

Adaptive Communications Scheduling for Embedded Systems

Greg Willden, Jeremy Price, Denise Varner, Ben Abbott
Southwest Research Institute®
{gwillden,jprice,dvarner,babbott}@swri.edu

Djaffer Ibaroudene
St. Mary's University
djaffer@stmarytx.edu

Abstract

Model-based technologies have been traditionally applied to embedded systems design from a functional perspective. Real-time scheduling issues and the associated modeling concerns are becoming increasingly more prevalent. Traditional schedulability analysis is being applied to modern model-based systems to generate schedulable systems. These schedulability analysis techniques are also applied to communications (e.g., network) scheduling. With adaptive communication scheduling, analysis need not simply decide the “pass/fail” criteria of a particular problem, but can explore the space of a variety of solutions.

This paper discusses adaptive communications scheduling for embedded systems. A fuzzy logic based scheduler is used in the system to allow for flexibility in experimentation of different scheduling algorithms. Various examples are presented that show how a model-based adaptive scheduler can be used to achieve a system that adapts to environment conditions at system design, synthesis, and runtime.

1. Introduction

Embedded systems are becoming increasingly more complex. These systems must meet not only functional requirements, but also timing and resource requirements. In real-time systems, the worst-case time required to respond to particular external events is part of the system’s functional requirements.

To meet the increasingly stringent functional and timing requirements, many systems must utilize

distributed processor solutions. In such cases, event response times become a matter of system end-to-end timing. That is, the response to a particular event may start on one processor in the system but will not be completed until the event effects have been communicated across the system network to one or more other processors that must also perform computations before the deadline of the particular event expires.

Many researchers have techniques for formal analysis of the schedulability in this type of distributed processor systems. A notable example is the work in [1]. Here, the paper describes an end-to-end deadline scheduling solution that attempts to break the circular dependency between deadline distribution and task assignment. It is interesting to note, that a core requirement for the successful application of this type of analysis is a deterministic and well-understood (and rolled into the analysis) communication-scheduling paradigm. Thus, in the case of this example of end-to-end scheduling work, communication scheduling is directly coupled to the analysis [1].

A main thesis of our work is that the coupling between communication scheduling and CPU scheduling must exist. As such, we explicitly open the connection between them. Further, we contend that not all scheduling decisions can be made in a “crisp” fashion, particularly in the case of distributed systems because global state cannot be precisely determined in a causal fashion [7].

2. Scheduling Policy vs. Schedulers

Typically researchers do not separate the concept of a particular scheduling policy from the framework in which it runs the scheduler. As such, the classic view of a scheduler is shown in the following figure. In this classic view, the scheduler is a single closed policy (or possibly a small set of policy algorithms) that chooses what to do next based on the state of currently available jobs. For example, a rate monotonic scheduler simply looks at the readiness and rate of the tasks and *always* chooses the highest rate task that is ready.

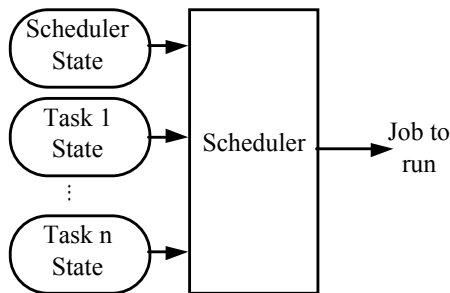


Figure 1 Classic view of a scheduler

The figure shown below separates the fuzzy decision “policy” from the scheduler “framework”. By doing so, it provides logical hooks for where to insert/modify scheduling policies. With the increasing usefulness of model-based systems (e.g. OMG’s Model Driven Architecture [8]) this form of abstraction will likely become more common. Separating these two items allows the policy to be synthesized from the model. In this way, a specialized adaptive scheduling policy can be generated for each new system deployment. The complex scheduler internals (context switch, task state, critical sections...) remain specialized for each hardware platform thereby maintaining high performance implementations. With this separation in mind, “schedule policy” model representation and synthesis strategies become a primary focus for scheduler research.

