Adding Timing Requirements to the CODARTS Real-Time Software Design Method

Bach, K.R.

Publication date:
1999

Document Version
Også kaldet Forlagets PDF

Adding Timing Requirements to the CODARTS Real-Time Software Design Method

Keld Ramstedt P. Bach - *keld@control.auc.dk*

***Abstract*— The CODARTS software design method consideres how concurrent, distributed and real-time applications can be designed. Although accounting for the important issues of task and communication, the method does not provide means for expressing the timeliness of the tasks and communication directly in the design as is otherwise the case with tasks and communication specifics. In this paper we propose an extension scheme which will enable for specifying timing requirements for tasks and communications within the CODARTS model.**

***Keywords*— Computer system design, real-time systems, timing specification.**

I. Introduction

With the immense growth in the use of real-time systems, there is also a need to express and design the functionality of these at an abstract level since the systems becomes more and more complex. A number of software design methods exists to accomplish this issue. [1] gives an elaborate discussion of various design methods and concludes with developing the CODARTS model.

The contribution of the CODARTS model to the software design phase, is mainly the idea of task structuring and details about communication e.g., if asynchronous buffers etc. should be used.

The CODARTS model describes task structuring and communication in concurrent, distributed, and real-time systems but does not address the issue of direct timely specification in the design phase.

The purpose of this paper is to introduce some methods to help specifying and verifying timely behavior of tasks and communication in the CODARTS scheme.

II. CODARTS

Dept. of Control Engineering, Aalborg University, 9220 Aalborg East, Denmark

When designing real-time systems using the CODARTS model, the following seven steps must be applied:

1. Develop environmental model. This is based on the COBRA method for analysing and modeling the problem domain.
2. Structure the system into distributed sub-systems. This step is only necessary if the system to be developed has a distributed nature.
3. Structure the system or sub-systems into concurrent tasks.
4. Structure the system into information hiding modules.
5. Integrate task and modules views.
6. Define component interface specifications.
7. Develop software incrementally.

The above roughly ends up first with a task architecture diagram and then finally the system architecture diagrams (SADs) for both the overall system and for the subsystems which are then the actual graphical designs. The SAD is constructed graphically using the approaches described in figure 1.

As an example, consider figure 2 of a software architecture diagram for a robot controler system:

As it can be seen, no indication of real-time specifications are given explicitly. Although guidelines for modeling tasks and communication are present, no method of specifying real-time behavior for these in the design are addressed.

For a more elaborate discussion of the CODARTS method and the necessary steps, please refer to [1].

III. CODARTS+

We have seen that CODARTS does not provide means for expressing real-time requirements in the design. In order to do this, we must first look at

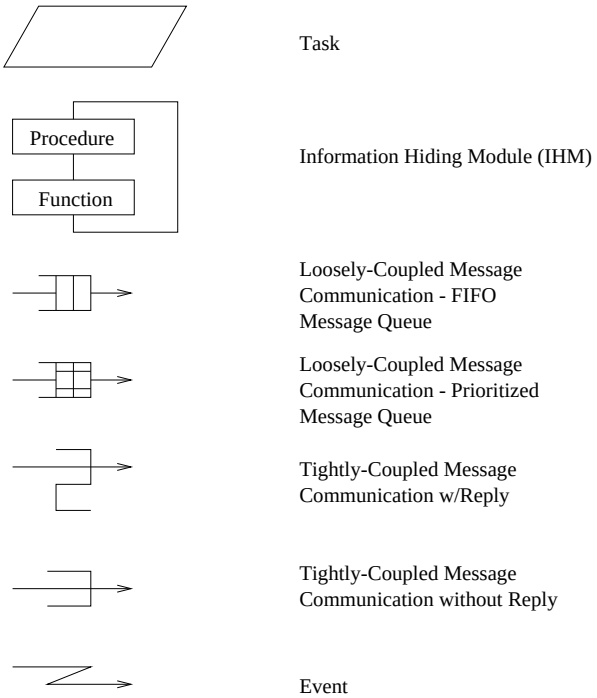

Fig. 1. Basics of SAD's

which timely requirements are needed for the task and communication entitites.

In real-time systems, tasks and communication may have timely behavior. In fact, they nearl always do. Dependent on the type of real-time system, all or some of the following issues may be applied to tasks:

- The tasks' needed CPU resources. In order to assess the period of a task, one must also define the CPU resources needed for a given task. The CPU resources are the total amount of computation time needed to finish a specific task, which may also include communication overhead times, context switch times, and interrupt handler times.
- The tasks' periodicity. This is the regularity by which the task must be executed. This is a constant factor, i.e., it doesn't vary over time since the intention with having hard-real-time system is responsiveness and predictability. To enforce these issues, entities are typically sampled periodically and tasks are also invoked periodically. If a tasks' period changes then it becomes difficult to predict the behavior. Also, per definition, a periodic occurrence *is* in fact an event occurring again and again, at constant intervals.

