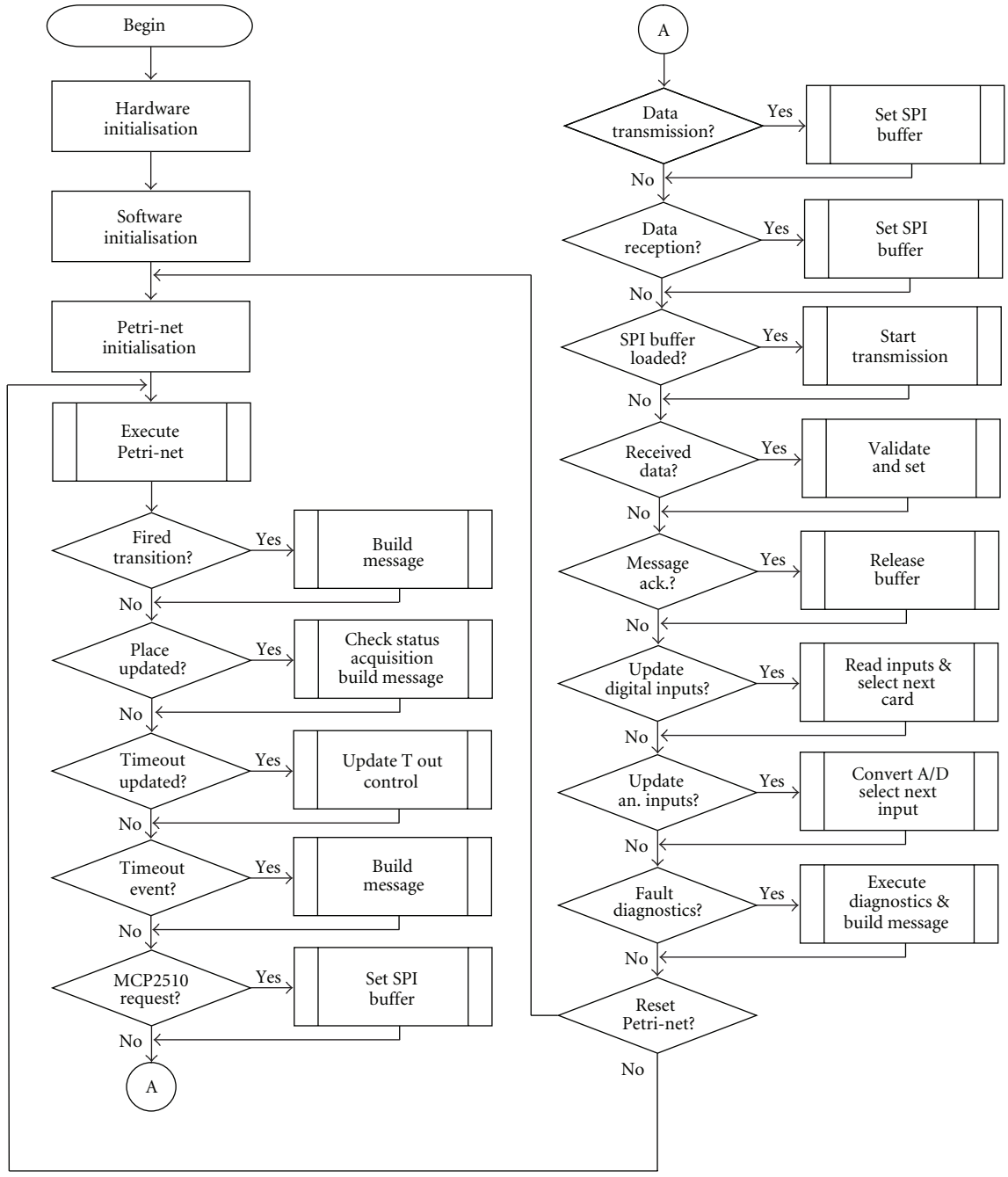


Figure 7: Monitoring module application flow diagram.

Finally, to support the implementation of this approach to real applications, data communication aspects were considered. In order to improve system's efficiency and ease software development, data buffers were set to hold a number of monitoring events recorded by the system. Each of these records is converted into a system's message, then assembled and stored in the transmit message buffer. This buffer is capable of handling only one message each time. At the next stage, the system's messages are handled by the "CAN application layer," before being stored in the "SPI transmit buffer." At this application layer, messages will be sized accordingly to the CAN protocol, and split when required. A sequencing method was developed, to enabled messages to be reassembled at the destination end. A transmission time-out feature was implemented to control message delivery. Transmitted messages that were not acknowledged by the recipient in a predefined time will be retransmitted. The CAN controller's commands are required to be appended to the application layer message, in order to properly set up the required task. Messages received from the CAN bus, such as subnet broadcast and system commands, will be transferred to and reassembled in the receive message buffer.

