Schedulability Analysis of CAN based Systems with Precedence Constraints

C. Amaya, F. Díaz del Río, J. L. Sevillano, G. Jiménez, S. Vicente, A. Civit Balcells.
University of Seville.

We present a schedulability analysis of CAN based system. Existing analyses consider that the worst-case occurs when all task (and messages) are released simultaneously (pessimistic assumption). In this paper, a systematic approach to precedence analysis is presented, considering the interference between task sequences instead of interference between single tasks. We also apply this analysis to a DX bus (a CAN based commercially available wheelchair controller), and we compare its predictions with those of a classical analysis. Results show that we obtain more precise estimations (≈14%)

1. Introduction

Hard real-time systems need accurate scheduling analysis in order to guarantee their temporal behaviour. This analysis computes the Worst-Case Response Time (WCRT) of processes and then concludes that the system is schedulable if all WCRTs are lower than their associated deadlines. Distributed control systems, which have recently gained great importance, present some difficulties because of the effects of communication between tasks. The usual approach in the literature is to directly apply single processor analysis, considering all tasks running independently. As a result, the worst-case occurs when all tasks, including messages, are released simultaneously. This approximation may produce pessimistic assumptions with important economic consequences.

A relatively simple way to improve the analysis is to consider that tasks form ***sequences*** with precedence constrains. This extension is especially useful for end-to-end real-time systems, with processes distributed along several computation nodes. Synchronisation is accomplished through tasks precedence constrains. When several tasks belonging to the same sequence run on the same node, then precedence constrains prevent these tasks to compete for the computational node because they are not released at the same time. This case is not unusual. For instance, a typical task sequence include several transmissions through the common channel between nodes, and all these messages are tasks of the "channel" node that belongs to the same sequence. If the scheduling analysis is able to include this effect predictions would significantly improve.

Several works in the literature partially include some of the effects of precedence constrains but in a heuristic way and limited to specific approach to precedence analysis, considering the interference between sequences instead of interference between single tasks. We first describe the model, with some simple examples, but avoiding formal proofs (a more detailed treatment can be found in []). Then we compare its predictions with those of a classical analysis when applied to a DXBus (CAN like) system. Finally, we briefly present some conclusions.

2. System Model.

The computing model characterises dedicated hard real time distributed systems widely used in CAN control applications. It assumes a fixed set of multitasking single processor nodes, connected by one or more CAN communication channels.

Every node uses pre-emptive scheduling to select running task. The number of processing nodes, task set and their priorities are fixed. There is no special treatment of CAN communication channel in the model, since it can be described as a multitasking node using pre-emptive scheduling of message transmission [14][1]. Tasks are arranged as sequences. A sequence is a set of task connected with precedence relationships. A task precedes another task if the second task is released (enters in runnable state) immediately after the first task ends. The model makes following assumptions on sequences[2]:

· There is neither ramifications nor cycles in chain of task executions: no more than one task is released after any task ends. There is only one starting task and only one (but not

[1] The model and algorithm described in this work can be directly used if communication system can be modelled in this way. It includes point-to-point communications and many field buses.

[2] Though the concept of sequence is widely used in end-

necessarily different) ending tasks in the sequence.

· The release of any sequence is strictly periodic. Non-periodic sequences can be included in the model if there is a minimum time between successive releases of fist task. The term "period" means this time in non-periodic sequences.

· The deadline of any sequence release is always less or equal to its period.

Table I. Model parameters.

	Description.
S_i	Refers to sequence number ***i***.
$\boldsymbol{D_i}$	Deadline of sequence ***i***.
$\boldsymbol{P_i}$	Period or minimum time between successive releases of sequence ***i***.
$\boldsymbol{R_i}$	Estimation of the worst-case response time (WCRT) of sequence ***i***.
$\boldsymbol{T_{i,j}}$	Refers to ***j***-th task of sequence S_i.
$\boldsymbol{E_{i,j}}$	Maximum task execution time (maximum $T_{i,j}$ run time if no one task pre-empts it)
$\boldsymbol{e_{i,j}}$	Minimum task execution time of $\boldsymbol{T_{i,j}}$.
$\boldsymbol{N_{i,j}}$	Node number assigned to the processing node where $T_{i,j}$ runs.
$\boldsymbol{Pri_{i,j}}$	Execution priority of $T_{i,j}$.
$\boldsymbol{B_{i,j}}$	Blocking time of $T_{i,j}$ (maximum $T_{i,j}$ waiting time for exclusive use of resources).
$\boldsymbol{R_{i,j}}$	Two meanings, depending on what method is used to calculate sequence WCRT: · WCRT of task $T_{i,j}$, or · Response time of task $T_{i,j}$ when a release of S_i has worst response time.
$\boldsymbol{\{H(T_{i,j})\}}$	Refers to set of tasks in node $N_{i,j}$ with higher priority than $T_{i,j}$.

Model assumptions and parameters (table I) can be found, with changes in terminology, in several works in the literature about end-to-end systems schedulability [11][13][14]. This work arranges model parameters as matrices, changing the usual notation to show the matrix structure (table I). Parameters on sequences have just one sub-index, their sequence number. They are arranged as ***m***-elements column vectors, where ***m*** is the number of sequences in the system.

