HURRAY-TR-9908
Date: 22-03-99

Engineering Real-Time Applications with WorldFIP: Tools and Analysis

Eduardo Tovar[1], Francisco Vasques[2]

[1] Department of Computer Engineering, Polytechnic Institute of Porto, Rua de São Tomé, 4200 Porto, Portugal; Tel.: +351.2.8340500; Fax: +351.2.8321159
emt@dei.isep.ipp.pt
[2] Department of Mechanical Engineering, University of Porto, Rua dos Bragas, 4099 Porto Codex, Portugal
vasques@fe.up.pt

Abstract. WorldFIP is standardised as European Norm EN 50170 - General Purpose Field Communication System. Field communication systems (fieldbuses) started to be widely used as the communication support for distributed computer-controlled systems (DCCS), and are being used in all sorts of process control and manufacturing applications within different types of industries. There are several advantages in using fieldbuses as a replacement of for the traditional point-to-point links between sensors/actuators and computer-based control systems. Indeed they concern economical ones (cable savings) but, importantly, fieldbuses allow an increased decentralisation and distribution of the processing power over the field. Typically DCCS have real-time requirements that must be fulfilled. By this, we mean that process data must be transferred between network computing nodes within a maximum admissible time span. WorldFIP has very interesting mechanisms to schedule data transfers. It explicit distinguishes to types of traffic: periodic and aperiodic. In this paper we describe how WorldFIP handles these two types of traffic, and more importantly, we provide a comprehensive analysis for guaranteeing the real-time requirements of both types of traffic. A major contribution is made in the analysis of worst-case response time of aperiodic transfer requests.

1. Introduction

Local area networks (LANs) are becoming increasingly popular in industrial computer--controlled systems. LANs allow field devices like sensors, actuators and controllers to be interconnected at low cost, using less wiring and requiring less maintenance than point-to--point connections [1]. Besides the economical aspects, the use of LANs is also reinforced by the increasing decentralisation of control and measurement tasks, as well as by the increased use of intelligent microprocessor-controlled devices.

Broadcast LANs aimed at the interconnection of sensors, actuators and controllers are commonly known as fieldbus networks. In the past, the fieldbus scope was dominated by vendor specific solutions, which were mostly restricted to specific application areas. Moreover, concepts behind each proposed network were highly dependent on the manufacturer of the automation system, each one with different technical implementations and also claiming to fulfil different application requirements, or the same requirements with different technical solutions [2]. More recently, standardised fieldbuses supporting the open system concept, thus vendor independent, started to be commonly used. Particular relevance

must be given to the European Standard EN 50170 [3], which encompasses three widely used fieldbuses: P-NET [4], PROFIBUS [5] and WorldFIP [6].

In this paper we address the ability of WordFIP to cope with the real-time requirements of distributed computer-controlled systems (DCCS). In essence, by timing requirements we mean that traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur.

Typically, DCCS include process variables that must be transferred between network devices both in a periodic basis and also in a sporadic (aperiodic) basis. WorldFIP protocol is designed to support both types of traffic. As it will be seen, timeliness requirements for the periodic traffic can be easily guaranteed by WordlFIP, however for the aperiodic traffic more complex analysis must be made *a priori*, in order to respect their timing requirements.

The remaining of the paper is organised as follows. In section 2 we describe the main characteristics behind the WorldFIP protocol. Mechanisms provided by WorldFIP to support both the periodic and aperiodic traffic are described in detail. Not much analysis has been devoted to the issue of setting the WorldFIP bus arbitrator table (BAT) such as it complies with the timing requirements of the periodic traffic. Therefore, we devote section 3 to provide an approach on how to schedule the periodic traffic in WorldFIP based DCCS. Based on this approach, in section 4 we evaluate the worst-case response time for the aperiodic traffic, highlighting the important improvements provided by our analysis as compared to the previously available from other related works [7, 8].

2. Concepts Behind WorldFIP

A WorldFIP network interconnects stations with two types of functionalities: bus arbitration and production/consumption functions. At any given instant, only one station can perform the function of active bus arbitration. Hence, in WorldFIP, the medium access control (MAC) is centralised, and performed the active bus arbitrator (BA).

WorldFIP supports two basic types of transmission services: *exchanges of identified variables* and *exchanges of messages*. In this paper we address WorldFIP networks supporting only exchanges of identified variables, since they are the basis of WorldFIP real-time services. The exchange of messages is used to support manufacturing message services (MMS) [9], which are out of the scope of this paper.

2.1. Concept of Producer-Distributor-Consumer

In WorldFIP, the exchange of identified variables services are based on a producer/ distributor/consumer (PDC) model, which relates producers and consumers within the distributed system. In this model, for each process variable there is one, and only one producer, and several consumers. For instance, consider the variable associated with a process sensor. The station that provides the variable value will act as the variable producer and its value will be provided to all the consumers of the variable (e.g., the station that acts as process controller for that process variable or the station that is responsible for building an historical data base).

In order to manage transactions associated with a single variable, a unique identifier is associated with each variable. The WorldFIP data link layer (DLL)[1] is made up of a set of

[1] WorldFIP protocol is based on a three layered architecture: physical layer, data link layer and application layer.

produced and consumed buffers, which can be locally accessed (through application layer (AL) services) or remotely accessed (through network services).

The AL provides two basic services to access the DLL buffers: *L_PUT.req*, to write a value in a local produced buffer, and *L_GET.req* to obtain a value from the local consumed buffer. None of these services generate activity on the bus.

Produced and consumed buffers can be also remotely accessed through a network transfer (service also known as *buffer transfer*). The bus arbitrator broadcasts a question frame *ID_DAT*, which includes the identifier of a specific variable. The DLL of the station that has the corresponding produced buffer responds with the value of the variable using a response frame *RP_DAT*. The DLL of the station that contains the produced buffer then sends an indication of transmission of the value to the AL (*L_SENT.ind*). The DLL of the station(s) that has the consumed buffers accepts the value contained in the *RP_DAT*, overwriting the previous value and notifying the local AL with a *L_RECEIVED.ind*.