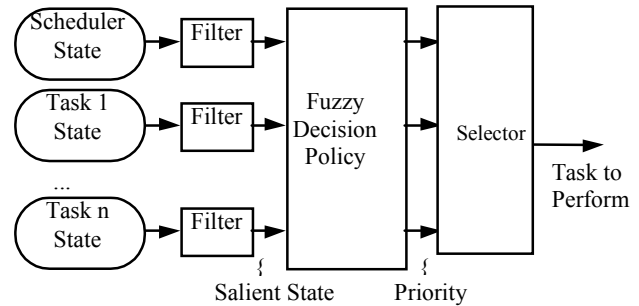


Figure 2 Scheduler with explicit fuzzy decision policy

In the case of a Fuzzy decision policy, crisp task state parameters are filtered (fuzzification) as to their degree of membership into various sets. These set memberships are then used by rules in the fuzzy decision policy to evaluate degree of membership in output membership sets that are then used by the selector to choose the task state with the highest priority set members (defuzzification). For those unfamiliar with fuzzy logic, the above process is described in detail in [2].

3. Fuzzy Logic Based

Representation of a particular scheduling policy has historically been provided as a set of “crisp” algorithmic rules. For example, the classic rate-monotonic crisp rules are: *the higher the rate of a process, the higher its priority, and higher priority processes always preempt lower priority processes* [9]. However, this overly restricts the range of possible policies. A logic paradigm supporting both hard and soft rules (as well as their interaction) allows a broader range of policies to be modeled. As such, we use fuzzy logic as a representation format for describing scheduler policy rules. Synthesis of a particular scheduler then becomes a matter of rule compilation and subsequent evaluation at runtime. It is interesting to note that automated model synthesis tools can often compile these fuzzy rules into very efficient full scheduler implementations [6].

Utilizing the fuzzy based approach does not lose generality over classic crisp schedulers. There exist many proofs that map classic CPU scheduling algorithms (RMS, EDF, PCP, DM - to name a few) to fuzzy logic implementations[2][6]. It can further be shown that there exist fuzzy logic based scheduling paradigms that are mathematically

equivalent to the classic real-time communication schedulers such as: Priority Driven Protocol (PDP), the Timed Token Protocol (TTP), and the distributed version of Generalized Rate Monotonic Scheduling (GRMS).

4. Timed Token Protocol

The Timed Token Protocol, standardized in [10] and [11], divides all communication into two classes, Synchronous and Asynchronous. Synchronous communication is periodic and subject to deadline constraints while asynchronous messages are aperiodic. A TTP network reserves system bandwidth for all synchronous messages and allows transmission of asynchronous messages as resources permit. The decision to transmit asynchronous messages depends on the early or late arrival of the synchronization token.

Appendix A contains a fuzzy logic based construction of the Timed Token Protocol that provides a solid base upon which to experiment and explore the space of hybrid scheduling paradigms. This exploration and experimentation benefits greatly from model based system design techniques that allow the designer to regenerate the complete system as the scheduler design parameters change.

5. Modeling

In order to experiment with adaptive communication scheduling techniques, the Embedded Systems Modeling Language (ESML) defined in [3], has been extended to support the fuzzy scheduler through Adaptive Services Coordination (ASC). The ASC extensions support modeling of linguistic variables used by the fuzzy rule set, as shown in the condensed metamodel in Figure 1 – Linguistic Variable Metamodel .

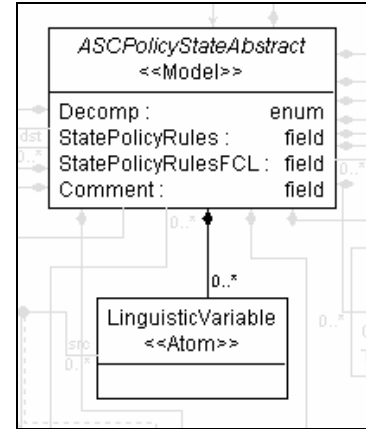


Figure 1 – Linguistic Variable Metamodel

Here, the metamodel is a language that allows the overall modeling environment to be defined. By modifying the metamodel, a new modeling paradigm is created. Metamodels are formally defined as UML Class Models [4].

The metamodel excerpt in Figure 2 demonstrates some small changes to the “Configuration” model of a pre-existing ESML paradigm, to implement the associations of fuzzy rules with communication ports.

