# USING WORLDFIP NETWORKS TO SUPPORT PERIODIC AND SPORADIC REAL-TIME TRAFFIC

Eduardo Tovar
Department of Computer Engineering
Polytechnic Institute of Porto
e-mail: emt@dei.isep.ipp.pt

Francisco Vasques
Department of Mechanical Engineering
University of Porto
e-mail: vasques@fe.up.pt

**Abstract** - In this paper we address the ability of WorldFIP to cope with the real-time requirements of distributed computer-controlled systems (DCCS). Typical DCCS include process variables that must be transferred between network devices both in a periodic and sporadic (aperiodic) basis. The WorldFIP protocol is designed to support both types of traffic. WorldFIP can easily guarantee the timing requirements for the periodic traffic. However, for the aperiodic traffic more complex analysis must be made in order to guarantee its timing requirements. This paper describes work that is being carried out to extend previous relevant work [1,2], in order to include the actual schedule for the periodic traffic in the worst-case response time analysis of sporadic traffic in WorldFIP networks.

## I. INTRODUCTION

A WorldFIP [3] 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 by 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, which is used to support manufacturing message services (MMS) [4], is out of the scope of this paper.

### *A. Producer/Distributor/Consumer Concept*

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) is made up of a set of 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 stations that has the consumed buffers accepts the value contained in the $RP\_DAT$, overwriting the previous value. Fig. 1 illustrates these mechanisms.

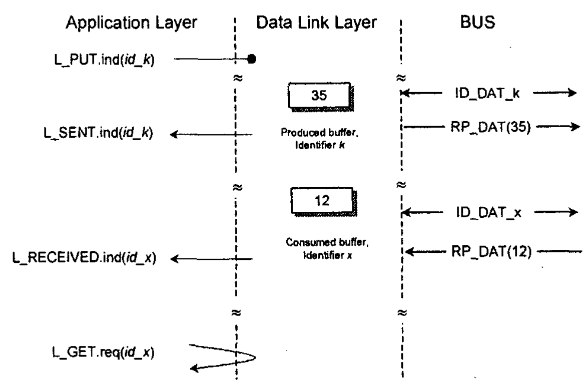


Fig. 1 - AL Services to Access DLL Buffers

### *B. 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. Fig. 2 illustrates the concept of an elementary transaction in WorldFIP.

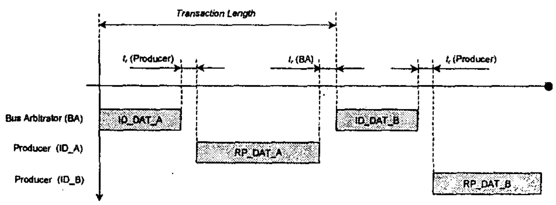


Fig. 2 - WorldFIP Elementary Transaction

Every transmitted frame is encapsulated with control information from the physical layer. Specifically, the frame is placed between a DTR field (begin of frame) and an FTR field (end of frame). All WorldFIP frames begin with a control byte, which is used by network stations to recognise the type of frame, and end with two FCS (Frame Check Sequence) bytes, used by the frame receiver to verify the integrity of the received frame. As a consequence, an *ID_DAT* frame has always 64 bits, whereas a *RP_DAT* frame has at least 48 bits, and the length of a message transaction is:

$$C = \frac{len(\text{id\_dat}) + len(\text{rp\_dat})}{bps} + 2 \times t_r \qquad (1)$$

where *bps* stands for the network data rate and *len*(<*frame*>) is the length, in bits, of frame <*frame*>.

For instance, assuming that all variables have a data field with 4 bytes (all *RP_DAT* have 92 bits), if $t_r = 20\mu s$ and the network data rate is 2.5Mbps then, the duration of an elementary transaction will be $(64+80)/2.5+2\times 20=$ $=97.6\mu s$ (equation (1)).

### *C. WorldFIP 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 section II.

Assume a distributed system within which 6 variables are to be periodically scanned, with 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 Fig. 3, where we consider $C = 97{,}6\mu s$ for each elementary transaction.