Figure 1 illustrates the case of a station with one produced buffer (for identifier *k*) and one consumed buffer (for identifier *x*). The first one can be locally written trough a *L_PUT.req* (to overwrite a new value) or through a buffer transfer, in which its value is made available to other stations. The second one can be either locally read, through a *L_GET.req* service or remotely written through a buffer transfer, being its value overwritten with the new value transferred from a remotely produced buffer

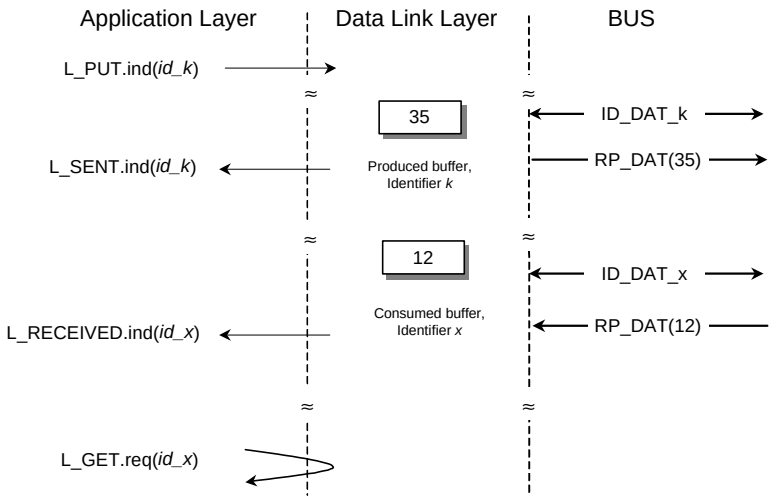
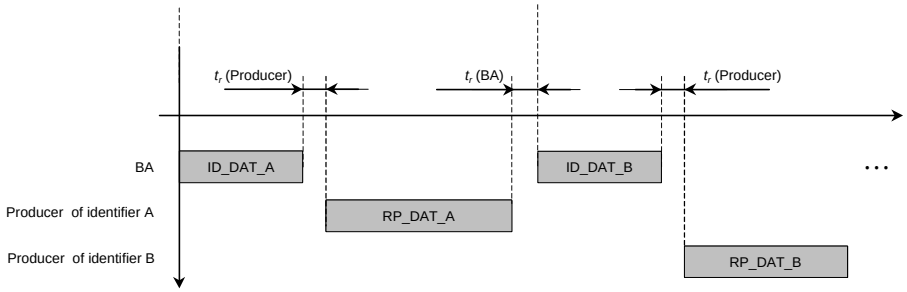


Figure 1

2.2. Buffer Transfer Timings

A buffer transfer implies the transmission of a pair of frames: *ID_DAT*, followed by a *RP_DAT*. We denote this sequence as an *elementary transaction*. The duration of this transaction equals the time needed to transmit the *ID_DAT* frame, plus the time needed to transmit the *RP_DAT* frame, plus twice the turnaround time (t_r). The turnaround time is the

time elapsed between any two consecutive frames. Figure 2 illustrates the concept of an elementary transaction in WorldFIP.



2.3. Bus Arbitrator Table

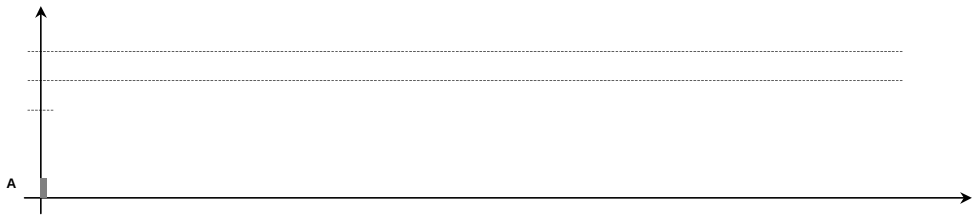
In WorldFIP networks, the *bus arbitrator table* (BAT) regulates the scheduling of all buffer transfers. In practice, two types of buffer transfers can be considered: periodic and aperiodic (sporadic). The BAT imposes the timings of the periodic buffer transfers, and also regulates the aperiodic buffer transfers, as it will be later explained in sub-section 2.4.

Assume a distributed system in which 6 variables are to be scanned periodically, with the scan frequencies as shown in table 1. The WorldFIP BAT must be set in order to cope with these timing requirements.

Table 1: Example Set of Periodic Buffer Transfers

Identifier	**A**	**B**	**C**	**D**	**E**	**F**
Periodicity (ms)	1	2	3	4	4	6

Two important parameters are associated with a WorldFIP BAT: the *microcycle* (elementary cycle) and the *macrocycle*. The microcycle imposes the maximum rate at which the BA performs a set of scans. Usually, the microcycle is set equal to *highest common factor* (HCF) of the required scan periodicities. Using this rule, and for the example shown in table 1, the value for the microcycle is set to 1ms. A possible schedule for all the periodic scans can be as illustrated in figure 4.



```
    <microcycle 3>: A, B;
    <microcycle 4>: A, C;
    <microcycle 5>: A, B, D, E;
    <microcycle 6>: A;
    <microcycle 7>: A, B, C, F;
    <microcycle 8>: A;
    <microcycle 9>: A, B, D, E;
    <microcycle 10>: A, C;
    <microcycle 11>: A, B;
    <microcycle 12>: A;
----------------------------------------
```

The HCF/LCM approach for building a WorldFIP BAT has the following properties:

1. The scanning periods of the variables are multiples of the microcycle;
2. The variables are not scanned at exactly regular intervals. For the given example, only variables *A* and *B* are scanned exactly in the same "slot"[3] within a microcycle. All other variables suffer from a slight communication jitter. For instance, concerning variable *F*, the interval between microcycles 1 and 7 is (1-5×0.0976)+5+(3×0.0976)= =5.8048ms, whereas the interval between microcycles 7 and 13 is (1-3×0.0976)+5+ +(5×0.0976)=6.1952ms.
3. The length of the macrocycle can induce a memory size problem, since the table parameters must be stored in the BA. For instance, if the scanning periodicities of variables *E* and *F* were, respectively, 5ms and 7ms, the length of the macrocycle would be 420 microcycles instead of only 12.

Both the communication jitter and memory size problems have been addressed in the literature. In [11], the authors discuss different methodologies for reducing the BAT size, without worstening the communication jitter problem. The idea is very simple, and basically consists on reducing some of the scan periodicities in order to make them harmonic. The problem of table size has also been addressed in [12,13], however, in a different perspective, since the authors discuss an online scheduler (instead of storing the schedule in memory), hence not directly addressing the WorldFIP case.