Parameters on tasks have two sub-indexes: the first one is sequence numbering. The second one is the task execution order in any sequence release. These parameters are arranged in ***mxn*** matrices, where ***m*** is the number of sequences and ***n*** is the highest task number in any sequence.

Matrix structure reflects in a simple and compact way the sequences structure of the system. Moreover, it easily allows comparing different methods for WCRT in end-to-end systems.

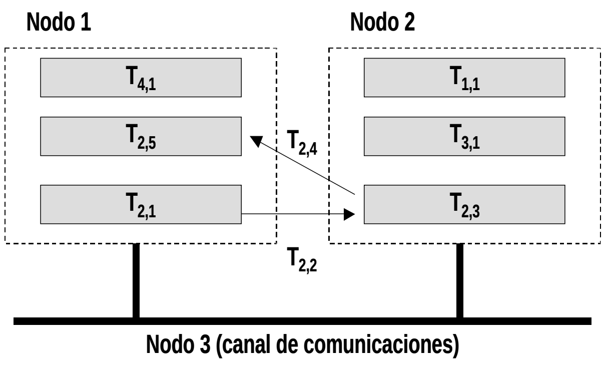


Fig. 1. Simple end-to-end system, with just tree processing nodes and four sequences.

3. Task WCRT in single processor systems.

Independent tasks in a single processor system are modelled as single task sequences. So, all parameters are arranged as column vectors, as well as sequences/ tasks WCRT. These can be computed directly applying Rate Monotonic Analysis RMA. For a given task, $T_{i,j}$, its WCRT is computed analysing how system behaves when any task releases reaches WCRT. We will name this ***critical behaviour for*** $T_{i,j}$.

There are a huge number of different critical behaviours for a given task, even for relatively simple system. However task response is identical in all of then. RMA uses this by computing response time of the simplest critical behaviour for every task. Fig. 2 illustrates the simplest critical behaviour for a sample task. The sample task and all higher priority tasks are released simultaneously and they all run during their maximum execution time. As any task release is periodic, some higher priority tasks can be released several times before the delayed task finishes (see $T_{1,1}$ in Fig. 2)

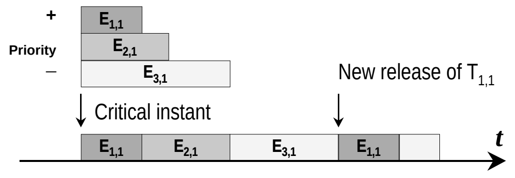


Fig. 2.Critical behaviour for $T_{3,1}$ in a single processor system with 3 tasks/sequences.

Several works have analysed critical behaviour in single processor systems, developing methods to compute task WCRT [4][5][8].

A widely used one, by Audsley et al. [4], states that task $T_{i,j}$ WCRT can be computed using equation:

$$R_{i,j} = B_{i,j} + E_{i,j} + {}_{T_{l,m} \text{ë} \{H(T_{i,j})\}} \frac{R_{i,j}}{P_l} E_{l,m} \qquad (1)$$

where B and E contains delays inherent to we can consider the summatory as the ***interference with task*** $T_{i,j}$, ***$In_{i,j}$***; that is, the maximum delay in $T_{i,j}$ from higher priority tasks, during a time interval of length $R_{i,j}$. Audsley et al demonstrate that equation (1) can be solved iteratively [4].

We will name ***interference function*** any function getting an estimation of the interference with a given task during a given time interval.

At n-th step of the iterative process associated with equation (1), the interference function can be denoted as:

$$In^n_{i,j} = In\left(R^n_{i,j}\right) = \underset{T_{l,m}\text{ë}\{H(T_{i,j})\}}{} \frac{R^n_{i,j}}{P_{l,m}} E_{l,m} \quad (2)$$

Equation (1) is applied task by task to all tasks in the system. Matrix organisation of parameters allows equation (1) to be expressed in matrix form, including all tasks:

$$R^{n+1} = B + E + \boldsymbol{In}\left(R^n\right) \qquad (3)$$
$$R^0 = B + E$$

In(R) is now a vector function operating over task WCRT vector. Elements in the resulting vector are computed using equation (2).

4. Sequence WCRT in distributed systems.

Equation (3) can be used in end-to-end systems replacing vectors with ***m***x***n*** matrices (***m*** is the number of sequences and ***n*** is the maximum number of tasks in any sequence). If we find a function ***In(R)*** satisfying the following conditions:

C1. ***R^n*** converges to some matrix ***R*** after a finite number of iterations in (3).

C2. Response time for any sequence S_i is less than or equal to the sum of ***i***-row elements in matrix ***R***.

The resulting matrix, ***R***, can be used to estimate WCRT of any sequence S_i just adding the elements in row ***i***.

Variations in estimation methods fundamentally came from different ways to deal with precedence constrains. Such variations can be expressed in equation (3) as different interference functions.

The easiest method to compute matrix ***In*** is simply use equation (2) for every element.

This method is widely used in practical end-to-end system analysis, because the interference function is simple and the iterative process satisfies conditions C1 and C2,

However, under certain conditions, sequence WCRT can be overestimated since precedence constrains are ignored.

