Using SCAN to Analyze the Schedulability of a Real-Time Application

Vasilis C. Gerogiannis\textsuperscript{1, 3} Manthos A. Tsoukarellas\textsuperscript{2, 3}

\textsuperscript{1}University of Patras, Dept. of Mathematics, Sector of Informatics
\textsuperscript{2}Technological Educational Institute of Patras
\textsuperscript{3}Advanced Informatics Ltd.

Advanced Informatics Ltd., 35 Gounari Ave., 262 21 Patras, Greece
Tel: +30-61-623639 Fax: +30-61-622308
e-mail: V.Gerogiannis@advinfo.pat.forthnet.gr

Abstract. Recent advantages in real-time scheduling theory have provided a rigorous set of analytical methods for guaranteeing a priori that real-time software will meet its timing requirements (schedulability analysis methods). In this paper, the architecture, the usage and the underlying methodology of an interactive SChedulability ANalysis tool (SCAN) are presented. The tool supports design-oriented schedulability analysis and incorporates an extensive set of analytical algorithms. The key characteristic of the tool is the underlying practical framework, which permits the systematic representation and analysis of a real-time application, as well as the selection of a feasible configuration that can meet all timing requirements. In addition, SCAN can cooperate with a software monitor, which allows a better characterization of the application actual timing parameters.

1. Introduction

Guaranteeing that real-time software will meet its timing requirements, a process well-known as schedulability analysis, is an integral and important part of the real-time software development process [[17]]. Existing schedulability tools/techniques are strictly related to the specific phase of software development in which they are involved:

- \textit{specification phase} (e.g., Real-Time Logic [[8]], Real Time Temporal Logic [[13]] etc.): these approaches can also be found in the literature as safety analysis techniques. However, they differ from traditional safety analysis techniques in two important aspects. First, they focus on the timing behaviour of systems and, second, they try to analyze the software safety requirements (i.e., system specification) and not the system safety requirements. In other words, they are concerned with the consistency of safety assertions with respect to the timing constraints.

- \textit{design phase} (e.g., Scheduler 1-2-3 [[18]], PERTS analyzer [[12]]): although design-oriented schedulability analysis techniques analyze in depth the feasibility of the timing requirements, they usually lead to an estimation of execution times, which is too pessimistic in comparison to the actual ones. Indeed, these approaches utilize certain...
analytical algorithms, which are based on a worst-case configuration and they do not consider any implementation issues.

- **implementation (run time) phase** (e.g., Stoyenko’s schedulability analysis [15], Haban and Shin’s monitoring approach [7]): the major characteristic of these approaches is that they compute an estimation of execution times based on either the knowledge of accurate timing information at a number of levels in a program or the analysis of monitored results. Compared with design-oriented schedulability analysis, implementation-oriented techniques are more accurate. However their limitations include lack of abstraction and implementation dependency. In addition, these approaches are cost/time effective since they do not provide any feedback to real-time engineers until the application has almost been developed.

In this paper, SCAN (SChedulability ANalysis tool) is presented, an implementation independent tool, which concentrates on the predictability of a hard real-time application’s timing behaviour at the design phase. The development of the first prototype of SCAN [6] has been motivated from the user requirements expressed in the ESPRIT project 8906-OMI/CLEAR [16]. As far as the current version of the system is concerned, it appears to be more versatile and powerful since it incorporates an extensive library of schedulability analysis algorithms and it features a graphical user interface. A subset of these algorithms is being developed in the context of the ESPRIT project 20899 - OMI/ANTI-CRASH [4].

SCAN is a system composed of software modules which realize and implement recently established analytical methods/algorithms and priority-driven scheduling strategies for analyzing the timing behaviour of hard-real time applications. The tool is similar to design-oriented schedulability analysis tools such as the PERTS schedulability analyzer, a system being developed in an ongoing project in the University of Illinois [12]. The user of PERTS analyzer represents the architecture of the target application as an abstract task graph. All task parameters and dependencies have estimated values derived from the system requirements. The user can also give a resource graph that represents the amounts of resources budgeted from the application. Consequently, PERTS analyzer can be used both to determine whether the budgeted amounts of all resources are sufficient to achieve the required degree of responsiveness, and execute certain analytical algorithms. In addition, the tool has been designed to be able to cooperate with an object-oriented simulation environment for allowing the experimental evaluation of alternatives in scheduling and resource management.