It is also worth mentioning that the schedule shown in figure 4 represents a macrocycle composed of synchronous microcycles, that is, for the specific example, each microcycle starts exactly 1ms after the previous one. Within a microcycle, the spare time between the end of the last scan for a periodic variable and the end of the microcycle can be used by the BA to process aperiodic requests for buffer transfers (see sub-section 2.4), message transfers and padding identifiers[4]. A WorldFIP BA can also manage asynchronous microcycles, not transmit padding identifiers at the end of the microcycle. In such case, a new microcycle starts as soon as the periodic traffic is performed and there are no pending aperiodic buffer transfers or pending message transfers. Initial periodicities are not respected, since identifiers may be more frequently scanned.

2.4. Aperiodic Buffer Transfers

In a WorldFIP system, not all identified variables are included in the BAT. Some variables may only be occasionally exchanged, and thus do not need to be periodically scanned. Typically these exchanges will concern application events or alarms, which by their own

[3] For simplification of the analysis, we are assuming always the same length for each elementary transaction.

[4] If after the periodic traffic, the sporadic traffic (if any) and message transfers (if any) there is still time within the microcycle, the BA transmits padding identifiers (not produced by any station), to indicate the other stations that it is still functioning.

nature do not occur with a periodic pattern. Therefore, it is preferable to map these variables into aperiodic buffer transfers. In this way network is not unnecessary overload.

The BA handles aperiodic buffer transfers only after processing the periodic traffic in a microcycle. The portion of the microcycle reserved for the periodic buffer exchanges is denoted as the *periodic window* of the microcycle. The time left after the periodic window is completed until the end of the microcycle is denoted as the *aperiodic window* of the microcycle. The aperiodic buffer transfers take place in three stages:

1. When processing the BAT schedule, the BA broadcasts an *ID_DAT* frame concerning a periodic variable, say identifier *X*. The producer of *X* responds with a *RP_DAT* and sets an aperiodic request bit in the control field of its response frame. The bus arbitrator stores variable *X* in a queue of requests for variable transfers[5].
2. In the aperiodic window the BA uses an identification request frame (*ID_RQ*) to ask the producer of the identifier *X* to transmit its list of pending aperiodic requests. The producer of *X* responds with a *RP_RQ* frame (list of identifiers). This list of identifiers is placed in another BA's queue, the *ongoing aperiodic queue*.
3. Finally, the BA processes requests for aperiodic transfers that are stored in the ongoing aperiodic queue. For each transfer, the BA uses the same mechanism as the used for the periodic buffer transfers (*ID_DAT* followed by a *RP_DAT* of the producer of the aperiodic variable).

A station that requests an aperiodic transfer can be: the producer of the variable; the consumer of the variable; both producer and consumer; neither producer nor consumer (third-party variables). It is however important to note that a station can only request aperiodic transfers using responses to periodic variables that it produces and which are configured in the BAT.

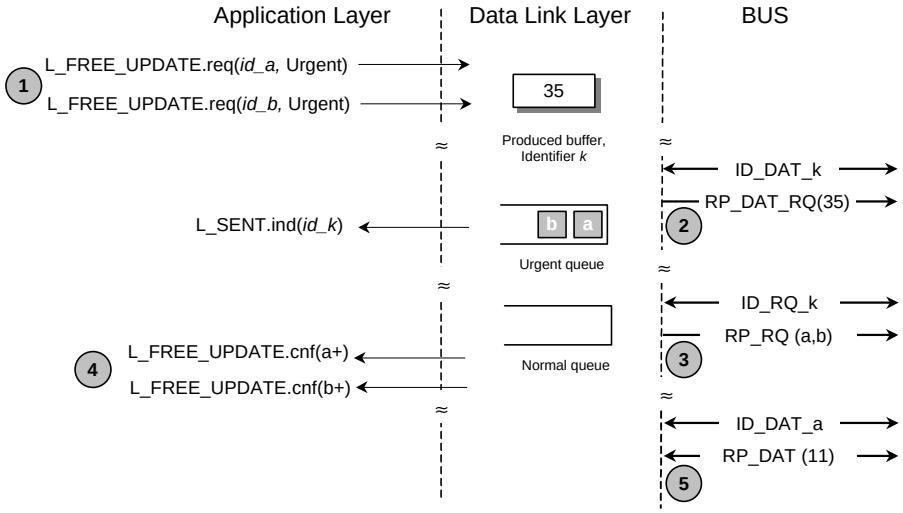


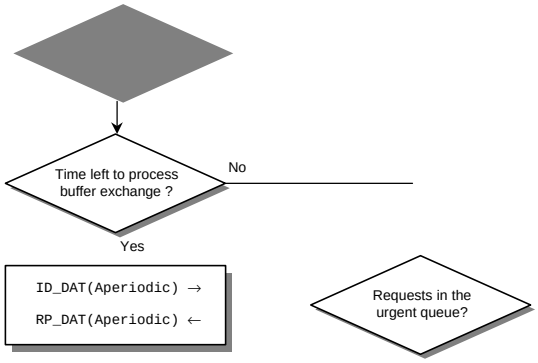
Figure 5

As previously mentioned, in this paper we only consider buffer transfers (identified variables exchanges), both periodic and aperiodic requests. Concerning the aperiodic

[5] Two priority levels can be set when the request for aperiodic transfer is made: Urgent or Normal. The BA has two queues, one for each priority level.

requests, we consider that all these requests concern the use of the AL service *L_FREE_UPDATE.req(, Urgent)*, thus considering only the urgent queues (both local and at the BA).

Finally, it is important to stress that the urgent queue in the BA is only processed if, and only if, the BA's ongoing aperiodic queue is empty, as detailed in figure 6.



3.1. Aperiodic Buffer Transfers

Assume a system with *np* periodic variables (Vp_i, i = 1, .., *np*). Each periodic variable Vp_i is characterised as:

$$Vp_i = (Tp_i, Cp_i) \quad \textbf{(2)}$$

where Tp_i corresponds to the periodicity of Vp_i (assume a multiple of 1ms) and Cp_i is the length of the transaction corresponding the buffer transfer of Vp_i (as given by equation (1)).

3.2. Tools for Setting the Bus Arbitrator Table

Following the HCF/LCM methodology to build the BAT, the value for the microcycle (μCy) is chosen as:

$$\mu Cy = \underset{i=1,..,np}{HCF}(Tp_i) \quad \textbf{(3)}$$

where HCF stands for the highest common factor and corresponds to the following value:

$$\mu Cy = \max\{\Omega\} \wedge \Omega \in \aleph, \quad \text{with } \frac{Tp_i}{\Omega} = \left\lfloor \frac{Tp_i}{\Omega} \right\rfloor, \forall_i \quad \textbf{(4)}$$

In appendix A.1, an algorithm for the evaluation of the microcycle value is suggested.