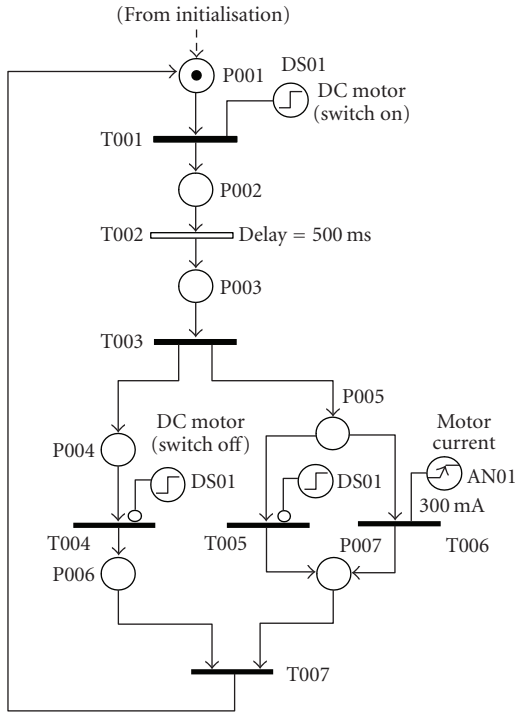


Figure 8: Petri-net monitoring approach example.

Once validated, commands will be executed or will result in flags requesting further actions (i.e., fault diagnostics).

The monitoring module application required 8,426 bytes of program memory. Considering that the data tables representing the process states to be monitored and those that trigger analogue/pulse acquisition require a fixed amount of 288 bytes (32 + 256), the representation of a process Petri-net can access 24,054 bytes. Assuming as a general example an ordinary transition with 2 input and 2 output places, a Petri-net with a maximum of 254 of such transitions would require 5,588 bytes of program memory, which is much less than the typical total available. If needs be the architecture can integrate any number of MMs to allow for more complicated monitoring tasks.

5.1. Example Application. The Petri-net, shown in Figure 8, represents a Subnet designed to monitor the operation of a DC motor. It is part of a larger monitoring task deployed in the IPMM Centre to monitor a scale model of a hydroforming press process.

The net is enabled to follow the motor operation by receiving a token from the system's initialisation. This token is placed in P001. The motor command is detected by the digital signal DS01 switching on. Place P001 and DS01 represent the condition required to fire T001, therefore indicating this event and resulting in a token in P002. Transition T002 is a "delay transition," introduced to retain the token for 500 ms. This is for practical reasons, avoiding any further monitoring system interpretation before the signal has stabilised (e.g., switching current peaks). After this delay the token follows through P003/T003, reaching places P004 and P005. Here a special feature is modeled. Place P004 holds its' token as long as the motor runs. The end of this state is detected by DS01 switching off (a condition indicated by a small circle at the sensor symbol connection near to the transition bar), therefore firing T004 and introducing a token into P006. Transition T006 was placed in the diagram to monitor an over-current condition while the motor is operated. This is achieved by employing an "analogue transition" element (T006), which has as input conditions P005 and "analogue signal 1," the motor current, with threshold defined as "above to" 300 mA. Transition T005 is required in order to remove the token from P005 in case the over-current event was not matched (normal condition) before the end of the "motor operating" state. After this, places P006 and P007 will enable transition T007, reestablishing the Petri-net initial marking with a token in P001 and consequently enabling the next operating cycle to be monitored.

In making an extended use of the Petri-net places as defined in this approach, additional information could be retrieved from the monitoring task. For example, by enabling P004 to issue "beginning state" and "ending state" records, it becomes possible to monitor how long the motor has been in use. In monitoring terms an increase in the time to perform such a cycle would potentially help in assessing the degradation of the entire moving system. A severe increase in the cycle time would be detected by a time-out record associated to this same place (P004). Furthermore, place P004 might be used to trigger the acquisition of an analogue channel, for instance to extract additional motor current information, apart from the over-current record provide by means of T006. This approach allows critical signals to be monitored during the enactment of an operation and offers the ability of capturing any unusual variations whilst allowing "normal" signals to be assessed, summarised, and discarded.

The intelligent application of this feature means that all deviations from normal can be captured at source without the need for the continuous streaming of data. In the system implementation employed in this investigation this information relates to the calculation of the mean value of the current required by the DC motor during the operating cycle. The actual signal can be captured, analysed, and if indicated retained for future interrogation. Thus, if a sudden overload appears due a "hard" fault such as a trapped workpiece or faulty cycle, the signal can be retained, even if the process ultimately continues. If needs be any unrecognised signal variation may be referred to more sophisticated condition monitoring software off-line, making the diagnosis of "soft" faults possible. These parameters enable operatives to assess the condition of the process and specific devices during the time of any fault like occurrences and therefore enable maintenance interventions based on real knowledge.

6. Conclusions

Any sequential process can be modelled in terms of a Petri-net, based on its states and the events that characterise the transitions between them. To facilitate the engineering of

a Petri-net process monitoring tool several extensions to conventional Petri-net representation have been specified and developed in order to interface and handle real-life process signals. The most important of these is the use of analogue signals aligned with the powerful microcontroller-based analysis tools now available. These allow enhanced monitoring functions, including data capture for subsequent analysis by more powerful systems. This is critical to the subsequent evolution of low-cost monitoring systems.