Compared with PERTS analyzer, SCAN employs a more practical and applicable framework. This framework has been recently presented in [9] and provides real-time engineers with comprehensible, practical and still systematic methods in order to represent, design and analyze their applications. Rather than using the formal task and resource graphs as reference models, the underlying framework of SCAN is based on a systematic representation of a real-time application in terms of specific tables (Situation, Implementation and Techniques Tables) which have a uniform format. In addition, SCAN can cooperate, instead with a simulator, with an event-trace software monitor called MAT, which is being developed in the context of the ESPRIT project 20576 - OMI/TOOLS [5]. MAT feeds the monitored data (e.g., actual execution and blocking times) back to SCAN and consequently, allows better characterization and prediction of the actual timing behaviour of the analyzed real-time application. Therefore, SCAN serves as an interactive tool that enables the user to select a feasible configuration, which can meet all the timing requirements.
2. Description of SCAN

SCAN supports a library of quantitative analysis methods which regard all the tasks of a real-time application as responses to certain events with hard real-time requirements (i.e., the

![Figure 1: Architecture of SCAN](image)

event/action model is considered). Events (either timed (T), environmental (E) or Internal (I) ones) are characterized according to their arrival pattern as follows:

- **periodic** (Per): events arriving at constant intervals (periods)
- **irregular** (Ir): events arriving at pre-specified, not constant intervals
- **bounded** (B): events having a minimum arrival separation (bounded arrival pattern)
- **bursty** (Bur): when the number of events over a particular burst interval is restricted
- **unbounded** (U): when each event arrival is described in terms of a probability distribution function.

Every event has an associated response (i.e., the response time is the length of time between the occurrence of the event and the completion of the response); responses can be decomposed into smaller segments of work called actions, which are implemented as single tasks. Usually, a response consists of a single action and, therefore, it corresponds to a single task. A job is simply the instance of a response that is associated with a specific event in an event sequence. The next occurrence of an event in the same event sequence will be a different job.

As far as resource constraints are concerned, schedulability analysis has to take under consideration the blocking times caused by the exclusive sharing of resources [14]. The blocking time of an event has to be specified in case of priority inversion. This situation may occur if a low priority task uses a resource, while a higher priority task is forced to wait for it. The analysis also takes into account the events’ priorities which can be defined either according to a specific scheduling policy or explicitly by the user.

The general architecture of SCAN is depicted in Figure 1. A WIMP-like (Windows, Icons, Menus and Pointing Device) User Interface provides an easy-to-use interactive interface, which presents the capabilities of the tool in a user-friendly way and enables the user (i.e., application designer) to describe the application characteristics.
The user interacts with the tool by following a three stage process:

- **representation of a real-time situation**: the user can specify the basic characteristics/requirements of an application in a *Situation Table*
- **identification of possible implementations**: for each possible implementation, the user constructs a corresponding *Implementation Table*, which presents more detailed information about the application
- **specific representation of the application**: for each implementation table, the tool proposes a corresponding *Technique Table*, which represents the necessary input in order to apply one of the supported schedulability methods.

A Situation Table is organized into four distinct areas, which present information about events, responses, actions and resources (Table 1). In such a table, the fields that vary between the possible different implementations correspond to variables or to the wild card notation (Table 2).

**Table 1: Situation Table Format**

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>Event Name</th>
<th>Type</th>
<th>Arrival Pattern</th>
<th>Deadline</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENTS</td>
<td>Action Id</td>
<td>Jitter</td>
<td>Resource Id</td>
<td>Task Id</td>
<td>Time Used</td>
</tr>
<tr>
<td>RESPONSES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Situation Table**

<table>
<thead>
<tr>
<th>TABLE X</th>
<th>Event Name</th>
<th>Type</th>
<th>Arrival Pattern</th>
<th>Deadline</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>T / E</td>
<td>Per, T1</td>
<td>H, [0,D1]</td>
<td></td>
<td>a1</td>
</tr>
<tr>
<td>e2</td>
<td>E</td>
<td>Bur, Count2, Int2</td>
<td>H, [0,D2]</td>
<td></td>
<td>a2</td>
</tr>
<tr>
<td>e3</td>
<td>T / E</td>
<td>Per, T3</td>
<td>H, [0,D3]</td>
<td></td>
<td>a3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Id</th>
<th>Jitter</th>
<th>Resource Id</th>
<th>Task Id</th>
<th>Time Used</th>
<th>Priority</th>
<th>Resource Id</th>
<th>Type</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task1</td>
<td>C1</td>
<td>*</td>
<td></td>
<td>CPU-1</td>
<td>Rate Monotonic</td>
</tr>
<tr>
<td>a2</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task2</td>
<td>C2</td>
<td>*</td>
<td></td>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task3</td>
<td>C3</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Implementation Table