The macrocycle (*MCy*) is defined as:

$$MCy = N \times \mu Cy \quad \textbf{(5)}$$

where *N* is the number of microcycles that compose a macrocycle. Using the LCM rule, *N* is evaluated as follows:

$$N = \min\{\Phi\} \wedge \Phi \in \aleph, \quad \text{with} \frac{\Phi}{Tp_i / \mu Cy} = \left\lfloor \frac{\Phi}{Tp_i / \mu Cy} \right\rfloor, \forall_i \quad \textbf{(6)}$$

In appendix A.2, an algorithm for the evaluation of the macrocycle value is suggested.

The BAT can easily be built considering the rate monotonic (RM) algorithm [14], adapted for the WorldFIP BAT characteristics, and using the following rules:

1. From variable with the shortest period until variable with the longest period
   1.1. If the load in each cycle plus the *Cp* of the variable is still shorter than the value of the microcycle, then schedule scans for that variable in each of the microcycles (of a macrocycle) multiple of the period of the variable. Update the value of the load in each concerned microcycle.
   1.2. If the load in some of the microcycles does not allow to schedule a scan for the variable, try to schedule it for the first of the subsequent microcycles up to the microcycle in which a new scan for that variable should be made. If this is not possible, the variable set is not scheduled.

For the example of table 1, in which all Cp_i = 0.0976ms, $\forall_i$, the BAT, using the RM algorithm, is:

Table 2: BAT (using RM) for Example of Table 1

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	1	0	0	0	0	0	1	0	0	0	0	0

where *bat*[*i*, *j*] is a table of booleans with *i* ranging from 1 to *np*, and *j* ranging from 1 to *N* (number of microcycles in a macrocycle).

In appendix A.3, a detailed algorithm for building the BAT using the rate monotonic methodology previously described is presented. In the alsgorithm, the vector *load*[ ] is used to store the load in each microcycle as the traffic is schedule. It also assumes that the array *Vp*[,] is ordered from the variable with the shortest period (*Vp*[1,]) to the variable with the longest period (*Vp*[*np*,]).

Note that by using this RM algorithm some of the variables with longer periods can be scheduled for subsequent microcycles, thus inducing a greater communication jitter for those variables. For example, if the network data rate is 1Mbps instead of 2.5Mbps (each Cp_i= = (64+80)/1+2×20=184μs), the BAT would not be as shown in table 2 but that as shown in table 3, which corresponds to the schedule shown in figure 7a).

Table 3: BAT (using RM) for Modified Example of Table 1

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	0	1	0	0	0	0	1	0	0	0	0	0

Other algorithms, rather than the RM algorithm, could be implemented [15] in order to have a more uniform number of periodic scans in each microcycle (using the same assumptions for determining the value for both the microcycle and the macrocycle). Figure 7b) illustrates a schedule where the scan pattern for some of the variables is shifted right a nm_i number of microcycles, with $nm_i \leq (Tp_i/\mu Cy)$. The advantage is the reduction of Jitter. Table 4 gives the BAT for this other algorithm.

Table 4: BAT with Modified Algorithm

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	0	1	0	0	0	1	0	0	0	1	0	0
bat[*E*,*cycle*]	0	0	0	1	0	0	0	1	0	0	0	1
bat[*F*,*cycle*]	0	1	0	0	0	0	0	1	0	0	0	0

which means that $nm_A = nm_B = nm_C = 0$, $nm_D = nm_F = 1$ and finally $nm_E = 3$.

$$NR_i = \left\lceil \frac{Cp + I_i}{\mu Cy} \right\rceil \tag{7}$$

with

$$I_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{NR_i \times \mu Cy}{Tp_j} \right\rceil \times Cp \right) \tag{8}$$

Equation (7) reflects the fact that in each microcycle only a number of transactions that completely fit in it are processed[6]. Assume for example that μCy=1ms and Cp=0.21ms. To process 5 buffer transfers 2 microcycles are needed (4 buffer transfers will be processed in the first microcycle and 1 buffer transfer in the second microcycle, as $\lceil (5 \times 0.21)/1 \rceil = \lceil 1.05 \rceil = 2$).

Equation (8) gives the load (as a multiple of Cp) corresponding to the number of buffer transfers to be scheduled ahead of Vp_i in NR number of microcycles. Note that with the rate monotonic algorithm, variables with shorter periods (higher priority) are scheduled ahead of variables with longer periods (lower priority). Therefore, if a lower priority variable is scheduled to a subsequent microcycle (no time left in that microcycle), its corresponding buffer transfer will only be performed after higher priority variables in that subsequent microcycle.

Combining (8) and (7), the feasibility test can be written as:

$$NR_i = \left\lceil \frac{Cp + \sum_{j \in hp(i)} \left(\left\lceil \frac{NR_i \times \mu Cy}{Tp_j} \right\rceil \times Cp \right)}{\mu Cy} \right\rceil \leq \frac{Tp_i}{\mu Cy}, \ \forall_i \tag{9}$$

that is, the maximum number of microcycles needed to transfer a variable must be smaller than the number of microcycles till the next "request" for a transfer of that variable.

Equation (9) embodies a mutual dependence, since NR_i appears in both sides of the equation. The easiest way to solve equation (9) is to form a recurrence relationship [18]:

$$W_i^{m+1} = \left\lceil \frac{Cp + \sum_{j \in hp(i)} \left(\left\lceil \frac{W_i^m \times \mu Cy}{Tp_j} \right\rceil \times Cp \right)}{\mu Cy} \right\rceil \tag{10}$$

The set of values $\{W^0, W^1, W^2, \ldots, W^m, \ldots\}$ is monotonically non-decreasing. Hence, starting with $W^0 = 0$; when $W^{m+1} = W^m$, the solution of equation (10) has been found.