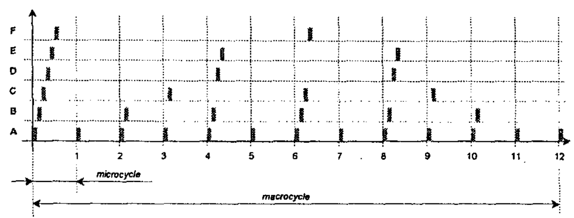


Fig. 3 - Schedule for Periodic Scans (Table 1)

It is easy to depict that, for this example, the sequence of microcycles repeats each 12 microcycles. This sequence of microcycles is said to be a *macrocycle*, and its length is given by the *lowest common multiple* (LCM) of the scan periodicities.

An important characteristic of the HCF/LCM approach is that 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" within the microcycle. All other variables suffer from a slight communication jitter.

It is also worth mentioning that the schedule shown in Fig. 1 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 Section II), message transfers and padding identifiers. A WorldFIP BA can also manage asynchronous microcycles, not transmit padding identifiers at the end of the microcycle.

## II. APERIODIC BUFFER TRANSFERS

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 until the end of the microcycle is denoted as the *aperiodic window* of the microcycle. The aperiodic buffer transfers take place in three stages (Fig. 4):

1. When processing the BAT schedule, the BA broadcasts an *ID_DAT* frame concerning a periodic

variable, say identifier $X$. The producer of variable $X$ 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.
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 its ongoing aperiodic queue. For each transfer, the BA uses the same mechanism as the used for the periodic buffer transfers ($ID\_DAT$ followed by $RP\_DAT$).

It is 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.

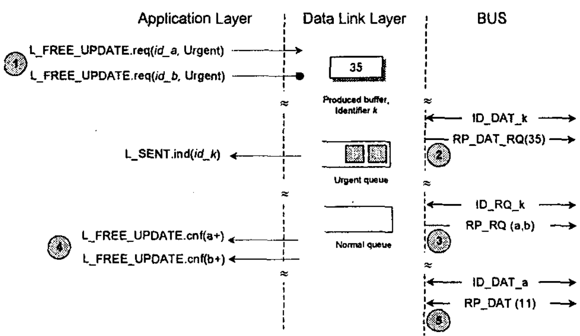


Fig. 4 - WorldFIP Aperiodic Buffer Transfers

In this paper we consider that all aperiodic requests concern the use of the application layer service `L_FREE_UPDATE.req(ID_*,Urgent)`, thus, only the urgent queues (both at the requesting station and at the BA) are considered. 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 Fig. 5. It is important to note that at the end of a microcycle, a transaction is processed if, and only if, there is still time to do it.

## III. RESPONSE TIME OF SPORADIC TRAFFIC

### A. Model for the Periodic Transfers

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

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

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 (1)).

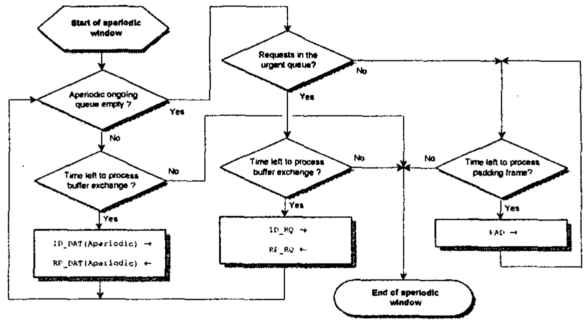


Fig. 5 - BA Management of Aperiodic Buffer Transfers

### B. Setting the Bus Arbitrator Table

Following the HCF/LCM methodology to build the BAT, the value for the microcycle ($\mu Cy$) is chosen as:

$$\mu Cy = \underset{i=1,\ldots,np}{HCF}(Tp_i) \tag{3}$$

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

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

The macrocycle ($MCy$) is defined as:

$$MCy = N \times \mu Cy \tag{5}$$

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

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

The BAT can be easily built according to a rate monotonic (RM) algorithm [5]. In [6] the authors show how the RM can be adapted to encompass the WorldFIP characteristics.