Consider the example showed in Fig. 1. $T_{4,1}$ has the highest priority and a much longer period than any other tasks. In any release of equation (2) is used as interference function, interference of $T_{4,1}$ is included twice in the WCRT estimation for S2, since it is independently added to WCRT of $T_{2,1}$ and $T_{2,5}$. This method overestimates sequence WCRT because critical behaviours always have lower response time.

These effects appear when any sequence has more than one task running in a processing node. We will say then that the sequence has ***recurrent tasks*** in this node. Sequences with ***recurrent tasks*** are common in distributed systems, mainly when there is shared communication links, like CAN systems. As the CAN bus is treated as a processing node, there are recurrent tasks in a sequence when there are two or more message transmissions included in it.

A wide range of recurrence-associated effects causes overestimation of sequence WCRTs. Several works deal with some of them to compute more accurate estimations than RMA [13][12]. However, computations always use heuristic methods to deal partially with precedence constrain effects. There is no systematic approach for the treatment of precedence constrains. In fact, works revised show correctness in case studies, but they neither give nor reference demonstrations on general correctness of results.

Next sections present the main features of a new computing method, named MIBS (Maximum Interference Between Sequences), for estimating sequence WCRT. Correctness demonstrations cannot be included it this work due to space limitations, but they can be found in [2], as well as detailed descriptions of method and its algorithms.

5. The MIBS method.

The MIBS method computes function ***In(R)*** taking into account task precedence constrains, getting accurate estimations of sequence WCRT. The mayor distinction from other methods is that it studies critical behaviour of sequences, instead of tasks. There is also a significant change in the meaning of matrix ***R*** elements. Now, ***$R_{i,j}$*** is the ***response time of*** $T_{i,j}$ ***for any release of*** S_i ***having WCRT.*** So, $R_{i,j}$ can be shorter than WCRT of $T_{i,j}$.

6. Computing interference matrix.

Function ***In(R^n)*** computes interference matrix in every iteration of algorithm associated to equation (3). Algorithm ***<ComputeIn>*** (see Fig. 3) implements interference function, using MIBS method, to get the interference matrix.

Interference matrix is computed row by row.

computed using algorithm ***<$S_\ell$inS$_i$>***. Given any orderly pair of sequences, i.e. S_ℓ y S_i, ***<$S_\ell$inS$_i$>*** adds in row ***i*** a vector with maximum interference of S_ℓ with S_i. This vector contains the delay due to S_ℓ, distributed over tasks of S_i, when releases of S_i respond with WCRT.
New ***In*** does not directly replace the interference matrix computed in preceding iteration. Before doing so, it is modified as shown by line 7 in Fig. 3. This ensures convergence of succession $R^1_{i,j}$, $R^2_{i,j}$, $R^3_{i,j}$, … for any element of ***R***, so convergence of the process associated to equation (3) is guaranteed.

<ComputeIn>

Matrices contain system parameters:

E	Maximum task executing times.
e	Minimum task executing times.
B	Maximum blocking times.
N	Task processing node numbers.
Pri	Task priorities inside their nodes.
P	Sequence periods (column vector).
D	Sequence deadlines (column vector).

Output:
In Interference matrix. ***In***$_{i,j}$ is interference (delay) in $T_{i,j}$ due to higher priority tasks when any release of S_i has WCRT.

```
1.  PreviousIn ← In
2.  <write 0 to In elements>
3.  FOR i ← 1: number of sequences
4.     FOR l ← 1: number of sequences
5.        IF l ≠ k
6.           <SℓinSi>
          ENDIF
       ENDFOR
    ENDFOR
7.  In_{i,j} ← max(PreviousIn_{i,j}, In_{i,j}),
                  for elements In_{i,j} in In.
```

Fig. 3. This algorithm computes interference matrix using MIBS methods (variables in **bold-face** are matrices).

7. Interference between sequences.

<$S_\ell$inS$_i$> (Fig. 10), computes the interference of S_ℓ with S_i looking for a ***critical behaviour of S_ℓ on S_i***. To do so the concept of sequence ***executions*** is introduced. A ***execution of*** S_ℓ ***on S_i*** is any possible behaviour of both sequences considering what tasks of S_ℓ delays tasks in S_i.

The algorithm analyses all possible executions of S_ℓ on S_i that may have maximum interference with S_i: that is, it analyses all possible executions that may be critical behaviours.
If interference of all these executions is computed, we just have to look for the maximum to get the maximum interference vector of S_ℓ with S_i.

Obviously, there are a huge number of different executions Behaviour of any execution depends on sequence release times, interactions with other sequences, task execution times in both sequences, etc. Given any pair of sequences, it can be null-interference executions (Fig. 4.A), or non-null interference executions, with interference taking place in different tasks and different moments (Fig. 4.B and C)

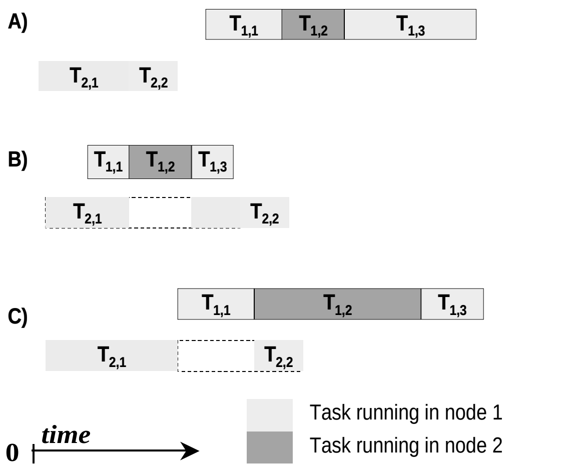


Fig. 4.Sample executions of S_1 on S_2 in a system with two processing nodes and two sequences. To simplify graphics, we assumed all tasks in S_1 having higher priority than tasks in S_2. Assuming both sequence periods much higher than WCRT saves us from draw multiple cycle executions.