Take the example of figure 7a), for periodic variable F. The sequence of iterations (using (10)) is as follows:

[6] $\lceil x \rceil$ is a ceiling function of x, where $\lceil x \rceil = 0$ if $x < 0$ and $\lceil x \rceil$ = smallest integer greater than the fractional number on which it acts.

$$W_F^1 = \left\lceil \frac{0.184 + (\lceil 0/1 \rceil + \lceil 0/2 \rceil + \lceil 0/3 \rceil + \lceil 0/4 \rceil + \lceil 0/4 \rceil) \times 0.184}{1} \right\rceil = \lceil 0.184 \rceil = 1$$

$$W_F^2 = \left\lceil \frac{0.184 + (\lceil 1/1 \rceil + \lceil 1/2 \rceil + \lceil 1/3 \rceil + \lceil 1/4 \rceil + \lceil 1/4 \rceil) \times 0.184}{1} \right\rceil = \lceil 1.104 \rceil = 2$$

$$W_F^3 = \left\lceil \frac{0.184 + (\lceil 2/1 \rceil + \lceil 2/2 \rceil + \lceil 2/3 \rceil + \lceil 2/4 \rceil + \lceil 2/4 \rceil) \times 0.184}{1} \right\rceil = \lceil 1.228 \rceil = 2$$

and iterations stop here, since $W_F^3 = W_F^2 = 2$.

In appendix A.4, we describe in detail an algorithm to perform this feasibility test.

4. Real-Time Guarantees for the Periodic Traffic

We define the worst-case response time for an aperiodic transfer as the time interval between the placement, at time instant t_0, of the `L_FREE_UPDATE.req(ID_Ap, urgent)` in the station's local urgent queue and the completion of the buffer transfer concerning the aperiodic variable Va_i in a BA's aperiodic window.

The response time for an aperiodic buffer transfer includes the following three components:

1. the time elapsed between t_0 and the time instant when the requesting station is polled for a periodic variable, being only then able to indicate the BA (via `RP_DAT`, with the request bit set) that there is an aperiodic transfer request pending. We define this time interval as the dead interval of a producer station;
2. the time that request indication stays in the BA's urgent queue till the related `ID_RQ` + `RP_RQ` pair of frames is processed in an aperiodic window;
3. and the time the buffer exchange request for variable Va_i stays in the BA's ongoing aperiodic queue till the related `ID_DAT_Ap` + `RP_DAT` pair of frames is processed in an aperiodic window.

As was explained in sub-section 2.4, when in an aperiodic window, the BA (note that we are only considering buffer transfers - message exchanges are not supported) executes the sequence shown in figure 6.

4.1. Upper Bound for the Dead Interval

The upper bound for the dead interval in a station is determined by the minimum of the scanning periodicities of a produced variable in that station. It is important to note that a periodic variable (Vp_i) is not polled at regular intervals given by its periodicity (multiples of the microcycle). A communication jitter exists due to the way the BAT is constructed. Therefore, the upper bound for the dead interval in a producer station k is:

$$s^k = Tp_j + J_{Vp_j} + Cp_j, \text{ with } j : Tp_j = \min_{Vp_i \text{ produced in } k} \{Tp_i\} \qquad \textbf{(11)}$$

where J_{Vpi} is the maximum communication jitter that Vp_j may suffer. For example, and concerning the set of periodic variables shown in table 1, if variable F is the only produced variable in a specific station (say station k), then s^k=6+0.1952+0.0976= 6.2928ms.

Inherent to equation (11) is the following assumption: the local aperiodic request is only processed (setting the request bit in the `RP_DAT` frame) if it arrives before the start of the related `ID_DAT`. Hence we include a Cp_j in equation (11).

In appendix A.5 we describe a detailed algorithm for the evaluation of the communication jitter of a periodic variable. This algorithm is the basis for the evaluation of the dead interval in a specific *k* station.

Using this algorithm, and assuming that all Cp_i = 0.21ms as concerning variable set of table 1, the value for the communication jitter for each periodic variable is:

Table 5: Communication Jitter for Table 1 (all Cp_i = 0.21ms)

Identifier	**A**	**B**	**C**	**D**	**E**	**F**
Periodicity (ms)	0	0	0.21	0.21	0.58	0.79

as the schedule would be as follows:

Table 6: BAT Concerning Table 1 (all Cp_i = 0.21ms), RM Algorithm

	Microcycle											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bat[*A*,*cycle*]	1	1	1	1	1	1	1	1	1	1	1	1
bat[*B*,*cycle*]	1	0	1	0	1	0	1	0	1	0	1	0
bat[*C*,*cycle*]	1	0	0	1	0	0	1	0	0	1	0	0
bat[*D*,*cycle*]	1	0	0	0	1	0	0	0	1	0	0	0
bat[*E*,*cycle*]	0	1	0	0	1	0	0	0	1	0	0	0
bat[*F*,*cycle*]	0	1	0	0	0	0	1	0	0	0	0	0

4.2. Aperiodic Busy Interval

The worst-case response time for an aperiodic variable transfer occurs if at the time the request is placed in the BA's urgent queue (s^k after t_0), requests for all other aperiodic variables in the network are already pending in the same queue.

We consider that:

1. for each aperiodic variable a request for identification must be made[7];
2. and that those requests will start to contend for the medium access when the BA is starting the first microcycle of a macrocycle (with the RM scheduling policy used to construct the BAT, this means maximum periodic load). This is defined as the *critical instant*.

We also consider that all aperiodic traffic has minimum time between requests that is greater than the worst-case response time. Therefore, no other aperiodic request appears before the completion of a previous one. Hence, the maximum number of aperiodic requests pending in the BA is *na*, with *na* being the number of aperiodic different requests that can be made in the network.

We define time span between the critical instant and the end of the processing of aperiodic requests that are pending at the critical instant as an aperiodic busy interval (ABI), since all aperiodic windows within the microcycles are used to process aperiodic traffic.

It is also clear that to process all those *na* requests, the aperiodic windows will perform alternately sequences of (`ID_RQ` + `RP_RQ`) and (`ID_DAT` + `RP_DAT`), as the BA gives priority to the ongoing aperiodic queue (see figure 6).

If all the aperiodic variables have a similar length of data field (a few bytes), it is reasonable to define Ca^* as:

[7] In practice this case only occurs if each of the station have only on aperiodic produced buffer, since the BA is able to manage the urgent queue such as to avoid redundancy in `ID_RQ` to a same station.

$$Ca^* = \max_{i=1,..,na}\left\{\frac{len(\text{id\_dat}) + len(\text{rp\_dat}_i)}{bps} + 2\times t_r, \frac{len(\text{id\_rq}) + len(\text{rp\_rq})}{bps} + 2\times t_r\right\} \qquad \textbf{(12)}$$

which gives the longest transaction in an aperiodic window during the ABI.