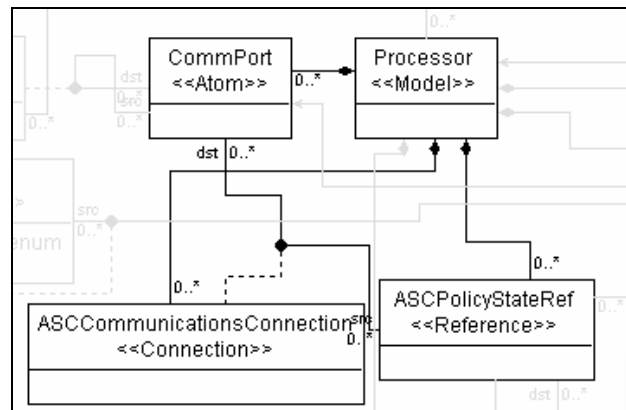


Figure 2 – Metamodel for Connection of Communication Port to Fuzzy Rules

These changes allow the domain expert to model the physical configuration of the processors and networks and to associate a fuzzy scheduler with the communications.

A summary of the two add-ons (shown in Figures 1 and 2) is that the ESML paradigm has been extended to support “ASCPolicyStates”. These ASCPolicyStates have associated rules for state changes, fuzzy scheduling (based on the formal FCL

grammar [5]), and connections to linguistic variables, which allow application specific parameters to affect the scheduling rules. Figure 2 highlights the ability to connect a scheduling policy not only to a processor, for classic CPU scheduling, but also the communication ports.

6. Practical Example

ESML was originally developed by Vanderbilt University. This modeling language allows us to create a simple Digital Signal Processing (DSP) example that illustrates the need for communications scheduling.

Figure 3 shows the interaction diagram for the example, which consists of five functional components distributed across two processors. The components model a system that calculates the Fast Fourier Transform (FFT) of the input signal, searches for peaks in the spectrum issues “alarms” for those peaks and displays both the peaks and the spectrum on separate parts of a display. For this example the FFT, Peak Search and Alarm Generator components reside on one processor while the Spectrum and Alarm Displays reside on another processor.

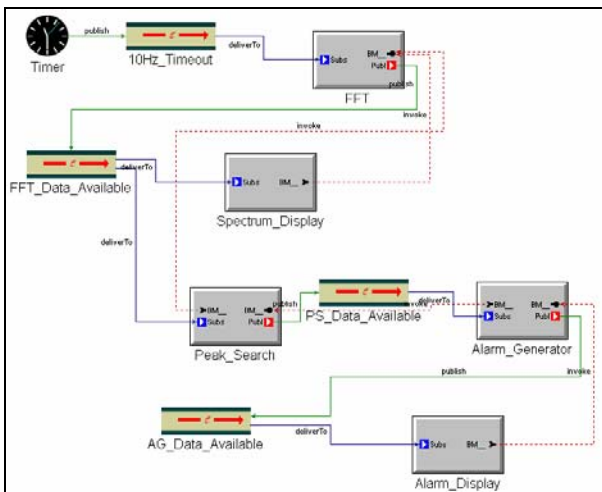


Figure 3 – Example DSP Interaction Diagram

The first observation that can be made about this example pertains to the relative size of the two messages being sent across the network. The Spectrum Display receives an FFT data block, every 100ms, consisting of on the order of 10^3 bytes or greater. However the Alarm Display receives many asynchronous messages, each on the order of 10 bytes in size.

In a simple networking situation, each message will finish sending before the next message will begin. This would cause some Alarm Display messages to be delayed due to pending transmission of FFT data blocks causing potentially harmful side effects.

One solution is to break the large FFT data blocks into smaller messages prior to scheduling communication over the network. This will allow the Alarm Display messages to be inserted between fragments of a larger FFT data block allowing the alarm to arrive with a lower end-to-end latency.

7. ASC Communication Components

To facilitate modeling of communications scheduling, Adaptive Services Coordination (ASC) components were created within the ESML model. Figure 4 demonstrates the use of ASC communication components in a simple Avionics example. This test scenario consists of six functional components assigned to two processors with three ASC components inserted at the processor-network boundary. The Pilot Control and Sensor Coordinator are assigned to processor one while the Waypoint, Flight Plan Display, Radar and Radar Display are assigned to processor two.

