Proceedings of the IEEE Workshop on Real-Time Applications, New York, USA, pp. 50-53, May 13-14, 1993.

TRACS: A Flexible Real-Time Environment for Traffic Control Systems[1]

P. Ancilotti, G. Buttazzo, M. Di Natale, M. Spuri[2]

Scuola Superiore S.Anna via Carducci, 40 - 56100 Pisa (Italy)

Abstract

In this paper we present a general architecture for a real-time system dedicated to traffic control applications. The proposed system is being developed within the TRACS Esprit III project, and applied to a vessel traffic control application. In this work, we focus our attention on the hard real-time aspects of the system, by addressing the design issues of the operating system kernel for achieving predictable and reliable behavior even in worst case conditions.

1. Introduction

Modern VTS's (Vessel Traffic Systems) are being used in many big harbours all over the world. They are especially used to collect data for traffic management, information service, navigational assistance service and support for other activities. However, one of the main goals of these systems is always to achieve safety, efficiency and protection of the environment.

The goal of the TRACS project is to investigate how advanced functionalities can be added to complex traffic control systems, such as VTS's. This implies the specification and the implementation of a prototype of a VTS, having the characteristics and the features described in the following.

Basically, the system must be able to collect data from several sensors, and to globally control the navigation of vessels in a harbour area and/or along narrow channels. Due to their capacity of operating in all weather conditions, the choice of radars as sensors is quite natural. Furthermore, modern advances allow us to extract from radar images several parameters like shape, area, position of the ship, etc.

The environment is supposed to be known by the system, which has geographical maps of the harbour area. Some assumptions are made on the scenario too, that is, the minimum and the maximum values for the ships size are assumed as well as the maximum traffic density, which specifies how many ships, buoys, wharves, etc. can be at most in a given area. For safety reasons, the VTS must be able to control the width of the ships paths, the ships speed, the cross distance between two ships and the distance of the ships from their anchorage points. All these values must be kept in certain specified ranges.In particular the software must deal with and predict for each vessel: the Estimated Time of Arrival (ETA), the path through the designated channel lanes, the Closest Point of Approach (CPA) and the Time to Closest Point of Approach (TCPA).

The control center must generate alarm signals when dangerous situations are detected. The nature of these situations and the main goal of avoiding accidents and especially loss of human lives, give rise to stringent deadlines for system alarms. To correctly manage these deadlines the application is supposed to run on a suitable hard real-time platform. Furthermore the control system has to handle ship guidance, stranding avoidance, moored ships monitoring, intrusion avoidance.

In order to obtain a more reliable and dependable application, several radar sensors are used to scan the same harbour area. The control of the maritime traffic is done through a multiradar tracking which is achieved carefully applying data fusion algorithms to the incoming data from the sensors. That is, all the information coming from the radar sensors and the a-priori knowledge of the scene have to be used in order to perform a higher level scene comprehension and a trajectory prediction, successively used for dangers or anomalies detection.

Two other higher level functionalities are required for the control center: the scene presentation and the data base management. All the information collected by the

[1] This work has been supported in part by EEC ESPRIT III and by MURST 40% of Italy

[2] This work has been supported in part by the IRI of Italy

0-8186-4130-4/93 $03.00 © 1993 IEEE

multiradar tracking has to be presented in a suitable graphic way, allowing the operator to control the whole harbour area through a clear image available even in critical conditions. In addition, any information regarding the traffic in the harbour, i.e., the ship information, has to be collected in a data base. Another data base is also necessary to collect all the a-priori knowledge of the environment, i.e., the maps of the harbour area.

2. Overall system architecture

Some of the operative requirements and of the functional specifications of the TRACS project have had a strong influence in the choice of the overall system architecture. In particular, the following characteristics have been considered:

a) the introduction of image processing techniques to the radar signals in order to extract significant features of the objects under control,

b) the need of implementing hard real-time features in order to control dangerous situations that can arise and that may imply damage or loss of goods and injury to people if not appropriately handled,

c) the usefulness of implementing a sophisticated man-machine interface in order to visualize the traffic scenario and to allow the operator to easily understand the scene and to immediately detect dangerous situations.

Taking into account the different computational needs derived from these operative requirements, the proposed computer architecture has been decomposed into two different logical subsystems: the first one (see point a) dedicated to the processing of the radar signals and to the implementation of the image processing techniques (Data Processing Subsystem); the other (see points b and c) responsible for processing data coming from the Data Processing Subsystem, for controlling dangerous situations, and for presenting the scene on the operator console (Console Subsystem). This logical modularization of the system corresponds also to a physical decomposition. In fact, keeping in mind both the physical locations of the sensors and of the console, and the functions of the two subsystems, a distributed architecture seems to be suitable.
The activities of the Console Subsystem can be carried out by two processing units:

- a Real-Time Control Subsystem (RTCS).
- a User Subsystem (US).

The overall system architecture is shown in figure 1. The RTCS is in charge of traffic monitoring and control, and includes typical functionalities of a hard real-time system. The functionalities of the US are those typically found in a time-sharing system. In this paper we focus our attention on the Real-Time Control Subsystem. The architectural specification of the RTCS directly comes from the tasks assigned to this subsystem.