So, it is unfeasible to compute interference for all possible executions except for very simple sequences.
<$S_\ell$inS$_i$> considers some properties of executions with critical behaviour to drastically reduce the executions to analyse.
The execution with critical behaviour in Fig. 5 can be found applying the algorithm to sequences in Fig. 4.

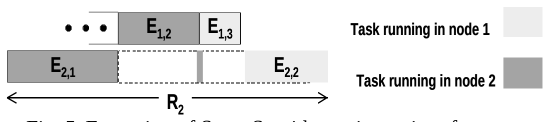


Fig. 5. Execution of S_1 on S_2 with maximum interference.

In this execution, the release of S_2 has WCRT because tasks in both sequences execute for maximum time ($E_{i,j}$), and $T_{1,2}$ is released just before the end of $T_{2,1}$. Next task in S_2, $T_{2,2}$, is delayed again by a task in S_1, $T_{1,3}$.
This is one of the many possible executions of S_1 on S_2 with maximum interference. They differ in execution of $T_{1,1}$, that does not affect the delays of S_1 in S_2.

8. Conditional interference.

This section uses sequences in Fig. 5 to illustrate how algorithm ***<S$_\ell$inS$_i$>*** works.

<S$_\ell$inS$_i$> processes S_2 task by task. It looks for some characteristics of execution of S_1 interfering as much as possible with $T_{2,1}$. Any execution of S_1 where $T_{1,2}$ runs for $E_{1,2}$ produces maximum delay to $T_{2,1}$. There are many executions behaving this way. Some of then are shown in Fig. 6.

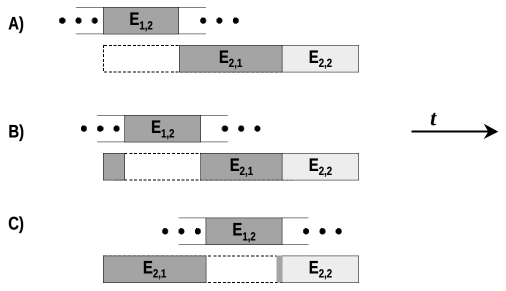


Fig. 6. Executions of S_1 producing maximum delay to $T_{1,2}$.

In general we can always categorise any non-null interference execution of S_ℓ on S_i attending to:

- What is the first task in S_ℓ interfering with any other in S_i.
- What is the fist task on S_i interfered by that task in S_i.

We will define the concept of ***conditional execution on <T$_{l,k}$ delays T$_{i,j}$>*** as any execution of S_ℓ on S_i satisfying condition <$T_{l,k}$ is first task in S_ℓ delaying S_i, and this delay starts in $T_{i,j}$>.

This concept allows us to group non-null interference executions of S_ℓ on S_i into families of conditional executions since there is one condition for every task pair ($T_{l,k}$, $T_{i,j}$) running in the same node with $T_{l,k}$ having higher priority than $T_{i,j}$.

We will define ***conditional interference on <T$_{l,k}$ delays T$_{i,j}$>*** as the maximum interference of any conditional execution on ***<T$_{l,k}$ delays T$_{i,j}$>***.

Fig. 6 shows examples of conditional executions on <$T_{1,2}$ delays $T_{2,1}$> depicted up to the end of task $T_{2,1}$. If we look for interference until that time, all examples have maximum interference in their family.

The importance of conditional interference comes from the fact that executions of S_ℓ on S_i with critical behaviour are conditional executions, but it is a priori unknown what task pair forms the condition. However, if we can find all the different families of conditional executions and their associated conditional interference, the maximum interference of S_ℓ with S_i will be the maximum value of this set.

All the executions in Fig. 6 delay to $T_{1,2}$ in the they do not have to be equivalent for tasks following $T_{1,2}$. In fact, in execution for Fig. 6.C, $T_{2,2}$ is delayed, while it is not in executions A and B ($T_{1,3}$ ends before $T_{2,2}$ begins)

Effects of conditional executions in tasks following $T_{2,1}$ can not yet be established while analysing $T_{2,1}$. So, ***<S$_\ell$inS$_i$>*** saves information about conditional executions that will be used later, in tasks following $T_{2,1}$ (Fig. 7).

This information is saved as elements of the set ***{CondID}***. They describe executions that could have maximum interference.