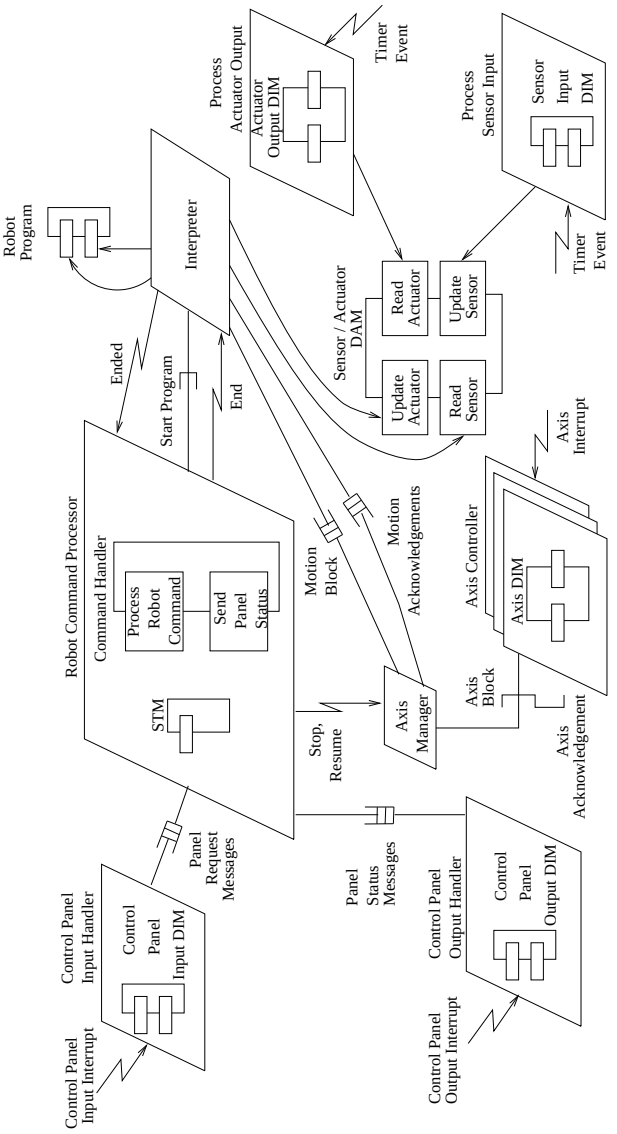

Fig. 2. Software Architecture Diagram for Robot Control System

Although aperiodic events may occur in hard real-time systems, these events may be served by periodic sporadic servers.

- The tasks' deadlines. In some periodic systems, the deadlines for a task for which it must complete, lies implicitly in the periodicity of the task. I.e., in order to maintain the periodicity of executing the task again and again, the task of course must have a deadline that are no later than the next scheduled execution of the same task. However, in some situations the task may have a specified deadline that is earlier or later than the end of its period.
- The tasks' release times. The release times of a task is typically also the start of its period.

However, it might be the case that a task has to wait for a certain event to occur (within some deadline of course), before it can spin off.

- Penalty value for tasks missing deadlines. The values may be all that is measurable, e.g., actual time compared to predefined values of missing a deadline. The higher the value, the bigger penalty is likely to occur to a given task. Hence, lower priority tasks are assigned lower penalty values. This applies to schedulers using, e.g., the best-effort technique where the tasks with the highest penalty values are scheduled first in order to avoid the penalty using an EDF strategy [2].

For communication, the following may be used:

- Response times. This is the maximum reponse time for a message to be acknowledged by the receipient, i.e., maximum transfer time for a message.
- Number of messages in a queue. By assigning maximum number of messages in a queue (queue length), it is possible to predict worst-case message handling times for messages. In addition, if this number is exceeded when messages are buffered then something may be wrong somewhere in the system, e.g., either the sending task is submitting messages too fast, or the receiving task is admitting messages too slow.