For the example of Table 1 (with $Cp_i = 0.0976$ms, $\forall_i$) and considering the RM algorithm, the BAT 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 up to $np$, and $j$ ranging from 1 up to $N$ (number of microcycles in a macrocycle).

Below we give the pseudo-code details of the algorithm for building the BAT using the RM scheme.

The algorithm gives whether all traffic is schedulable or not (line 21). In the algorithm, the vector *load*[ ] is used to store the load in each microcycle as the traffic is scheduled. 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*, ]).

```
---------------------------------------------------------------
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 */
                /* ORDERED by periodicities */
                /* 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] div µCy)));
 16:        if cycle1>=(cycle+(Vp[i,1] div µCy)) then
 17:          bat[i,cycle1] = 1;
 18:          load[cycle1] = load[cycle1] + 1;
 19:          cycle = cycle + (Vp[i,1] div µCy)
 20:        else
 21:          /* MARK Vpi NOT SCHEDULABLE */
 22:          cycle = cycle + (Vp[i,1] div µCy)
 23:        end if
 24:      end if
 25:    until cycle > N
 26:  end for
return bat;
---------------------------------------------------------------
```

Note that by using the RM algorithm some of the variables with longer periods can be scheduled in subsequent microcycles, thus inducing an increased communication jitter for those variables.

### C. Assumptions for the Aperiodic Traffic

We define the worst-case response time for an aperiodic transfer as the time interval between placing, at time instant $t_0$, the `L_FREE_UPDATE.req(ID_Va_i, urgent)` in the 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 associated to 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 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 the 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.

### D. Upper Bound for the Dead Interval

The upper bound for the dead interval in a station $k$ is related to the smallest scanning period of a produced variable in that station. It is important to note that a periodic variable ($Vp_i$) is not polled at regular intervals, since there is a communication jitter inherent to the BAT setting. Therefore, the upper bound for the dead interval in a station $k$ is:

$$\sigma^k = Tp_j + J_{Vp_j} + Cp_j,\ \text{with}\ Vp_j : Tp_j = \min_{Vp_i\ \text{produced in}\ k} \{Tp_i\} \quad (7)$$

where $J_{Vp_j}$ is the maximum communication jitter of $Vp_j$.

The following assumption is inherent to (7): a 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, the term $Cp_j$ is included in (7).

Below we give the pseudo-code details of the algorithm for the evaluation of the communication jitter associated to a periodic variable. This algorithm is the basis for the evaluation of the dead interval in a specific $k$ station.

Assuming the variable set of table 1, with all $Cp_i$ = 0.21ms, the value for the communication jitter for each periodic variable is:

Table 3. Communication Jitter for Table 1

| Identifier | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Comm. Jitter (ms) | 0 | 0 | 0.21 | 0.21 | 0.58 | 0.79 |

```
---------------------------------------------------------------
- evaluation of the maximum comm. jitter of a periodic var
---------------------------------------------------------------
function Jitter;
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
  for i = 1 to np do

    /* Evaluate number of hits of variable Vpi */
    hits = 0;
    for cycle = 1 to N do
      if bat[i,cycle] = 1 then
        hits = hits + 1;
      end if
    end for;

    /* Find first hit in a macrocycle */
    cycle = 0;
    repeat
      cycle = cycle + 1
    until bat[i,cycle] = 1;

    /* Find last hit in a macrocycle */
    cycle1 = N + 1;
    repeat
      cycle1 = cycle1 - 1
```

```
    until bat[i,cycle1] = 1;

    /* Evaluate time span between the last hit in a */
    /* macrocycle and the 1st in a subsequent */
    span = (N - cycle1 + cycle - 1) × μCy;
    load_par = 0;
    for j = 1 to i - 1 do
      if bat[j,cycle] = 1 then
        load_par = load_par + Vp[j,2]
      end if
    end for;
    span = span + load_par;
    load_par = 0;
    for j = 1 to i - 1 do
      if bat[j,cycle1] = 1 then
        load_par = load_par + Vp[j,2]
      end if
    end for;
    span = span + (μCy - load_par);

    /* Evaluate time span between each of the hits */
    /* within a macrocycle */
    for k = 1 to hits - 1 do
      cycle1 = cycle;
      repeat
        cycle1 = cycle1 + 1
      until bat[i,cycle1] = 1;
      span1 = (cycle1 - cycle - 1) × μCy;
      load_par = 0;
      for j = 1 to i - 1 do
        if bat[j,cycle] = 1 then
          load_par = load_par + Vp[j,2]
        end if
      end for;
      span1 = span1 + (μCy - load_par);
      load_par = 0;
      for j = 1 to i - 1 do
        if bat[j,cycle1] = 1 then
          load_par = load_par + Vp[j,2]
        end if
      end for;
      span1 = span1 + load_par;

      if span1 > span then
        span = span1
      end if;
      cycle = cycle1;
    end for;
    J[i] = span - Vp[i,1];
  end for
return J;
---------------------------------------------------------------
```

