Designing Multiprocessor/Distributed Real-Time Systems Using the ASSERTS Toolkit

Kanad Ghose[1], Sudhir Aggarwal[1], Abhrajit Ghosh[1], David Goldman[1], Peter Sulatycke[1], Pavel Vasek[1], and David R. Vogel[2]

[1] Department of Computer Science, State University of New York, Binghamton, NY 13902-6000
ghose@cs.binghamton.edu
[2] Lockhead-Martin Federal Systems, Owego, NY

Abstract. We describe a real-time software engineering toolkit called ASSERTS that provides complementary analysis and simulation capabilities to assist the designer of multiprocessor/distributed real-time systems. ASSERTS allows users to describe the parameters of the hardware platform, interconnection and the kernel, the programming model (shared or distributed memory) and the task system for the purpose of simulation and analysis. The simulation component of ASSERTS is quite detailed and features a number of *built-in* simulation models for practical real-time schedulers, interconnections as well as models of resource access protocols (such as priority inheritance and priority ceiling protocol). Users can describe the behavior of the tasks at various levels of abstraction using a fairly small set of about 20 macroinstructions for the purpose of simulation.

1. Introduction

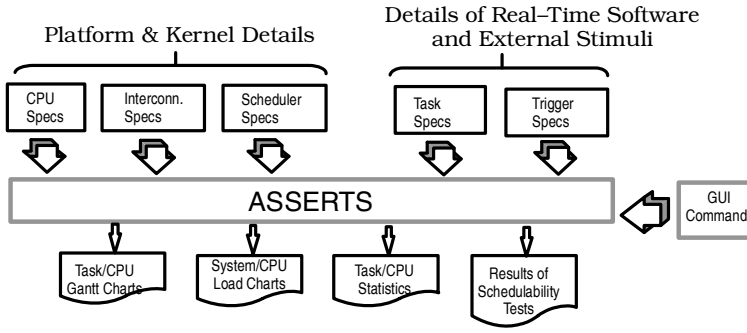
Real-time system design has emerged as a mature computer science/engineering discipline over the recent years. This has been made possible through the evolution of a science for real-time systems and the availability of formal analysis and simulation tools for the real-time software design process itself [ABRW 94], [LRD+ 93], [StLi 96], [Ti98]. Analytical techniques for testing the validity of timing specifications of real-time typically take the form of schedulability tests. Schedulability tests of a rigorous and sufficiently general nature exists for uniprocessor systems and allow software designers to get a hard guarantee on the timing performance when the test passes. Unfortunately, when the schedulability tests fail for an uniprocessor system, it rarely tells the designer of the sequence of events that lead to the timing violation. With multiprocessor/distributed systems, there is a marked lack of scheduling theories and schedulability tests of a general nature. This is primarily due to the delays introduced by interactions among tasks, which reflect task dependencies and introduce delays that depend on potential contention within the interconnection. While such delays may be easily enumerated for a small number of tasks for the purpose of analysis, the situation quickly gets out of control even with a moderate number of tasks on a system with a few nodes. This is precisely where accurate simulation techniques are useful. We thus believe that analysis and simulation play complementary and useful roles in the design of real-time software, particularly for multiprocessor/distributed real-time systems.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 505-510, 1999.

This paper focuses on facilities within a prototype real-time software design environment called ASSERTS (*A S*oftware *S*imulation *E*nvironment for *R*eal *T*ime *S*ystems) that supports such capabilities. Additionally, ASSERTS also incorporates an analysis component to do formal schedulability tests where applicable. In particular, the simulation component of ASSERTS can identify the duration of blocking of a task due to dependencies with other tasks (located in possibly different nodes), which can then be used in analytical tests, such as the progressive schedulability test [SRL 88]. ASSERTS is currently available for Solaris, AIX, NT and Windows 95 platforms.

2. Overview of ASSERTS

ASSERTS has been specifically designed to support real-time software design for both uniprocessor and multiprocessor systems and incorporates both analysis and simulation components that can be used by the software designer starting from the early design phase. Figure 1 summarizes the inputs and outputs of ASSERTS. Platform and kernel details include specifications of the CPUs (relative speed, RAM size), real-time kernel details (scheduling discipline, context-switching time etc.) and details of the interconnections used (types, parameters, software layer delay parameters etc.). Details of the real-time tasks include a description of tasks, their placements on the CPUs, inter-task dependencies (through messages or accesses to shared variables). Each task is described using two components - the task attributes (periodicity, period, deadline, static priority - if any, CPU on which task is to be run etc.) and the task body, which is the functional description of the various phases of the task. The task bodies can be progressively refined as the design evolves. The duration of the compute phases of a task are described using the notation {seconds.millisecs.microsecs.nanoseconds}; the time quoted is the duration on a reference CPU. Specified timing durations gets scaled appropriately based on the relative speed of the CPU on which the task is run. The delays of all other phases of a task are obtained through actual simulation.



igure 1. Inputs and outputs for ASSERTS

Most of the descriptions can be entered and altered through a GUI. Users may specify the outputs needed from ASSERTS through the GUI. These outputs could be Ganntt charts (for individual tasks, CPUs, interconnection links etc.), load charts

(scheduler queue states, blocked task queues, CPU ready task queues etc.), detailed logs or the result of analytical tests (such as the Liu/Layland test, progressive schedulability tests [SRL 88], specific rate-monotonic schedulability tests for PCP etc.) .Other analytical tests are also possible for other types of schedulers (such as EDF, cyclic executives etc.) for uniprocessor systems.