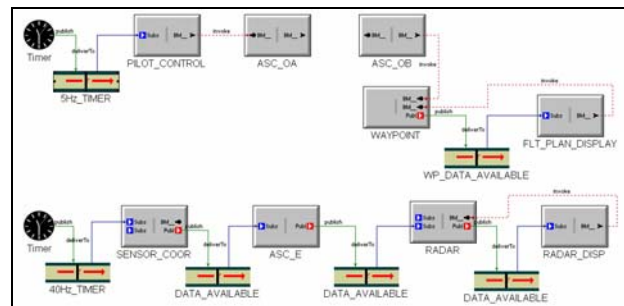


Figure 4 – A Simple ASC Scenario

The Pilot Control component transfers a large block of data to the Waypoint at 200ms intervals. The ASC_OA component breaks the pilot data into message fragments that the ASC_OB component receives and reconstructs prior to passing it to the Waypoint. The Sensor Coordinator sends a small notification event to the Radar component at 25ms intervals. The ASC_E component receives the event and passes it to the Radar component in a communication-scheduled fashion.

The ASC communication components coordinate the transfer of each message or message fragment according to the fuzzy logic based communication-scheduling algorithm generated by the modeling tools. A simple rule-set for this example, written in the Fuzzy Control Language (FCL)[5] is shown in Figure 5.

```

FUNCTION_BLOCK Comm
VAR_INPUT
  MsgSize : REAL; (* RANGE(0 .. 1048576) *)
END_VAR
VAR_INPUT
  qLength : REAL; (* RANGE(0 .. 100) *)
END_VAR
VAR_OUTPUT
  Priority : REAL; (* RANGE(0 .. 1) *)
END_VAR
FUZZIFY MsgSize
  TERM high := (524287, 0) (1048576, 1) (1048576, 0);
  TERM low := (0, 0) (0, 1) (524289, 0);
END_FUZZIFY
FUZZIFY qLength
  TERM high := (70, 0) (100, 1) (100, 0);
  TERM optimum := (25,0) (50, 1) (75,0)
  TERM low := (0, 0) (0, 1) (30, 0);
END_FUZZIFY
FUZZIFY Priority
  TERM high := (0, 0) (1, 1) (1, 0);
  TERM low := (0, 0) (0, 1) (1, 0);
END_FUZZIFY
DEFUZZIFY valve
  METHOD: CoG;
END_DEFUZZIFY
RULEBLOCK No1
  AND:MIN;
  ACCU:MAX;
  RULE 0: IF (MsgSize IS low) AND (qLength IS optimum) THEN (Priority IS high);
  RULE 1: IF (MsgSize IS low) AND (qLength IS empty) THEN (Priority IS high);
  RULE 2: IF (MsgSize IS low) AND (qLength IS full) THEN (Priority IS low);
  RULE 3: IF (MsgSize IS high) AND (qLength IS full) THEN (Priority IS high);
END_RULEBLOCK
END_FUNCTION_BLOCK

```

Figure 5 – FCL Rules based on Message Size and Queue Length

In the normal case, the smaller messages correspond to higher priorities. However, as the queue length approaches full the larger messages get higher priorities (to drain the queue faster). A three-dimensional view of this rule-set is shown in Figure 6.

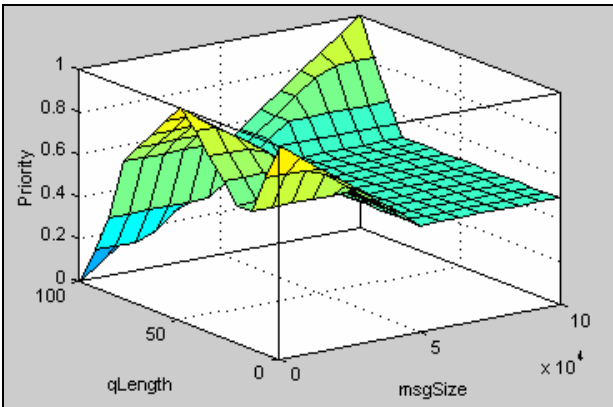


Figure 6 – 3-D view of FCL Rules

It is interesting to note that the flat area seen in the surface of Figure 6 represents the classic communications paradigm working within its expected region, that is, all messages have equal

priority. As long as the design assumptions of this paradigm remain correct, the synthesized scheduler will operate in an identical fashion to its classic counterpart. However when/if the assumptions prove incorrect due to unforeseen environment changes, the control surface slopes off in a more desirable fashion than its crisp (and thereby brittle) counterpart.