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Type</th>
<th>Arrival Pattern</th>
<th>Deadline</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>T / E Per, 25</td>
<td>H, [0,25]</td>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>e2</td>
<td>E Bur, 25, 200</td>
<td>H, [0,200]</td>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>T / E Per, 350</td>
<td>H, [0,350]</td>
<td>a3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Id</th>
<th>Jitter</th>
<th>Resource Id</th>
<th>Task Id</th>
<th>Time Used</th>
<th>Priority</th>
<th>Resource Id</th>
<th>Type</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task1</td>
<td>10</td>
<td>High</td>
<td>CPU-1</td>
<td>CPU</td>
<td>Rate Monotonic</td>
</tr>
<tr>
<td>a2</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task2</td>
<td>1</td>
<td>Very High</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>n/a</td>
<td>CPU-1</td>
<td>task3</td>
<td>120</td>
<td>Low</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Techniques Table

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Arrival Period</th>
<th>Action Id</th>
<th>Execution Time</th>
<th>Priority</th>
<th>Blocking Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>e2</td>
<td>200</td>
<td>a2</td>
<td>25</td>
<td>Very High</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>e1</td>
<td>25</td>
<td>a1</td>
<td>10</td>
<td>High</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>e3</td>
<td>350</td>
<td>a3</td>
<td>120</td>
<td>Low</td>
<td>0</td>
<td>350</td>
</tr>
</tbody>
</table>

During the second phase, the user identifies possible configurations by providing the Implementation Tables. Such a table (Table 3) has identical format to the original Situation Table, but any implementation details are filled in. The user may give several Implementation Tables, which represent alternative views of the same application. Each table corresponds to a database file stored in the Database Module.

The Preprocessing Module of SCAN is responsible to propose a restricted representation of the application in a Techniques Table format (Table 4). An Implementation Table is transformed into a Techniques Table that can be used as input by a specific schedulability analysis method. Such a table is also stored in the Database Module and presents a worst-case scenario, in which aperiodic events arrive at the fastest possible rate: all hard real-time aperiodic events (either bursty or bounded) are transformed to periodic ones with the minimum possible interarrival separation as period. The user can select among a variety of possible implementations, which transform aperiodic events to equivalent periodic ones. Currently, the supported methods allows the user to use an interrupt handler, service the aperiodic events at specified priorities, or use techniques such as polling tasks and sporadic servers [[9]]. In addition, the user is always able to modify the proposed Techniques Table and to specify different values from the suggested ones.

3. The Schedulability Analysis Module of SCAN

The Schedulability Analysis Module of SCAN incorporates an extensive set of quantitative analysis methods that are based firmly on recent advantages of real-time scheduling theory.
This module is responsible to select and execute the analysis method corresponding to the obtained Techniques Table. The main body of the supported methods has been derived from [9].

The Schedulability Analysis Module serves as an interactive tool which performs a two stage process, namely the *First* and the *Second-Level Analysis*. The First-Level Analysis provides the user with an initial characterization of the application timing behaviour. When deadlines for all events in the Techniques Table are not after the end of the periods and the response to each event executes at a single priority, the user may select to execute methods that compute the utilization bound [11, 14] for each event or for all the set of events. These methods give a general sense of the timing behaviour, since utilization bounds offer only a rough, although sufficient, schedulability condition. For this reason, when the utilization bound check fails to determine the schedulability, the user may select to execute a method that calculates worst-case response times. It should be noted that the user is always able to select directly such a method in order to obtain an exact characterization of the application timing behaviour.