As ***<S$_\ell$inS$_i$>*** processes S_i task by task, starting from the first on, given a task $T_{i,j}$, the information available covers execution behaviours up to this task. The elements of ***{CondID}*** contain two kind of information:

- Interference vector. It stores conditional interference from $T_{i,1}$ to $T_{i,j}$.
- Phase information. It includes what task of S_ℓ last delays $T_{i,j}$ and what is the interval such a S_ℓ task ends, in any execution from the described family (Fig. 7). This information will be used later to find out executions of this family that can interfere with tasks following $T_{i,j}$.

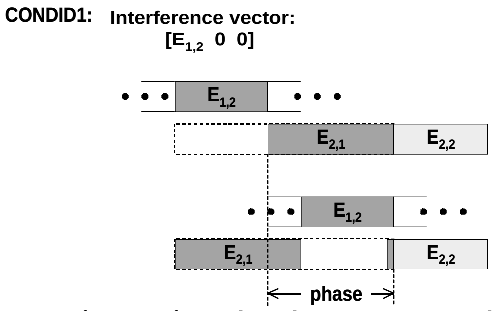


Fig. 7. Information for conditional executions on <$T_{1,2}$ delays to $T_{2,1}$> contains interference until $T_{2,1}$ and phase information.

For every task $T_{i,j}$ of S_i, the algorithm ***<S$_\ell$inS$_i$>***:

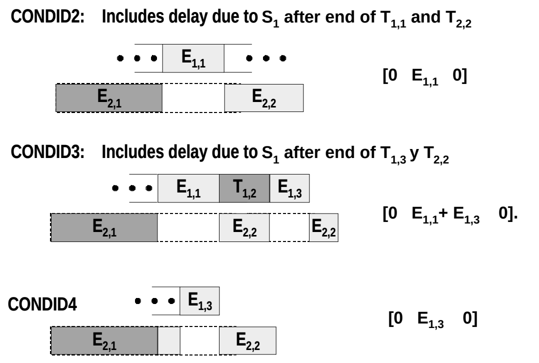
P1. Creates a description of conditional interference for the task of S_ℓ that can delay $T_{i,j}$.

P2. Analyses the elements in ***{CondID}*** checking for execution that can interfere with $T_{i,j}$.

Returning to our example, process P1 creates only one description of conditional interference for $T_{2,1}$ (CONDID1). For $T_{2,2}$, process P1 creates three new descriptions: CONDID2, CONDID3[3] and CONDID4 (Fig. 8). Then, ***<S$_\ell$inS$_i$>*** uses process P2 with elements of ***{CondID}*** (in this case CONDID1).

[3] CONDID2 and CONDID3 are in family of conditional executions on <$T_{1,1}$ delays $T_{2,2}$>. However, after $T_{2,2}$ ends, executions described by CONDID2 can behave in a completely

The result can be new elements containing conditional interference descriptions of the executions described by preceding elements that can delay $T_{2,2}$ (Fig. 9).
Every element of ***{CondID}*** has an interference vector containing interference from $T_{i,1}$ up to the task in process when the element was created. After the last task of S_i is analysed, the algorithm adds to row ***i*** of interference matrix the vector with maximum interference (line 9 in Fig. 10).



CONDID2: Includes delay due to S_1 after end of $T_{1,1}$ and $T_{2,2}$
$E_{1,1}$
$E_{2,1}$
$E_{2,2}$
[0 $E_{1,1}$ 0]
CONDID3: Includes delay due to S_1 after end of $T_{1,3}$ y $T_{2,2}$
$E_{1,1}$
$T_{1,2}$
$E_{1,3}$
$E_{2,1}$
$E_{2,2}$
$E_{2,2}$
[0 $E_{1,1}$+ $E_{1,3}$ 0].
CONDID4
$E_{1,3}$
$E_{2,1}$
$E_{2,2}$
[0 $E_{1,3}$ 0]

Fig. 8. Conditional interference descriptions created for $T_{2,1}$.

The element of ***{CondID}*** containing this vector is a description of the executions of S_ℓ on S_i with critical behaviour.

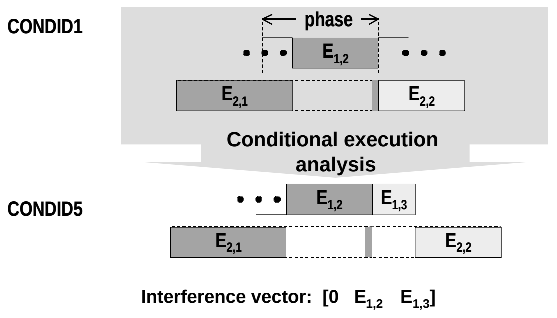


Fig. 9.Process P2 analyses elements in *{CondID}* to generate new elements if executions described can delay $T_{2,2}$.

<$S_\ell$enS$_i$>
Inputs:
S_ℓ Sequence source of delay.
S_i Delayed sequence.

Output:
Adds to row ***i*** of ***In*** maximum interference of S_ℓ with S_i.

```
1.  {CondID} ← { }
2.  FOR j ← 1: last task in S_i
3.     {NewsCondID} ← { }
4.     FOR every task T_l,k  {H(T_i,j)}
5.        <P1:adds els. to {NewsCondID}>
      ENDFOR
6.     FOR every CondIDElement   {CondID}
7.        <P2:analyses it, adding ressults
                 to {NewsCondID}>
      ENDFOR
8.     {CondID} ← {CondID}   {NewsCondID}
   ENDFOR
9.  In_i  ←In_i  + <interference vector in
         {CondID}  containing  maximum
         interference>
```

Fig. 10. Maximum interference of a sequence with other.