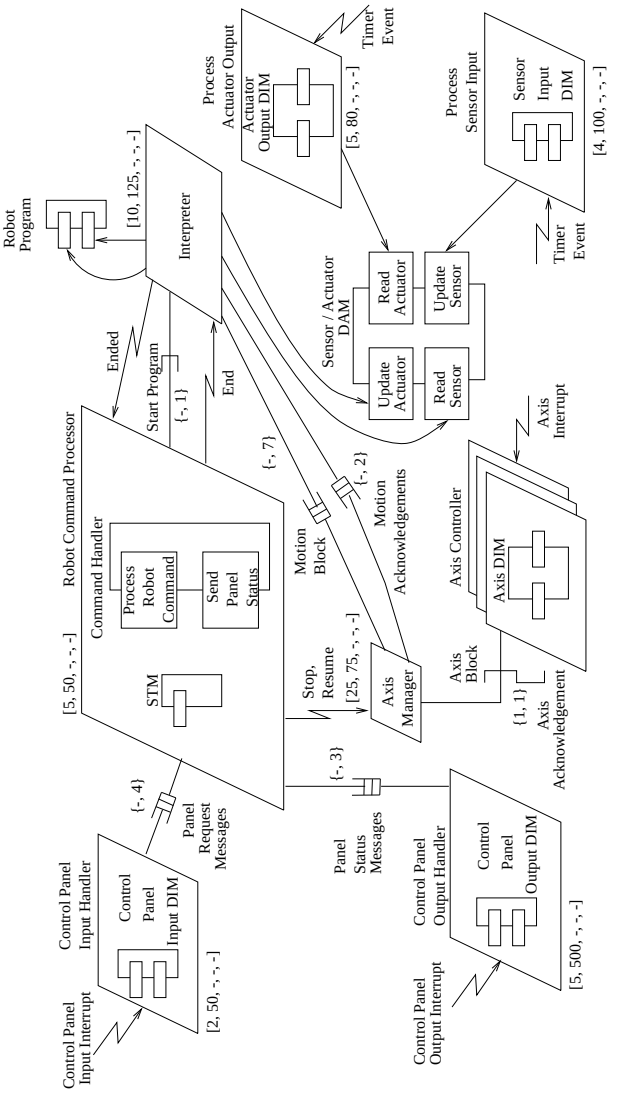

Fig. 3. Software Architecture Diagram for Robot Control System with Real-Time Specification Tuples

In our model we enhance the CODARTS model with tuples to signify the needed real-time requirements. For tasks we need a quintuple denoting the above mentioned issues. For communication we need a duple. For convenience we will attach these tuples to the associated graphic entities in CODARTS as depicted on figure 3.

The following shows a few examples of the use of tuples:

1. [25, 200, -, -, -]. Here, the tuple denotes a task with 25 time units of CPU resource claims and a period of 200 units.
2. [17, 350, 200, -, -]. This is a tuple where the associated task has 17 time units of CPU requirements, a period of 350, and a deadline of 200 time units after the task has commenced its execution. In the previous example 1, the deadline was defined implicit in the period, i.e., deadline and period were equal.
3. [7, 100, -, 10, -]. The connotation of this is a task having a CPU requirement of 7 time units, a period (and deadline) of 100, and release time of 10. Hence, the task must be commenced within 10 time units from the start of its periods.
4. [55, 375, -, -, 65]. Here, a task has CPU requirements of 55 time units, a period of 375, and a penalty of missing that deadline of 65. The penalty value is used in, e.g., the best-effort approach.

For communication the tuples will be:

1. $\{5, -\}$. This tuple is attached to message with the requirement that that messages be delivered within 5 time units. This could be applied to messages with reply.
2. $\{-, 9\}$. Here a queue may only hold maximum 9 messages at a time. No timing requirements are present.
3. $\{8, 4\}$. This is associated with a message queue. It says that a message must take no longer than 8 time units while in transfer from one task to another, and that the queue may only hold up to 4 messages at a time. This corresponds to multiple messages with replys.

The tuples may in the final implementation of an actual design program based on CODARTS be hidden so that they only appears when the entities are highlighted (clicked).

The aim of having the tuples located on the SAD graphical layout are:

1. Clarification of real-time behavior of each entity in question. By attaching the tuples to the actual entities (task and communication), a larger deal of clearness occurs. Also, the tuples may be changed at will in the subsystems they belong to.
2. Support for low-level processing of tuple values. I.e., the values may be used by, e.g., the operating system for various purposes. This requires a tool to extract the tuple information to the operating system. One could easily imagine that the values be used for, e.g., scheduling purposes.

IV. Conclusion

In this paper we started out by showing that the CODARTS software design method does not offer means for specifying real-time requirements in the design (graphical layout). Then we developed our own model, CODARTS+, which uses tuples to express the timeliness of tasks and communcation in the design. It does so by the assigning appropriate values connected to these entities in tuples attached graphically to the design.

References

[1] H. Gomaa, *Software Design Methods for Concurrent and Real-Time Systems.* Addison-Wesley, 1993. ISBN: 0-201-52577-1.

[2] C. D. Locke, *Best-Effort Decision Making for Real-Time Scheduling.* PhD thesis, CMU-CS-86-134, Carnegie Mellon University, Pittsburgh, PA, USA., May 1986.