### E. Aperiodic Busy Interval

The worst-case response time for an aperiodic variable transfer occurs if, when the request is placed in the BA's urgent queue ($\sigma^k$ after $t_0$), the queue is already with requests for all the other aperiodic variables in the network. We consider that:

1. for each aperiodic variable a request for identification must be made, and thus the network load is maximised;
2. and that those requests will start to contend for the medium access at the begin of the macrocycle: *critical instant*.

We also consider that all aperiodic traffic has a minimum inter-arrival time between requests, which is greater than its 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 requests (to different station can require an aperiodic buffer transfer of a same variable) that can be made in the network.

We define the 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 Fig. 2). If all the aperiodic variables have a similar length, $Ca^*$ may be defined as the maximum length of all the ($ID\_RQ$ / $RP\_RQ$) and ($ID\_DAT$ / $RP\_DAT$) transactions concerning the aperiodic traffic. Therefore, the number of transactions to be processed during the ABI is $2 \times na$, corresponding to the set of $ID\_RQ$ / $RP\_RQ$ transactions and the set of $ID\_DAT$ / $RP\_DAT$ transactions.

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

### F. Worst-Case Response Time

The worst-case response time of aperiodic transfers is function of the network periodic load during the ABI, since it bounds the aperiodic windows length. The length of the aperiodic window in the $l^{th}$ cycle ($l = 1, .., N, N+1, ..$) may be evaluated as follows:

$$aw(l) = \mu Cy - \sum_{i=1}^{np} \left(bat[i,l^*] \times Cp_i\right) \tag{8}$$

with $bat[i,l^*]$ as defined in sub-section III.B, and $l^* = [(l-1) \bmod N] + 1$. Therefore, the number of aperiodic transactions that fit in the $l^{th}$ aperiodic window is:

$$nap(l) = \left\lfloor \frac{aw(l^*)}{Ca^*} \right\rfloor \tag{9}$$

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

$$N' = \min\{\Psi\}, \quad \text{with } \Psi = \sum_{l=1}^{N'} nap(l^*) \wedge \Psi \geq 2 \times na \tag{10}$$

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

Below we give the pseudo-code details of the algorithm for the evaluation of the number of microcycles of an aperiodic busy interval.

```
---------------------------------------------------------------
- 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,1]   /* i ranging from 1 to np */
             ca         /* length of any aperiodic transaction */
             cp         /* length of all periodic transaction */
output:      ncy_abi    /* number of cycles of the abi */

begin
    cycle = 0;
    na_aux = 0;
    repeat
      cycle = cycle + 1;
      count_p = 0;
      for i = 1 to np do
        if bat[i,cycle] = 1 then
```

```
          count_p = count_p + 1;
      end if;
    end for;
    aw = µCy - count_p x cp;
    na_aux = na_aux + aw div ca;
  until na_aux >= 2 x na;
  ncy_abi = cycle;
return ncy_abi;
```