In particular, the following two classes of methods are incorporated in the Schedulability Analysis Module:

*Class 1 - Methods Based on the Computation of Utilization Bounds:* Two methods have been implemented. The first one computes the utilization bound for a whole set of periodic events [11]. It is applied when all deadlines for all events in the techniques table are equal to the end of their periods and priorities are assigned according to the Rate Monotonic policy. The second method [10] computes the utilization bound for each specific periodic event and can be applied when deadlines for all events in the Techniques Table are less than or equal to the end of the corresponding periods. In addition, the method can be applied to other fixed-priority scheduling cases (different than the Rate or the Deadline Monotonic priority assignment).

*Class 2 - Methods Based on the Computation of Response Times:* This class of methods focuses on the computation of worst-case response times [1, 9]. Algorithms in this class involve more complex computations but they are more precise and always guarantee timing constraints (i.e., the computation of worst-case response times provides a sufficient and necessary measure for schedulability analysis). The following three methods have been implemented:

- calculation of the worst-case response time for each event / deadlines in the Techniques Table are within the periods and there are no blocking delays,
- calculation of the worst-case response time for each event / arbitrary deadlines and blocking times appear in the Techniques Table,
- calculation of the worst-case response time for each event / deadlines have arbitrary values and events are composed of multiple actions, each of which is associated with a different priority level.

Second-Level Analysis provides alternative feasible solutions so that all deadlines are met, as well as suggests acceptable modifications/changes concerning the resource and the timing parameters of the event sequence involved in the Techniques Table. Furthermore, the Schedulability Analysis Module incorporates methods suggesting values for the blocking delays. In particular [9, 10]:

*Class 3 - Methods for Providing Guidelines and Alternative Feasible Solutions:* Methods in this class can be used during the Second-Level analysis in order to compute the following useful parameters:
• the spare capacity which is the amount of execution time that can be added to the response of an event while preserving the fact that lower priority events will meet their deadlines.
• how much the execution and blocking times can be increased and still preserve the timing requirements.
• the amount of execution time (i.e., the overrun) that can be subtracted from a response, in order to guarantee the deadline requirement.

Class 4 - Analyzing the Schedulability of Tasks that Synchronize to Share a Common Resource: Methods in this class compute the blocking term that can be associated with an event. In the case that several responses share a single data resource, specific synchronization protocols can be examined with respect to the blocking delay that may be introduced (e.g., a FIFO semaphore is used at the priority of the task that accesses the resource, Interrupt Masking or Basic Priority Inheritance Protocol).

The possibility of sharing multiple resources may create additional problems such as deadlock and chain blocking. Deadlock occurs when a response uses the resource and also waits for another resource to be released. Chain blocking occurs whenever a new resource lock is requested. In particular, if there are multiple shared resources, SCAN considers the Basic Priority Inheritance and the Priority Ceiling synchronization protocols.

4. Interaction with the Monitoring Tool

All methods of the Schedulability Analysis Module depend on the user’s a priori estimate of the worst-case execution time of each task. Such an estimate may lead to useful conclusions about the application timing behaviour only at the early stages of design. However, at the later phases of the design process the application designer needs more accurate timing information, in order to proceed with a more realistic schedulability analysis.

The problem of determining the execution time has traditionally been viewed from two perspectives: through analyzing the source code of a task (by determining the path that results in the longest execution time) [[15]] or measuring the actual execution time of a task [[7]]. The first approach has the major disadvantage that it usually leads to an unreasonably large estimate of worst-case execution times due to the complexity of modern microprocessors. Hardware factors (such as pipelining, processor bandwidth and memory speed) or software factors (programming language and compiler optimization) result in a better performance than the one estimated by source-code analysis. For this reason, the second approach has been adopted in SCAN: a software monitoring tool called MAT [[5]] is used to measure the actual execution and the possible blocking time of each task. Although the followed approach is implementation dependent, it provides the most accurate results. However, SCAN can also be used as a stand alone, completely implementation independent tool, without considering the interaction with MAT.

MAT performs event-trace monitoring on applications running on the EOS Real-Time Operating System [[2]]. The MAT user is able to trace in a real-time application the timings of the corresponding EOS system calls. Figure 2 presents the task state diagram and the corresponding functions EOS. The actual execution time of a task is measured between subsequent calls to “resched” or “sys_suspend_task” (these primitives change the task state from “Running” to either “Ready” or “Suspended” state). As far as blocking delays are concerned, a task is blocked when it waits on a semaphore that has been previously locked by a lower priority task. Therefore, the blocking time is measured between subsequent executions of “sys_wait_semaphore” and “sys_signal_semaphore”.

