7.7 Bus Schedule

For each module in the system, we have identified the required messages and mapped them to frames, but the communication windows of different frames could overlap. Therefore we collect the frames required by all modules and apply on this global set a variation of the Reversed EDF scheduling algorithm, that is, the Latest Release Time (LRT) [4].

This decides when each frame must be sent on the network, depending on the release, deadline and worst case transmission time of each frame. Furthermore, the bus scheduler might have additional constraints that result from the physical properties of the communication infrastructure. For example, it includes gaps in the schedule, because it has to align the sending time according to the inter frame gaps and the clock resolution on the computing nodes. The bus scheduler might also generate extra frames, for example for time synchronization. Furthermore, it might merge adjacent frames in the sorted list of frames if they are sent by the same node. This leads to the remapping of the corresponding messages to the merged frame.

8 Case Study

The case study comprises one module with two modes that provides the results of its computations to at least one other module on another node. It illustrates step-by-step the algorithm presented in the previous section.

Specification of module M

For the case study we assume a module M which runs on a node that is part of a distributed system. It provides the values produced by its tasks t1, t2, and t3 to other modules. The module has two modes in which it executes different tasks.

Mode	Mode period	Tasks with their LETs in brackets
m1	5 ms	t1 (5 ms)
m2	10 ms	t2 (10 ms), t3 (10 ms)

List of Tasks

The table lists all tasks with their worst-case execution time (WCET) and the size of their output values.

Name	WCET	Output Size
t1	1 ms	4 byte
t2	1 ms	2 byte
t3	1 ms	2 byte

Bus Period Calculation

Above we defined mspGCD_M as the GCD (greatest common divisor) of mode periods and mode switch periods in all modes in module M. In our case study the mode switch period equals the mode period in all modes, as otherwise the mode switches would not be harmonic, meaning that a mode switch must not occur during the LET of every task invocation of the currently active mode.

$\text{mspGCD}_M = \text{GCD}(5, 10) = 5$ ms

The bus period of the generated bus schedule is computed as the GCD of the mspGCD_M of each module M which communicates on the bus. As we have only one such module, the bus period equals mspGCD_M.

List of Messages

The following table contains a list of all messages that have to be sent by the tasks in all modes. The release and deadline times are relative to the mode period of the task that sends the message. The release time is equal to the beginning of the LET plus the WCET, the deadline time is equal to the end of LET.

Message	Mode	Task	Invocation	Size	Release	Deadline
1	m1	t1	1	4	1 ms	5 ms
2	m2	t2	1	2	1 ms	10 ms
3	m2	t3	1	2	1 ms	10 ms

For the assignment of messages to frames we need the release and deadline constraints relative to the bus period for all phases of the two modes:

Message	Mode	Task	Invocation	Size	Phase	Release	Deadline
1	m1	t1	1	4	1	1 ms	5 ms
2	m2	t2	1	2	2	0 ms	5 ms
3	m2	t3	1	2	2	0 ms	5 ms

Frame Assignment

According to the algorithm we iterate over all modes and phases and check if we assign messages to existing frames or if we create new ones. We start with mode m1 and phase 1, which is the only phase in this mode. As we have no frames at the beginning, we create one that inherits the timing constraints from the first message. The frame size equals to the message size plus 1 byte for the tag.

Frame	Size	Release	Deadline	Bound Messages
1	5	1 ms	5 ms	1

Next we consider mode m2 and as there is nothing to do in phase 1 we proceed to message 2 in phase 2. Here we need to evaluate the metric in order to determine whether to bind the message to the existing frame. In this example we use a threshold value of 0.5. Note that in every new phase the available space of every frame is reset to its size. For this message the overlapping metric is:

$$overlapping = Min(5,5) - Max(1,0) = 4$$

$$metric_{overlapping} = \frac{\frac{4}{5-1} + \frac{4}{5-0}}{2} = \frac{1 + \frac{4}{5}}{2} = 0.9$$

As the message size plus tag is 3 bytes and therefore smaller than the available frame space, we do not have to enlarge the frame at all which consequently yields 1 for the enlargement metric. This yields 0.95 for the overall metric and so message 2 gets bound to frame 1, reducing the available space in this phase to 2 bytes.