Knowing $N'$, the length of the aperiodic busy interval (*len_abi*) may be evaluated as follows:

$$len\_abi = (N'-1)\times \mu Cy + \\ +\sum_{i=1}^{np}(bat[i,N^*]\times Cp_i)+\left(2\times na-\sum_{l=1}^{N'-1}nap(l^*)\right)\times Ca^* \quad (11)$$

where $\sum_{i=1,..,np}(bat[i,N^*]\times Cp_i)$ gives the length of the periodic window in microcycle $N'$, with $N^*=[(N'-1) \bmod N] + 1$, 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 $N'$.

Below we give the pseudo-code details of the 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 */
         ca        /* length of any aperiodic transaction */
         cp        /* length of all periodic transactions */
         ncy_abi   /* number of microcycles of the abi */
output:  len_abi   /* length of the aperiodic busy interval */
begin
  /* determine number of aperiodic transfers in */
  /* the ncy_abi - 1 microcycles */
  for cycle = 1 to ncy_abi - 1 do
    count_p = 0;
    for i = 1 to np do
      if bat[i,cycle] = 1 then
        count_p = count_p + 1
      end if
    end for;
    aw = µCy - count_p x cp;
    na_aux = na_aux + aw div ca
  end for;

  /* determine the number of periodic scans */
  /* in microcycle number ncy_abi */
  count_p = 0;
  for i = 1 to np do
    if bat[i,ncy_abi] = 1 then
      count_p = count_p + 1
    end if
  end for;
  len_p = count_p x cp;

  /* determine length of aperiodic busy window */
  len_abi = (ncy_abi -1) x µCy + len_p + (2 x na -
     na_aux) x ca
return len_abi;
```

The worst-case response time for an aperiodic transfer requested at station $k$ is:

$$Ra^k = \sigma^k + len\_abi \quad (12)$$

and the maximum admissible interval between consecutive aperiodic requests of an aperiodic variable in a station $k$ is:

$$MIT(Va^k) \geq Ra^k = \sigma^k + len\_abi \quad (13)$$

### *G. Example of Aperiodic Traffic Scheduling*

Assume that a system configured for 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 $Cp_i=Cp=0.0976$ms, and that for sporadic traffic $Ca^*=0.1$ms. If the BAT is implemented as shown in table 2, transactions concerning the 9 aperiodic variables are schedule as shown in Fig. 6.

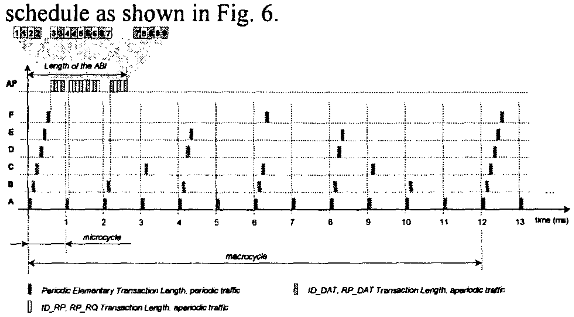


Fig. 6 - Example of Aperiodic Traffic Schedule

## IV. CONCLUSIONS

In this paper we addressed the ability of WorldFIP to cope with the real-time requirements of distributed computer-controlled systems (DCCS).

This paper describes work that is being carried out to extend previous relevant work [1,2], in order to include the actual schedule for the periodic traffic in the worst-case response time analysis of sporadic traffic in WorldFIP networks.

## V. REFERENCES


[1] Vasques, F and G. Juanole. Pre-run-time Schedulability Analysis in Fieldbus Networks. *Proceedings of IECON'94*, pp. 1200-1204, 1994.

[2] 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.

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

[5] Liu, L. and J. Layland, 1973, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM*, 20(1), (ACM) 46-61.

[6] Tovar, E. and F. Vasques. Factory Communications: On the Configuration of the WorldFIP Bus Arbitrator Table. March 1999, technical report, available at http://www.hurray.isep.ipp.pt.