9. Sample application: DXBus based wheelchair controller.

As sample application, we apply MIBS method to a DXBus based wheelchair developed at the University of Seville, which we call DXSIR (Fig. 11).

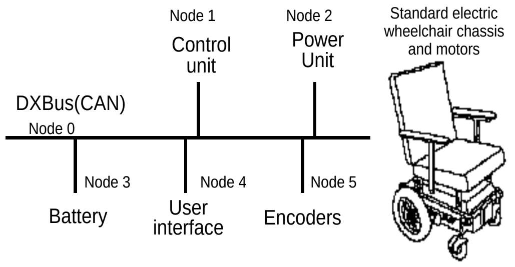


Fig. 11. DXSIR block diagram.

DXBus is a four wire communications system (two power lines and two data lines, CANL and CANH) with a maximum length of 15 meters and a transmission rate of 2/19 Mbits/s. Basically, DXBus uses CAN protocol with some extensions, i. e. special voltage values on the data lines are used for "emergency stops" or power up.
CAN identifiers are assigned to modules and not to messages. As a result, 1 byte in the message data field must be used to indicate the message type, and up to 7 bytes for message data.
During normal operation, DX modules communicate using ***Network Variables*** (NVs). NVs are transmitted periodically (during time slot), with period either 20ms (fast NV) or 200ms (slow NV).
A more detailed description of DXBus can be obtained from [10]. In this paper, we focus on the processes associated to DXSIR functions.
DXSIR is a wheelchair with two driver wheels

The DXSIR controller is made up of five nodes connected through DXBus (Fig. 11).

In this simple prototype, the User interface is reduced to a joystick and some buttons. We are particularly concerned with people with uncontrolled movements or tremor (Parkinson, cerebral palsy). In slight cases, we simply filter by software the joystick input eliminating unintentional movements [1].

The Control Unit performs a closed-loop automatic control that converts the joystick command into linear and turning velocity. It also monitors the system bus, battery, etc. and implements several navigational aids including fixed trajectories (straight ahead or back, 90 degrees turn) and play back of recorded trajectories. The latter avoids the difficulties of backwards driving, and may be very useful in small areas like bathrooms. It has been implemented by tracking the memorised path following [7] which requires the accurate determination of internal coordinates. DXSIR gets them through two optical encoders at each drive wheel that provides rotation angle increments $\Delta\theta_R$ and $\Delta\theta_L$. This information is also needed as feedback for the closed-loop control, but it only requires information about the present state. On the other hand, memorisation of the performed trajectory needs information along a period of time. Besides, accuracy is important because small errors produce great trajectory deviations. As a result, it is recovery of performed trajectories what imposes the maximum period between messages from the Encoders to the Control Unit. Actually, bus load depends mainly on this traffic.

Díaz [6] shows that for sampling error to be negligible with trajectories of a few tens of meters (the maximum reasonable trajectory length), a new pair of encoders should be saved when $[(\Delta\theta_R)^2+(\Delta\theta_L)^2] \geq 800$ e.u., where e.u. means wheelchair encoders units (4000 e.u. are equivalent to 2π rad or 1 meter of linear advance). Anyway, the maximum frequency occurs with maximum (linear) velocities, that is, $\dot{e}_R = \dot{e}_L > 0$. If we assume a maximum velocity of 1.5 ms^{-1} (probably too high in practice), then the sample period would be 5ms. This sample frequency is too high for the DXBus, but we can use an incremental codification of the encoders, so that encoder increments are represented with 7 bits for each wheel. Since the data field of DX packets is 7 bytes, we can send 4 pairs of encoder values every 20ms (fast NVs).

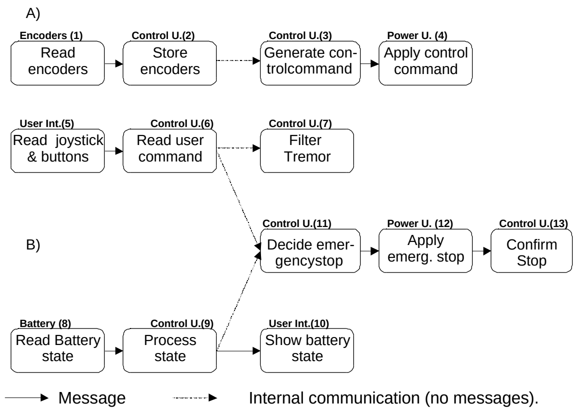


Fig. 12. Processes implementing DXSIR main functions.

Fig. 12 shows the DXSIR processes with the tasks precedence constrains. When an emergency stop occurs, a new process is scheduled in parallel with both the process that interprets (and filters) user commands and the battery gauge process. Thus, in order to include these ramifications in our model we need to define the concept of ***"false replica"*** of a task.

They exist only in the model and are included to remove ramifications while maintaining precedence constrains. If T'i,j is a false replica of Tl,k then:

· $B'_{i,j} = B_{l,k} + E_{l,k}$.