All these timings are stored in a trace data base (Figure 1). Consequently, the SCAN user may give to the Techniques Table maximum, minimum or average measured values, and thus a more realistic experimentation can be performed.

![Task State Diagram and Corresponding EOS System Calls]

Figure 2: Task State Diagram and Corresponding EOS System Calls

5. Example

In this section a case study of the design of an actual application is presented. The application is a vetronics braking system developed in the context of OMI/ANTI-CRASH by THOMSON-CSF and has the following basic characteristics/requirements (provided by the user in the Situation Table presented in Table 2):

- periodic and bursty events (interrupts) are involved
- no blocking appears
- deadlines appear at the end of the periods
- hard deadlines are associated with interrupts
- fixed priority scheduling (low priority for periodic tasks - higher priority for interrupts)
- Rate Monotonic scheduling policy.

Following that, the user provides an Implementation Table (Table 3), which presents possible values of the application characteristics and selects the Interrupt Handler implementation for handling the bursty events: each periodic event is associated with a single action (response) which is implemented as a single periodic task and each bursty event is associated with a single action (response) which is implemented as an interrupt handler and executes at the highest priority level). This implementation considers that the priorities of the periodic tasks follow the Deadline Monotonic priority policy. However, in
the current situation this does not cause problems, since deadlines appear at the end of periods and therefore, the Deadline Monotonic is equivalent to the Rate Monotonic policy. The preprocessing module will transform the bursty event to one periodic event. In particular, the event with the event density of 25 arrivals per 200 msec is equivalent to a set of 25 events, each of which having the following characteristics:

- it is bounded
- it has an execution time of 1 msec
- it has a minimum interarrival interval of 200 msec.

All these 25 bounded events are equivalent to one periodic event with period equal to 200 msec and execution time equal to 25 msec. Consequently, the tool proposes to the user the Techniques Table presented in Table 4, in which all events are now periodic. The user may select to execute either a method of Class 1 (Methods Based on the Computation of Utilization Bounds) or Class 2 (Methods Based on the Computation of Response Times). For example, the following iterative method of class 2 can be used to compute the worst case response time for each event $e_i$ (this method is applicable, since all deadlines are less than or equal to the corresponding periods and there are not any blocking delays):

1. Calculate the first approximation of the response time $a_0$. The execution time $C_i$ of $e_i$ and of all higher priority events are summed.

   $$ a_0 = \sum_{j=1}^{i} C_j $$

2. Calculate the next approximation of the response time $a_{n+1}$, by considering the preemption times.

   $$ a_{n+1} = C_i + \sum_{j=1}^{n} \left( \frac{a_n}{T_j} \right) C_j $$

3. If $a_{n+1} < D_i$ and $a_{n+1} \neq a_n$, then go to 2.
   - If $a_{n+1} > D_i$, then event $e_i$ cannot meet its deadline.
   - If $a_{n+1} = a_n$, then $a_n$ is the response time of event $e_i$ and the algorithm is terminated.

Analysis with SCAN has provided the following results:

- the response time of event $e_2$ is equal to 25 msec, and thus it meets its deadline at 200 msec
- the response time of event $e_1$ is equal to 35 msec, and thus it misses its deadline at 25 msec
- the response time of event $e_3$ is equal to 290 msec, and thus it meets its deadline at 350 msec.

6. Conclusions

SCAN answers the question whether all the hard real-time requirements of a set of real-time tasks are guaranteed to be met or not. The tool performs a real feasibility analysis and focuses on the predictability of an application timing behaviour. The results of the analysis allow designers and developers not only to quickly determine the timing correctness of the processing requirements, but also to get a first view of the timing behaviour of the application implementation, its future upgrades and modifications.

SCAN employs a practical and systematic framework for representing real-time applications based on tabular descriptions. In addition, SCAN can cooperate with an event-
trace software monitor, which allows a better characterization of the actual timing behaviour of the analyzed real-time application.

The current research and development efforts, in the context of OMI/ANTI-CRASH, are oriented towards the embodiment into SCAN of certain safety analysis methods (such as the Fault Tree Analysis technique [[3]]), which focus on the identification of hazards and the assessment of the propagation of failures through a software system. In this way, the tool will provide real-time engineers with a high confidence level for guaranteeing the correctness of both timing and logical behaviour of a real-time application.

References