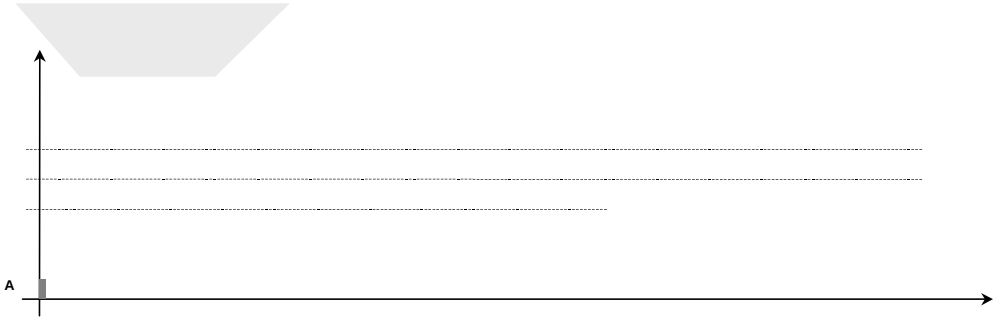
Therefore, the number of transactions with the length Ca^* to be processed during the ABI is $2\times na$, na corresponding to `ID_RQ` + `RP_RQ` transactions and another set of na corresponding to `ID_DAT` + `RP_DAT` transactions. In fact, the number of transactions (`ID_RQ` + `RP_RQ`) could be reduced to the number of stations that require aperiodic transfers. In this case however the length of (`ID_RQ` + `RP_RQ` + $2 \times t_r$) would be higher, hence leading to a higher Ca^*. Hence, considering $2\times na$ is a reasonable trade-off.

With these assumptions, the analysis for the worst-case response time for the aperiodic traffic is as follows.

4.3. Worst-Case Response Time for the Aperiodic Requests

Assume that the system configured for the 6 periodic variables (table 1), must also support 9 aperiodic buffer exchanges. Assume also that the length of the each Vp_i, $\forall_i$ is Cp_i= =Cp=0.0976ms, and that for all sporadic traffic Ca^* = 0.1ms.

If the BAT is implemented as shown in table 2, transactions concerning the 9 aperiodic variables are schedule as shown in figure 8.



and all the remaining 5 Ca^* transactions are processed. Therefore, the length of the aperiodic busy interval (ABI) is: $2\times\mu Cy+2\times0.0976+5\times0.1=2.695$ms.

For a request made at the station that produces periodic variable F (assume that F is the only periodic variable produced in that station) the dead interval (given by equation (11)) is 6.2928ms. Therefore, the worst-case response time of an aperiodic variable that is required in that station k is: Ra^k=6.2928 +2.695=8.9879ms.

We provide now analysis for the generic case of na number of aperiodic variables in a system.

The length of the aperiodic window in the l^{th} cycle (l = 1, .., N, N + 1, ..) is:

$$aw(l) = \mu Cy - \sum_{i=1}^{np} \left(bat[i,l] \times Cp_i \right) \tag{13}$$

Therefore, the number of aperiodic transactions that fit in the l^{th} aperiodic window is:

$$nap(l) = \left\lfloor \frac{aw(l)}{Ca^*} \right\rfloor \tag{14}$$

where Ca^* is as given by (12).

The number of microcycles (N') in an ABI is:

$$N' = \min\{\Psi\}, \text{ with } \Psi = \sum_{l=1}^{N'} nap(l) \wedge \Psi \geq 2 \times na \tag{15}$$

that is, the minimum number of microcycles in which the number of "slots" available for processing aperiodic transactions (each "slot" with the length of Ca^*) is at least $2\times na$.

In appendix A.6 we describe a detailed algorithm for the evaluation of the number of microcycles of an aperiodic busy interval.

Knowing N', the length of the aperiodic busy interval (len_abi) is:

$$len\_abi = (N'-1) \times \mu Cy + \sum_{i=1}^{np} \left(bat[i,N] \times Cp_i \right) + \left(2 \times na - \sum_{l=1}^{N'-1} nap(l) \right) \times Ca^* \tag{16}$$

where $\sum_{i=1,..,np}(bat[i,N]\times Cp_i)$ gives the length of the periodic window in microcycle N', and $(2\times na-\sum_{l=1,..,N'-1}nap(l))\times Ca^*$ gives the length of the aperiodic window, concerning the aperiodic busy interval, also in microcycle N'.

In appendix A.7 a detailed algorithm for evaluating the length of the ABI is provided.

Therefore, the worst-case response time for an aperiodic transfer request made at station k is:

$$Ra^k = s^k + len\_abi \tag{17}$$

and the admissible interval between consecutive aperiodic requests of a periodic variable in a station k must comply with the following condition:

$$MIT(Va^k) \geq Ra^k = s^k + len\_abi \tag{18}$$

4.4. Related Work

Both the works described in [7,8] address the issue of finding worst-case response times for aperiodic requests in WorldFIP networks. However they do not draw the analysis as

dependent of the way the BAT is built. For that reason communication jitter is not considered for the evaluation of the dead interval. Although the analysis in [8] is an improved version of that made in [7] (in [7] the authors approach very pessimistically by considering that the periodic traffic is processed after all the periodic traffic within a macrocycle), none of the analysis considered different lengths of periodic windows. More importantly, none of the analysis considered that at the end of each microcycle some transactions would not simply fit.

5. Conclusions

In this paper we provide a comprehensive study on how it is possible to support the real-time requirements of both periodic and aperiodic traffic in WorldFIP networks.

Important contributions were made by the definition of feasibility tests for periodic traffic scheduled according to the rate monotonic algorithm. Also concerning the periodic traffic, exact definition and evaluation of communication jitter is provided, as a function of the bus arbitrator table. Concerning the schedulability of the aperiodic traffic, we improved previous results by providing a more accurate definition of the dead interval and by extending the analysis to the case of non-constant periodic window lengths. We introduced also in the analysis the wasted portion in the aperiodic window.

References

[1] Lenhart, G. A Fieldbus Approach to Local Control Networks. *Advances in Instrumentation and Control*, Vol. 48, No. 1, pp. 357-365, 1993.

[2] Cardoso, A. and E. Tovar. Industrial Communication Networks: Issues on Heterogeneity and Internetworking. *Proceedings of the 6th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM'96)*, Atlanta, USA, pp. 139-148, Begell House Publishers, 1996.