In defining a Petri-net model that describes each event as a self-contained data structure, a method has been proposed which, although being primarily developed to exploit the ever increasing potential of microcontroller devices, has no hardware dependency. The Petri-net functionality was further extended by enabling elements to be deployed to trigger the acquisition of processes' specific parameters to monitor the "beginning" and "ending" of processes active states. This extended functionality was supported by a set of messages that enable database records to be produced and supported fault diagnosis by comparing actual measurements to those previously established as representing "normal" behaviour..

This research has provided the framework for the continued development and enhancement of the tools required for the implementation of a monitoring system. The benefits of deploying the monitoring system on a microcontroller relate to the practicality of embedding such devices within the process being monitored as part of a layered architecture that, supported by the Petri-net approach, enables the development of a distributed structure which offers great potential in the area of low-cost process monitoring systems.

References

[1] J. L. Peterson, *Petri-net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[2] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat, *Practice of Petri-nets in Manufacturing*, Chapman & Hall/CRC, London, UK, 1993.

[3] C. Girault and R. Valk, *Petri-nets for Systems Engineering*, Springer, Berlin, Germany, 2003.

[4] R. David and H. Alla, *Discrete, Continuous and Hybrid Petri-nets*, Springer, Berlin, Germany, 2005.

[5] L. Jiao, "A note on regular Petri-nets," *Information Processing Letters*, vol. 108, no. 3, pp. 110–114, 2008.

[6] H. Alla and R. David, "A modelling and analysis tool for discrete events systems: continuous Petri-net," *Performance Evaluation*, vol. 33, no. 3, pp. 175–199, 1998.

[7] A. Zimmermann and G. Hommel, "Modelling and evaluation of manufacturing systems using dedicated Petri-nets," *International Journal of Advanced Manufacturing Technology*, vol. 15, no. 2, pp. 132–138, 1999.

[8] M. Dotoli, M. P. Fanti, A. Giua, and C. Seatzu, "First-order hybrid Petri-nets. An application to distributed manufacturing systems," *Nonlinear Analysis: Hybrid Systems*, vol. 2, no. 2, pp. 408–430, 2008.

[9] S. K. Yang and T. S. Liu, "A Petri-net approach to early failure detection and isolation for preventive maintenance," *Quality and Reliability Engineering International*, vol. 14, no. 5, pp. 319–330, 1998.

[10] P. Prickett, "A Petri-net based machine tool maintenance management system," *Industrial Management and Data Systems*, vol. 97, no. 4, pp. 143–149, 1997.

[11] G. Frey and M. Minas, "Internet-based development of logic controllers using signal interpreted Petri-nets and IEC 61131," in *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI '01)*, pp. 297–302, Orlando, Fla, USA, July 2001.

[12] S. Peng and M. Zhou, "Sensor-based stage Petri-net modelling of PLC logic programs for discrete-event control design," *International Journal of Production Research*, vol. 41, no. 3, pp. 629–644, 2003.

[13] K. J. Hintz, "Microcontroller software design using Petri tables," *Journal of Microcomputer Applications*, vol. 15, no. 4, pp. 313–325, 1992.

[14] K. J. Hintz and D. Tabak, *Microcontrollers: Architecture, Implementation, and Programming*, McGraw-Hill, New York, NY, USA, 1992.

[15] M. R. Frankowiak, R. I. Grosvenor, and P. W. Prickett, "A Petri-net based distributed monitoring system using PIC microcontrollers," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 189–196, 2005.

[16] P. Prickett and R. Grosvenor, "A Petri-net-based machine tool failure diagnosis system," *Journal of Quality in Maintenance Engineering*, vol. 1, no. 3, pp. 47–57, 1995.

[17] M. Bolic, V. Drndarevic, and B. Samardzic, "Distributed measurement and control system based on microcontrollers with automatic program generation," *Sensors and Actuators A*, vol. 90, no. 3, pp. 215–221, 2001.

[18] E.-J. Manders, L. A. Barford, and G. Biswas, "An approach for fault detection and isolation in dynamic systems from distributed measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 2, pp. 235–240, 2002.

[19] D. K. Baek, T. J. Ko, and H. S. Kim, "Real time monitoring of tool breakage in a milling operation using a digital signal processor," *Journal of Materials Processing Technology*, vol. 100, no. 1–3, pp. 266–272, 2000.

[20] W. Amer, R. I. Grosvenor, and P. W. Prickett, "Sweeping filters and tooth rotation energy estimation (TREE) techniques for machine tool condition monitoring," *International Journal of Machine Tools and Manufacture*, vol. 46, no. 9, pp. 1045–1052, 2006.

[21] R. A. Siddiqui, W. Amer, Q. Ahsan, R. I. Grosvenor, and P. W. Prickett, "Multi-band infinite impulse response filtering using microcontrollers for e-Monitoring applications," *Microprocessors and Microsystems*, vol. 31, no. 6, pp. 370–380, 2007.