When multiple Ganntt charts are opened, one per window, the cursors in the various windows can be synchronized to allow cause-effect events to be easily identified. These windows also allow users to zoom-in or zoom-out, or scroll along the time scale, again in a synchronized fashion. Task phases displayed within the task windows are color coded and users can click on any portion of a Ganntt chart to see what macroinstructions they correspond to within the body of a task. Similar facilities are available for the other types of windows that can be opened (for CPUs, interconnection etc.).

We believe that a particularly useful feature in ASSERTS is the ability to progressively describe the functional behavior of a task at a very high level as its design evolves. This description identifies various phases of a task, such as the duration of computation phases, task interaction (via message passing or shared variable access), conditional waiting etc. A small set of “macroinstructions" are defined to describe each phase of a task (see [GAV+ 97). Initially, when the details of a task are unclear, the task may be described as a single computation phase; its description may then be progressively refined by adding other phases as the design evolves. Figure 2 shows the progressive refinement of the description of a task. The functional description of a task is used by the simulation component of ASSERTS, along with the task attributes. The parameters used by the analytical tests are typically gleaned from the task attributes.

```
start_task
    compute 0.10.0.0
end_task
```

(a)

```
start_task
    compute 0.5.0.0
    lock(sema)
    compute 0.5.0
    unlock(sema)
end_task
```

(b)

```
start_task
    compute 0.3.0.0
    send b ”server” 1 256 ro
    lock(sema)
    if (level > 20) go to L
    compute 0.4.0
L:  compute 0.1.0
    unlock(sema)
end_task
```

(c)

igure 2. Progressive refinements of the functional behavior of a task from the initial description based on the allocated timing budget shown in (a).

3. Message Passing, Interconnections and Shared Memory Access

At the level of the tasks, user's describe communication phases within tasks using the following send-receive macroinstructions:

```
send <type> <destination> <message_tag>  <message_size> [<waiting_mode>]

receive  <type> <source> <message_tag>  <buffer_mode> [<waiting_mode>]
```

The send and receive macroinstructions specify the type of the primitive used. For the `send`, the type can be blocking (sending task waits till the message transmission

is completed), blocking with an acknowledgement (sending task block till the receiving task sends an acknowledgement back) and non-blocking. For a `receive` the primitive can be either blocking or non-blocking. The recipient task and the sending task are explicitly named in the `<destination>` and `<sender>` fields, respectively. A message tag is also included to match up the sends and receives between a pair of tasks. The `<message_size>` field gives the message size in bytes for a send. For a receive, the `<buffer_mode>` field indicates the mode of reading the buffer (read only the matching message, read the first message in the buffer irrespective of the message tag and read the first message irrespective of its tag). The `<waiting_mode>` field specifies the type of waiting where applicable - busy (i.e., spin) waiting or sleep waiting. The send and receive macroinstructions thus allow a wide variety of communication primitives to be modeled.

The simulation component simulates the basic actions of these primitives to determine the times needed for the task phases described by these macroinstructions, including any contention delays on the interconnection. Another variant of the communication macroinstructions also exist - these are used for describing the actual message contents.

In the interest of accuracy, ASSERTS models both the software, hardware and queueing delays of message passing. The software delays are due to copying delays between the user's and kernel's address space, as is typical in many systems. The number of copying steps may be specified as one (direct DMA between user-space buffers and the network interface) or two (transfers between the user-space buffer and the network interface going through a kernel-level buffer). The delay of each copy step is given by an expression of the form:

$$T_{copy} = \text{flat\_overhead} + \text{constant} * \text{message\_size}$$

Consequently, ASSERTS requires users to input the flat overhead and the constant. The actual software delay(s) is(are) then computed based on the message size. The “hardware" components of the delays in message passing are specific to the type of the interconnection used, as well as on the interconnection topology. For convenience, ASSERTS incorporates standard simulation models of a variety of commonly used interconnections, such as Ethernet, IEEE Futurebus and point-to-point interconnections (such as ATM, Myrinet).

The simulation models for these interconnections capture the impact of contention, arbitration and propagation delays. An extensive C++ class library also exists that allow advanced users to model other types of interconnections. The interconnection topology is specified by the user, as part of the platform description. Relevant delay parameters for each type of interconnection are also specified. These delay parameters can be easily altered through the ASSERTS GUI to undertake what-if studies, such as gauging the impact of moving from 155 Mbits/sec. ATM to 1.2 Mbits/sec ATM etc. or moving from a 10 Mbits/sec. Ethernet to a Futurebus interconnection.