Frame	Size	Release	Deadline	Bound Messages
1	5	1 ms	5 ms	1, 2

The next message in this phase is message 3. As the timing requirements are the same as for message 2, we get again 0.90 for the overlapping metric.

For the enlargement metric we have msg.size (including tag)= 3, frame.available = 2 and frame.size = 5 and get:

$$enlargement = Max(0,(3-2)) = 1$$

$$metric_{enlargement} = \frac{5}{5+1} = 0.83$$

The overall metric then is 0.87 which is above our threshold and so this message also gets bound to frame 1, increasing its size to 6 bytes:

Frame	Size	Release	Deadline	Bound Messages
1	6	1 ms	5 ms	1, 2, 3

Bus Schedule Generation

For the identified frame a suitable bus schedule based on the release and deadline constraints must be generated. As described above this is done using the LRT (Latest Release Time) scheduling algorithm. This results in a bus schedule where frame 1 is scheduled so that its transmission ends at 5 ms.

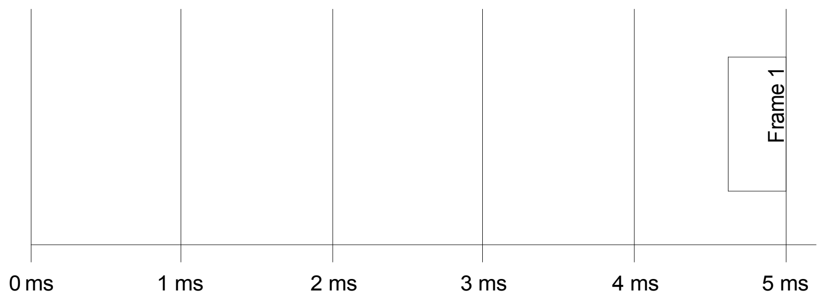

Fig 10. Location of frames inside the bus period

Figure 11 shows the dynamic multiplexing of messages in frame 1 for both modes of module M. Note that dynamic multiplexing allows us to fill the frame which has a capacity of 6 bytes either with message 1 with its size of 5 bytes, or with messages 2 and 3 with their size of 3 bytes each, or to leave the frame empty.

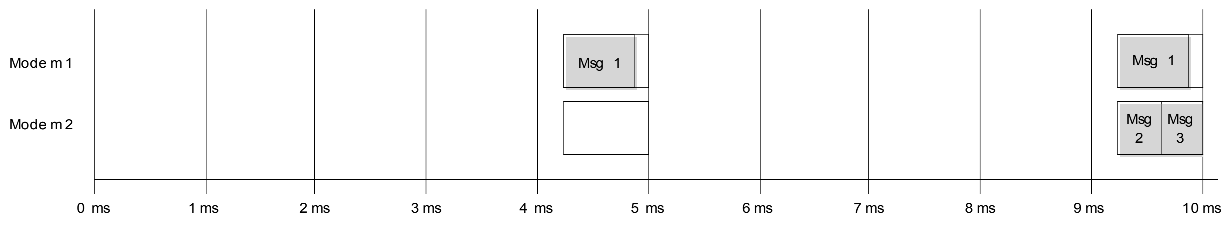

Fig 11. Dynamic multiplexing of messages

Static multiplexing would require the sending of an additional frame that has a capacity of 3 bytes. So the communication medium is used less efficiently: 5 bytes + 3 bytes are sent instead of 6 bytes.

References

[1] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the First International Workshop on Embedded Software* (EMSOFT), Lecture Notes in Computer Science 2211, Springer-Verlag, 2001, pp. 166-184.

[2] J. Templ, 2004, TDL Specification and Report. Technical Report, Department of Computer Science, University of Salzburg, http://www.cs.uni-salzburg.at/pubs/reports/T004.pdf

[3] H. Kopetz, 1997, Real-time Systems: Design Principles for Distributed Embedded Applications. Kluwer, 1997

[4] Jane W. S. Liu. Real-Time Systems. Prentice-Hall, 2000