[3] Cenelec. General Purpose Field Communication System. EN 50170, Vol. 1/3 (P-NET), Vol. 2/3 (PROFIBUS), Vol. 3/3 (FIP), Cenelec, 1996.

[4] Pnet. The P-NET Standard. International P-NET User Organisation ApS, 1994.

[5] Profibus. PROFIBUS Standard DIN 19245 part I and II. Translated from German, Profibus Nutzerorganisation e.V., 1992.

[6] Afnor. Normes FIP NF C46-601 to NF C46-607. Union Technique de l'Electricité, 1990.

[7] Vasques, F and G. Juanole. Pre-run-time Schedulability Analysis in Fieldbus. *Proceedings of the 20th Annual Conference of the IEEE Industrial Electronics Society (IECON'94)*, pp. 1200-1204, 1994.

[8] Pedro, P. and A. Burns. Worst Case Response Time Analysis of Hard Real-time Sporadic Traffic in FIP Networks. *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pp. 3-10, June 1997.

[9] ISO 9506. Industrial Automation Systems - Manufacturing Message Specification. ISO, 1990.

[10] Afnor. Normes FIP NF C46-605 - Gestion de Réseau. Union Technique de l'Electricité, 1990.

[11] Kim, Y., Jeong, S. and W. Kown. A Pre-Run-Time Scheduling Method for Distributed Real-Time Systems in a FIP Environment. *Control Engineering Practice*, Vol. 6, pp. 103-109, Pergamon/Elsevier Science, 1998.

[12] Kumaran, S. and J.-D. Decotignie. Multicycle Operations in a Field Bus: Application Layer Implications. *Proceedings of IEEE Annual Conference of Industrial Electronics Society (IECON'89)*, pp. 531-536, 1989.

[13] Raja, P. and G. Noubir. Static and Dynamic Polling Mechanisms for Fieldbus Networks. *Operating Systems Review*, Vol. 27, No. 3, 1993.

[14] Liu, L. and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1), pp. 46-61, 1973.
[15] Tovar, E. and F. Vasques. Setting the Bus Arbitrator Table in in WorldFIP Networks. Technical Report of the Polytechnic Institute of Porto, HURRAY-TR-9909, March 1999.
[16] Joseph, M. and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, Vol. 29, No. 5, pp. 390-395, 1986.
[17] Tovar, E., Vasques, F. and A. Burns. . Adding Loac Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach. *To appear in the Proceedings of the 11th Euromicro Workshop of Real-Time Systems*, York, England, June 1999.
[18] Audsley, N., Burns, A., Richardson, M., Tindell, K. and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5), pp. 284-292, 1993.

Appendix

A.1. Algorithm for the Evaluation of the Microcycle

```
-------------------------------------------------------------------------
- Evaluation of the Microcycle Value
-------------------------------------------------------------------------
function microcycle;
input:   np       /* number of periodic variables */
         tp[i]    /* vector containing the periodicity of the variables */
output:  µCy      /* value of the microcycle */

begin
   1:    min = MAXINT;
   2:    for i = 1 to np do
   3:       if tp[i] < min then
   4:          min = tp[i]
   5:       end if
   6:    end for;
   7:    µCy = min + 1;
   8:    repeat
   9:       µCy = µCy - 1;
   10:      ctrl = TRUE;
   11:      for i = 1 to np do
   12:         if tp[i] mod µCy <> 0 then
   13:            ctrl = FALSE
   14:
```

```
   12:      for i = 1 to np do
   13:         if N mod (tp[i]/μCy) <> 0 then
   14:            ctrl = FALSE
   15:         end if
   16:      end for
   17:   end while;
   18:   MCy = N × μCy;
return MCy;
--------------------------------------------------------------------------
```

A.3. Rate Monotonic (RM) Algorithm for Building the BAT

```
--------------------------------------------------------------------------
- Rate Monotonic for Building the BAT
--------------------------------------------------------------------------
function rm_bat;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of the variables */
                  /* i ranging from 1 to np */
                  /* and the length of Cpi, j ranging from 1 to 2 */
         μCy      /* value of the microcycle */
         N        /* number of microcycles in the macrocycle */
output:
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
begin
   1:    for i = 1 to np do
   2:       cycle = 1;
   3:       repeat
   4:          if load[cycle] + Vp[i,2] <= μCy then
   5:             bat[i,cycle] = 1;
   6:             load[cycle] = load[cycle] + 1;
   7:             cycle = cycle + (Vp[i,1] div μCy)
   8:          else;
   9:             cycle1 = cycle1 + 1;
   10:            ctrl = FALSE;
   11:            repeat
   12:               if load[cycle1] + Vp[i,2] <= μCy then
   13:                  ctrl = TRUE
   14:               end if;
   15:            until (ctrl = TRUE) or (cycle1 >= (cycle + (Vp[i,1]
   16:                                 div μCy)));
   17:            if cycle1 >= (cycle + (Vp[i,1] div μCy)) then
   18:               bat[i,cycle1] = 1;
   19:               load[cycle1] = load[cycle1] + 1;
   20:               cycle = cycle + (Vp[i,1] div μCy)
   21:            else
   22:               /* MARK Vpi NOT SCHEDULABLE with RM Algorithm */
   23:               cycle = cycle + (Vp[i,1] div μCy)
   24:            end if
   25:         end if
   26:      until cycle > N
   27:   end for
return bat;
--------------------------------------------------------------------------
```

A.4. Algorithm for the Feasibility Test of the BAT

```
------------------------------------------------------------------------
- Number of Microcycles to Process a Buffer Transfer Using the RM al.
------------------------------------------------------------------------
function NR;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of the variables */
                  /* i ranging from 1 to np */
                  /* and the length of Cpi, j ranging from 1 to 2 */
         µCy      /* value of the microcycle */
         N        /* number of microcycles in the macrocycle */
output:
         NR[i] /* i ranging from 1 to np */

begin
   1:    for i = 1 to np do
   2:       w = 0; w1 = 0;
   3:       repeat
   4:          w = w1;
   5:          for j = 1 to i - 1 do
   6:             if ((w × µCy) mod Vp[j,1]) <> 0 then
   7:                w1 = w1 + (((w × µCy) div Vp[j,1]) + 1) × Vp[j,2]
   8:             else
   9:                w1 = w1 + (((w × µCy) div Vp[j,1])) × Vp[j,2]
   10:               end if
   11:         end for;
   12:         if (Vp[i,2] + w1) mod µCy <> 0 then
   13:            w1 = w1 + ((Vp[i,2] + w1) div µCy) + 1
   14:         else
   15:            w1 = w1 + ((Vp[i,2] + w1) div µCy)
   16:         end if
   17:      until w1 = w;
   18:      NR[i] = w1
   19:   end for
return NR;
------------------------------------------------------------------------
```