Any queueing delay in message passing is also captured in the simulations. One use of simulation will be to extract accurate blocking times due to dependencies across tasks, which can then be used in analytical schedulability tests. Events occurring on the interconnection links for message passing can be seen on the Ganntt charts for the interconnections. Queue states for the network interface can also be seen through the

GUI. These information are extremely useful in identifying communication bottlenecks, errors in programs or in performing post mortems when timing violations are detected.

There are several aspects to modeling shared variables in ASSERTS. First, the location of shared variables within the system are described using a statement of the form:

```
shared <variable_name> <location> <size>
```

The `<location>` field specifies the CPU node where this variable is located. Within the task body (which is the functional description of a task), shared variables are accessed by each of the following macroinstructions:

```
lock(<variable>)  [<wait_type>],  unlock(<variable>)
wait(<condition>)  [<wait_type>]
if (<condition>) go to <label>
```

where the condition is a relational expression involving one or more shared variable

```
<variable_1> = <variable_2>|<literal>   (assignment)
```

Since the shared variable accessed by a task using one of the above macroinstructions can be located on a different node, the appropriate delay has to be accurately modeled. This model depends on the type of primitives that are used in the actual system: the remote access can be implemented on top of message passing primitives or as a direct access to a memory module (as in a bus connected system, with a memory module board and CPU boards). In the first case, the delay in accessing a shared variable may be modeled as a send-receive pair. In the latter case, the delay has to be modeled as a request-response pair on the interconnect to the relevant memory module, with a specification of the access time of the memory module (plus any queueing delay at the module) as service time of the request. In ASSERTS, for this latter case interconnect cycles corresponding to remote reads, writes and a read-modify-write cycle (for locking) are explicitly modeled. Users specify the memory access time and software delays in the access path, if any. Interconnect events of the appropriate type are automatically generated at the time of compiling the macroinstructions that refer to remote shared variables. These events are simulated, as in the case of message passing, for the specified interconnection.

As in the case of message passing, the simulation can provide an accurate estimate of the task blocking times due to shared variable access, which can then be plugged into analytical tests for schedulability. Also, similar to the case for message passing, windows can be opened through the GUI to see the interconnection events due to shared variable access, as well as the state of relevant queues.

4. ASSERTS and Other Real-Time Design Tools

ASSERTS is only one of a large number of tools that have been designed to assist real-time software development and design. PERTS [LRD+ 93], now marketed by Wind River (and its simulation component, DRTSS [StLi 96]), STRESS [ABRW 94] and the TimeWiz tool offered by TimeSys Corp. [Ti 98] offer a comprehensive system-independent analysis and simulation facility, similar to ASSERTS.

The PERTS tool uses a task graph model to describe the task system, where a task is essentially a computation phase preceded and terminated by optional communication phases, making the model somewhat unrealistic. PERTS includes a task allocation component and implements a schedulability test based on assuming idealistic, contention-free communication. Progressive refinement of the task bodies are not possible in PERTS, which also lacks models for interconnections. STRESS and TimeWiz are both quite similar to ASSERTS. STRESS, however, does not include built-in models for interconnections and scheduling algorithms; its GUI also appears to be somewhat limited. TimeWiz is the closest analog of ASSERTS, but like PERTS it fails to support progressive refinements of task bodies. Unlike ASSERTS, TimeWiz incorporates extensive design documentation generation facilities.

Our current efforts include the addition of an allocation component for ASSERTS, a two-level scheduling algorithm and the ability to interface with UML design tools. ASSERTS is also being used within Lockheed-Martin on a limited scale. Examples of the screen shots for ASSERTS, as well as an on-line manual for the ASSERTS 1.0 release can be found using the url: http://cs.binghamton.edu/~ghose/ASSERTS.

References

[ABRW 94] Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J., "STRESS: a Simulator for Hard Real-Time Systems", in Software-Practice and Experience, Vol. 24(6) (June 1994), pp. 543-564.

[GAV+ 97] Ghose, K., Aggarwal., S., Vasek, P., et al, "ASSERTS: A Toolkit for Real-Time Software Design, Development and Evaluation", in Proc. 9-th Euromicro Workshop on Real-Time Systems, 1997, pp. 224-232.

[LRD+ 93] Liu, J. W.-S., Redondo, J.-L., Deng, Z. et al, "PERTS: A Prototyping Environment for Real-Time Systems", in Proc. 14th. IEEE Real-Time Systems Symposium, 1993, pp. 184-188

[SRL 88] Sha, L., Rajkumar, R. and Lehoczky, J. P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", Tech. Report, Software Engineering Institute, CMU, May 23, 1988.

[StLi 96] Storch, M. F., and Liu, J. W.-S., "DRTSS: A Simulation Framework for Complex Real-Time Systems", in Proc. IEEE Real-time Technology and Applications Symposium, 1996, pp. 160-169.

[Ti 98] Timesys Inc. web pages at: http://www.timesys.com