· It is executed in the same node and with same priority than $T_{l,k}$. This allows $R_{i,j}$ to be computed using the same method as $R_{l,k}$.

· $E'_{i,j} = 0$. $T'_{i,j}$ does not really exist, it must not interfere with lower priority tasks.

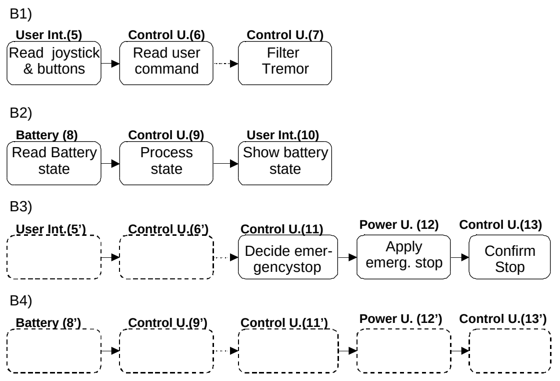


Fig. 13. Sequences modelling the processes in Fig. 12.B.

Table II lists all DXSIR tasks and their parameters. Task priorities have been assigned using EDM (Effective-Deadline Monotonic) technique [9].

Table II. DXSIR model parameters.

Tasks	***P_i &D_i (ms)***	***$E_{i,j}$ (ms)***	***$e_{i,j}$ (ms)***	***$B_{i,j}$ (ms)***	***Node & pri.***
$T_{1,1}$: Read encoders	20	1	0,5	0	***5,*** 1
$T_{1,2}$: Encoder values		1.23	1.23	0.69	***0,*** 1
$T_{1,3}$: Store encoders & compute control cmd.		5	2	0	***1,*** 1
$T_{1,4}$: Control command		0.69	0.69	0.69	***0,*** 2
$T_{1,5}$: Apply control cmd.		1	0,5	0	***2,***1
$T_{2,1}$: Read user interface	20	4	1	0	***4,*** 1
$T_{2,2}$: Interface values		0.69	0.69	0.69	***0,*** 3
$T_{2,3}$: Read user cmd.		1	0,5	0	***1,*** 2
$T_{2,4}$: Filter tremor		6	4	0	***1,*** 3
$T_{3,1}$: Read batt. status	200	1	0,5	0	***3,*** 1
$T_{3,2}$: Battery status		0.69	0.69	0.69	***0,*** 6
$T_{3,3}$: Process status		2	1	0	***1,*** 5
$T_{3,4}$: Processed status		0.69	0.69	0.69	***0,*** 4
$T_{3,5}$: Show batt. status		1	0,5	0	***4,*** 2
$T_{4,1}$: False replica of $T_{2,1}$	200	0	1	4	***4,*** 1
$T_{4,2}$: False replica of $T_{2,2}$		0	0.69	1.38	***0,*** 3
$T_{4,3}$: False replica of $T_{2,3}$		0	0,5	1	***1,*** 2
$T_{4,4}$: Decide emerg. stop		3	2	0	***1,*** 4
$T_{4,5}$: Emergency stop		0.69	0.69	0.69	***0,*** 4
$T_{4,6}$: Apply emerg. stop		5	2	0	***2,***2
$T_{4,7}$: Emerg. stop ACK		0.69	0.69	0.69	***0,*** 5
$T_{4,8}$: Read ACK		6	2	0	***1,*** 6
$T_{5,1}$: False replica of $T_{3,1}$	200	0	0,5	1	***3,*** 1
$T_{5,2}$: False replica of $T_{3,2}$		0	0.69	1.38	***0,*** 6
$T_{5,3}$: False replica of $T_{3,3}$		0	1	2	***1,*** 5
$T_{5,4}$: False replica of $T_{4,4}$		0	2	3	***1,*** 4
$T_{5,5}$: False replica of $T_{4,5}$		0	0.69	1.38	***0,*** 4
$T_{5,6}$: False replica of $T_{4,6}$		0	2	5	***4,*** 2
$T_{5,7}$: False replica of $T_{4,7}$		0	0.69	1.38	***0,*** 5
$T_{5,8}$: False replica of $T_{4,8}$		0	2	6	***1,*** 6

There are two possible message transmission times (for tasks in node 0), 0.69 ms for 2 bytes and 1.23 ms for 7 bytes of data (assuming 105.3 Kits/sec DXbus). In priority column, lower numeric values mean higher priority

Tables III and IV summarise the sequence WCRT estimations using both Rate Monotonic Analysis based method and MIBS method.

We can see that both models are equivalent for sequence 1, since it contains highest priority tasks so they do not suffer from interference. For the rest of sequences MIBS estimations are more accurate. Note that example S_2 will be predicted as non-schedulable if RMA is used to estimate WCRT. However, MIBS estimates a WCRT lower than its deadline as it detects that $T_{1,2}$ and $T_{1,4}$ have precedence constrains, and they cannot simultaneously interfere with $T_{2,2}$, while RMA includes interference from both tasks.

Table III. Sequence WCRT estimated with RMA method.