A.5. Algorithm for the Evaluation of the Communication Jitter

```
------------------------------------------------------------------------
- evaluation of the maximum communication jitter of a periodic variable
------------------------------------------------------------------------
function NR;
input:   np       /* number of periodic variables */
         Vp[i,j]  /* array containing the periodicity of the variables */
                  /* i ranging from 1 to np */
                  /* and the length of Cpi, j ranging from 1 to 2 */
         µCy      /* value of the microcycle */
         N        /* number of microcycles in the macrocycle */
         bat[i,cycle]   /* i ranging from 1 to np */
                        /* cycle ranging from 1 to N */
output:  J[i]     /* maximum polling jitter of variable Vpi */

begin
   1:    for i = 1 to np do
   2:
   3:       /* Evaluate number of hits of variable Vpi */
   4:       hits = 0;
   5:       for cycle = 1 to N do
```

```
6:          if bat[i,cycle] = 1 then
7:             hits = hits + 1;
8:          end if
9:       end for;
10:
11:      /* Find first hit in a macrocycle */
12:      cycle = 0;
13:      repeat
14:         cycle = cycle + 1
15:      until bat[i,cycle] = 1;
16:
17:      /* Find last hit in a macrocycle */
18:      cycle1 = N + 1;
19:      repeat
20:         cycle1 = cycle1 - 1
21:      until bat[i,cycle1] = 1;
22:
23:      /* Evaluate the time span between the last hit in a */
24:      /* macrocycle and the first hit in a subsequent macrocycle */
25:      span = (N - cycle1 + cycle - 1) × μCy;
26:      load_par = 0;
27:      for j = 1 to i - 1 do
28:         if bat[j,cycle] = 1 then
29:            load_par = load_par + Vp[j,2]
30:         end if
31:      end for;
32:      span = span + load_par;
33:      load_par = 0;
34:      for j = 1 to i - 1 do
35:         if bat[j,cycle1] = 1 then
36:            load_par = load_par + Vp[j,2]
37:         end if
38:      end for;
39:      span = span + (μCy - load_par);
40:
41:      /* Evaluate the time span between each of the hits */
42:      /* within a macrocycle */
43:      for k = 1 to hits - 1 do
44:         cycle1 = cycle;
45:         repeat
46:            cycle1 = cycle1 + 1
47:         until bat[i,cycle1] = 1;
48:         span1 = (cycle1 - cycle - 1) × μCy;
49:         load_par = 0;
50:         for j = 1 to i - 1 do
51:            if bat[j,cycle] = 1 then
52:               load_par = load_par + Vp[j,2]
53:            end if
54:         end for;
55:         span1 = span1 + (μCy - load_par);
56:         load_par = 0;
57:         for j = 1 to i - 1 do
58:            if bat[j,cycle1] = 1 then
59:               load_par = load_par + Vp[j,2]
60:            end if
61:         end for;
62:         span1 = span1 + load_par;
63:
64:         if span1 > span then
65:            span = span1
66:         end if;
67:         cycle = cycle1;
68:      end for;
```

```
   69:      J[i] = span - Vp[i,1];
   70:   end for
return J;
-------------------------------------------------------------------------
```

A.6. Algorithm for the Evaluation of the Number of Microcycles in an ABI

```
-------------------------------------------------------------------------
- Number of Cycles of the Aperiodic Busy Interval
-------------------------------------------------------------------------
function ncy_apbi;
input:   np       /* number of periodic variables */
         na       /* number of aperiodic variables */
         μCy      /* value of the microcycle */
         bat[i,l] /* i ranging from 1 to np */
         cra      /* length of composite aperiodic transaction */
         cp       /* length of periodic elementary transaction */
output:  ncy_abi  /* number of cycles of the aperiodic busy interval */

begin
   1:    cycle = 0;
   2:    na_aux = 0;
   3:    repeat
   4:       cycle = cycle + 1;
   5:       count_p = 0;
   6:       for i = 1 to np do
   7:          if bat[i,cycle] = 1 then
   8:             count_p = count_p + 1;
   9:          end if;
   10:      end for;
   11:      aw = μCy - count_p × cp;
   12:      na_aux = na_aux + aw div ca;
   13:   until na_aux >= 2 × na;
   14:   ncy_abi = cycle;
return ncy_abi;
-------------------------------------------------------------------------
```

A.7. Algorithm for the Evaluation of the Length of the ABI

```
-------------------------------------------------------------------------
- Length of Aperiodic Busy Interval
-------------------------------------------------------------------------
function len_apbi;
input:   np       /* number of periodic variables */
         na       /* number of aperiodic variables */
         μCy      /* value of the microcycle */
         bat[i,l] /* i ranging from 1 to np */
         cra      /* length of composite aperiodic transaction */
         cp       /* length of periodic elementary transaction */
         ncy_abi  /* number of cycles of the aperiodic busy interval */
output:  len_abi  /* number of cycles of the aperiodic busy interval */

begin
   1:    /* determine number of aperiodic transfers in the */
   2:    /* ncy_abi - 1 microcycles */
   3:    for cycle = 1 to ncy_abi - 1 do
   4:       count_p = 0;
   5:       for i = 1 to np do
```

```
   6:          if bat[i,cycle] = 1 then
   7:             count_p = count_p + 1
   8:          end if
   9:       end for;
   10:         aw = µCy - count_p × cp;
   11:         na_aux = na_aux + aw div cra
   12:   end for;
   13:
   14:   /* determine the number of periodic scans in microcycle */
   15:   /* number ncy_abi */
   16:   count_p = 0;
   17:   for i = 1 to np do
   18:         if bat[i,ncy_abi] = 1 then
   19:            count_p = count_p + 1
   20:         end if
   21:   end for;
   22:   len_p = count_p × cp;
   23:
   24:   /* determine length of aperiodic busy window */
   25:   len_abi = (ncy_abi -1) × µCy + len_p + (2 × na - na_aux) × ca*
return len_abi;
-------------------------------------------------------------------------
```