8. Summary

All communication schedulers are connected to their respective CPU schedulers, either implicitly or explicitly. Those scheduling paradigms that explicitly handle communication and CPU activity jointly, in an adaptive fashion, are more robust and flexible than their separated, “crisp” counterparts. The flexibility to define and utilize linguistic variables that are domain specific provides the power to tailor the system to domain specific needs. Adaptive schedulers based on fuzzy logic rule-sets support classic communication and CPU scheduling paradigms as well as hybrid designs synthesized on a deployment-by-deployment basis. Therefore a scheduler can easily be created which, under light load, is equivalent to a classic paradigm, but with increasing system load transitions smoothly across the control surface to better handle resource-constrained conditions.

9. Acknowledgement :

The work presented in this paper could not have been completed without the Generic Modeling Environment (GME) tool from Vanderbilt University [3]. Specifically, the help from Gabor Karsai and Sandeep Neema was indispensable.

Appendix A. Fuzzy Logic Based TTP

Theorem: There is a fuzzy scheduler that implements the timed token protocol. The protocol is not a priority schedule so this fuzzy scheduler apportions transmit time rather than priority

Proof: By construction.

Each station in the network receives a token. Each station S_i has a synchronous message transmit time (C_i) and an asynchronous message frame time or asynchronous overtime (T_{af}). The fuzzy input sets are the arrival time of the token (linguistic variables:

early, on-time, or late) and the dummy fuzzy set I1 for the presence or absence of an asynchronous message in the station message queue. The fuzzy output set is transmission time. Defuzzification is accomplished by calculating the maximum membership of the outputs.

At ring initialization, a Target Token Rotation Time (TTRT) is determined and synchronous bandwidths are set for each station (H_i). Two constraints are added:

$$1) \sum_{i=1}^n H_i = TTRT - \tau$$

2) $X_i > C_i$ where X_i is the minimum amount of time needed to transmit message I and overheads are ignored

The arrival time of the token is modeled by its membership in one of three sets: early, on-time, late.

If the token is early, its earliness ($E_t = TTRT - \text{time of token arrival}$) defines a limit on the number of asynchronous frames that can be transmitted, $nT_{af} < E_t$ where T_{af} is the fixed asynchronous frame transmission time and n is an integer

If **token early** AND **async message** = TRUE, **transmit time membership** = $C_i + nT_{af}$, $n = \lfloor E_t / T_{af} \rfloor$.

Output is: **min(transmit time membership, H_i)** If **token ontime** OR **async message** = 0

$$\text{transmit time} = C_i$$

If **token late** transmit time = 0 If the token arrives before the TTRT, an asynchronous message is ready, and there is at least one asynchronous period before the allocated bandwidth for that station is exceeded, then an asynchronous message is transmitted. Otherwise, only the synchronous messages are passed. This implements the TTP.

10. References

[1] Jan Jonsson and Kang G. Shin "Robust adaptive metrics for deadline assignment in distributed hard real-time systems", *Real-Time Systems Journal*, (in press), 2001

[2] F. Brown, Real time scheduling with fuzzy systems, Ph.D. Dissertation, Utah State University, 1998.

[3] G. Karsai, S. Neema, B. Abbott, D. Sharp. "A Modeling Language and its Supporting Tools for Avionics Systems", 21st Digital Avionics Systems Conference, August 2002.

[4] Ledeczki A., Maroti M., Bakay A., Nordstrom G., Garrett J., Thomason IV C., Sprinkle J., Volgyesi P., "GME 2000 Users Manual (v2.0)", document, December 18, 2001.

[5] "Fuzzy Control Language", International Electrotechnical Commission (IEC) document 61131-7.

[6] S. Neema, B. Abbott, "Real-Time Scheduler Based on Fuzzy Logic," Int. Conf. on Parallel and Distributed Processing Techniques and Applications. Las Vegas, Nevada, August 1997.

[7] Chandy, K., and Lamport, L., "Distributed snapshots: determining global states of distributed systems," *ACM TOCS*, V3 N1, Feb. 1985, pp. 63-75.

[8] Frankel, David S., "Model-Driven Architecture", OMG Press, 2003.

[9] C. Liu and J. Layland. Scheduling algorithm for Multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1); 46-61, Jan. 1973.

[10] ANSI Standard X3.139, FDDI Token Ring Medium Access Control, 1987.

[11] IEEE/ANSI Standard 802.4, Token Passing Bus Access Method and Physical Layer Specifications, IEEE, New York, 1985.