	Response time: Task (msecs)								***Seqs.***
Seq.	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***R_i***
1	1	1,92	5	1,38	1				10,31
2	**4**	**3,31**	**6**	**11**					**24,31**
3	1	5,39	36	5,39	5				52,78
4	4	3,31	6	27	4	6	4	44	98,32
5	1	5,39	36	27	4	6	4	44	127,4

Table IV. Sequence WCRT estimated with MIBS method.

	Response time: Task (msecs)								***Seqs.***
Seq.	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***R_i***
1	1	1,92	5	1,38	1				10,31
2	4	2,62	6	6					18,62
3	1	3,31	29	3,31	5				41,62
4	4	2,62	7	21	2,62	6	2,62	44	89,86
5	1	3,31	28	19	2,07	6	3,31	44	106,70

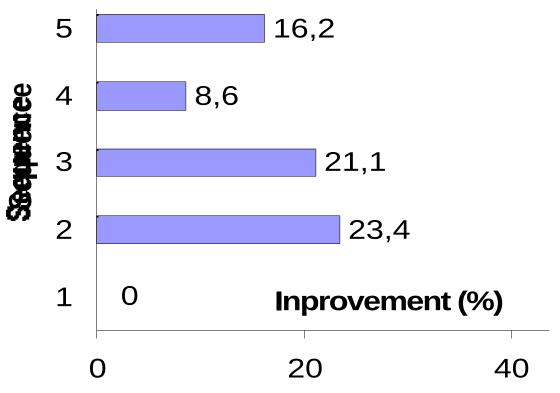


Fig. 14Improvement of sequence WRCT estimations from MIBS method over estimations from RMA method.

10. Conclusions.

In this paper, a systematic approach to precedence analysis is presented. The MIBS analysis considers the interference between sequences instead of between single tasks. When applied to a DXBus based wheelchair controller, we show that predictions of the WCRT significantly improve. The mean improvement in MIBS estimations as compared with RMA estimation is about 14% (Fig. 14). When applied to other systems, the mean improvement is 13-18% [2] although it is higher for sequences with a high number of tasks with relatively low priorities in their nodes.

11. References.

[1] A. Civit-Balcells, M. A. Rodríguez, C. Amaya, F. Díaz del Río, L. Miró, J.L. Sevillano: "A system for the analysis and scanning of tremor on handicapped people". In *Assistive Technology on the threshold of the new millenium.* Eds. C. Bühler and H. Knops. pp. 539-544. IOS Press, Netherlands. 1999.

[2] Amaya Rodríguez, C. "Análisis y evaluación de la planificación de sistemas distribuidos de tiempo real. Aplicación a tecnologías de la rehabilitación". PhD Thesis. University of Sevilla. December, 1999. It is avaible in http://icaro.fie.us.es/~claudio/tesis.html.

[3] Amaya, C., et al. "Análisis de un sistema de tiempo real duro distribuido". X Jornadas de Paralelismo, Murcia, septiembre, 1999.

[4] Audsley, N. C., Burns, A., Richardson, M. F., Wellings, A. J. "Hard Real_Time Scheduling: The Deadline Monotonic Approach". Proceedings 8th IEEE Workshop on Real_Time Operating Systems and Software, May 1991

[5] Audsley, N. C., Burns, A., Richardson, M., Tindell, K.,

[6] Díaz del Río, F. *"Análisis y evaluación del control de un robot móvil: aplicación a sillas de ruedas eléctricas"*. PhD Thesis. University of Sevilla. 1997.

[7] F. Díaz del Río, G. Jiménez, J. L. Sevillano, S. Vicente, A. Civit-Balcells, "A Generalization of Path Following for Mobile Robots". Proc. IEEE Int. Conf. Robotics and Autom, pp. 7-12. Detroit, MI (USA). 1999.

[8] Joseph, M., Pandya, P. *"Finding Response Times in a Real_Time System"* BCS Computer Journal, Oct 86.

[9] Kao, B., García-Molina, H. *"Deadline Assignement in a Distributed Soft Real-Time System"*. IEEE Transaction on Parallel and Distributed Systems, Dec. 1997.

[10] Meade, M. "DX Key Technical Description. For DX Key Application Designers". Dynamic Controls Ltd. 1997.

[11] Sun, J. *"Fixed Priority End-to-End Scheduling in Distributed Real-Time Systems"*. Ph.D Thesis in Computer Science, University of Illinois, Urbana-Champaign, 1997.

[12] Sun, J., Gardner, M.K., Liu, J.W.S. *"Bounding Completion Times of Jobs with Arbitrary Release Times, Variable Execution Times, and Resource Sharing"*. IEEE Transactions on Software Engineering, Oct. 1997.

[13] T.-Y. Yen, W. Wolf. *"Performance estimation for real-time distributed embedded systems"*. IEEE Transactions on Parallel and Distributed Systems, Nov 1998.

[14] Tindell, K., Clark, J. *"Holistic schedulability analysis for distributed hard real_time systems"* Microprocessing and Microprogramming, April 1994.

Company: Universidad de Sevilla. Development group: Robotic and Computing for Assistive Technologies.

Address: Facultad de Informatica. Avda. Reina Mercedes, s/n. 41012 Sevilla (Spain).

Phone: +34 954 55 27 79 Fax +34 954 55 27 59

e-mail: claudio@atc.us.es **web**: icaro.fie.us.es.