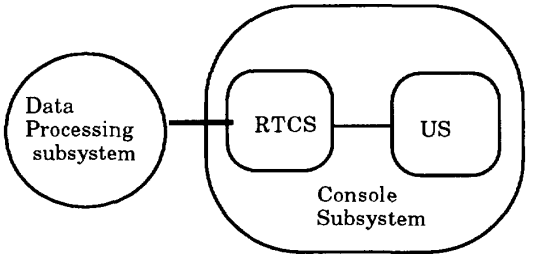

Figure 1: The TRACS architecture.

The main goal of RTCS is to keep the harbour area under control in order to avoid dangerous conditions. Synthetic data of the objects detected by the radar sensors are extracted the Data Processing Subsystem and sent to the RTCS. Using these data, RTCS performs multiradar tracking and data fusion in order to detect dangerous conditions and generate warning and alarm signals.

The RTCS must be equipped with a hard real-time operating system able to support the execution of application processes with explicit time constraints specified in terms of both hard and soft deadlines. This subsystem must be able to run *hard* real-time processes whose goal is to control the environment and generate alarm signals when dangerous situations are detected. Concurrently, the RTCS is supposed to run also *soft* real-time processes, which are activities causing no catastrophic consequence in case of missed deadlines.

3. Real-Time Control Subsystem

When dealing with time critical applications, real-time computing is not equivalent to fast computing. In fact, whereas the objective of fast computing is to minimize the average response time of a given set of tasks, the objective of real-time computing is to meet the individual timing requirement of each task [Sta88]. Therefore, rather than being fast, a real-time system should be predictable.

Within TRACS project, we want the timing properties of tasks to be easily predictable and testable, not only for the prototype, but also for more complex applications, when the number of tasks is expected to grow together with the number of the sensors.

From the analysis of the system specifications, the underlying operating system developed for this project has been designed to:

- deal with periodic tasks for data acquisition and control activities;
- satisfy and guarantee stringent timing constraints for critical activities, such as alarms and safety condition monitoring;
- allow to directly express timing constraints in absolute time units, rather than process priorities;
- deal with tasks of different nature, to integrate real-time and non real-time activities.

Such characteristics are being developed, as a hard real-time layer, on the CHORUS operating system [Cho89]. Within our project, the system interface allows the programmer to express deadlines or periods explicitly, and provides support for a predictable scheduling of time constrained tasks. Moreover, to obtain a predictable behavior, a guarantee routine analyzes the feasibility of the scheduling every time a critical process is activated, so that a possible time-overflow can be seen in advance, and hence avoided.

Based on the environment characteristics considered in TRACS, the scheduling policy has been designed according to the Rate-Monotonic algorithm [Liu73]. The Rate-Monotonic algorithm can be easily implemented in a priority-based kernel, such as the CHORUS kernel, without imposing high overheads.

In order to deal with different classes of activities that may be created in a real-time control application, such as a traffic control system, the modified kernel has been designed to deal with different scheduling algorithms for handling four types of tasks:

- periodic processes, with critical deadlines; for instance, periodic tasks checking for the existence of life-critical conditions and alarm generation;
- sporadic processes, with irregular arrival times and critical deadlines; an example is given by triggered alarms that need to be activated only once;
- soft tasks, with non critical time constraints, as tasks devoted to periodic warning signals or task performing non-critical periodic data-fusion algorithms;
- non real-time tasks, both periodic and aperiodic, with no time constraints at all; like monitoring tasks, or tasks for data-base inquiring.

Each task is characterized by a number of additional parameters to those required by the commercial CHORUS kernel, in order to specify its type, its periodic or aperiodic nature, its timing constraints, or its priority if non real-time, and its estimated maximum execution time. These parameters feed a guarantee mechanism, used to achieve a predictable scheduling of time critical tasks.

To guarantee execution time for sporadic activities, a server mechanism has been used, which periodically reserves some processor time for sporadic activities according to process requests and their minimum interarrival time. The chosen mechanism is based on the *Sporadic Server*, proposed by Sprunt, Sha and Lehoczky [Spr89], that, under the Rate-Monotonic scheduling policy, improves the response times for soft aperiodic tasks and can guarantee hard deadlines for both periodic and aperiodic processes.

The minimum interarrival time of each sporadic process can be computed considering the physical restrictions of the objects under control. For example, the minimum interarrival time of a process handling alarms related to a restricted area can be computed considering the minimum time required by any ship to exit and enter the restricted area.

Soft and non real-time processes can be handled by assigning them a priority lower than the lowest priority of time critical processes. This guarantees that soft and non real-time processes have no disturbing effect on the scheduling of time critical processes.

An important feature that our predictable real-time system provides for supporting a critical application is the capability of guaranteeing the execution of all time critical activities. The guarantee for critical task executions is based on the knowledge of their worst case computation time. Schedulability analysis in presence of shared resources is also performed by evaluating the worst case blocking time of each task. Blocking time in critical sections is bounded using monitors together with the Priority Inheritance Protocol [Sha90]. Within TRACS project a tool for the static analysis of the timing properties of the application code is under development.

Since TRACS application requires both real-time and non real-time processes, the kernel should provide synchronous and asynchronous time bounded communication primitives to adapt to different task requirements. However, synchronous interactions should be avoided among critical tasks, unless they are time bounded, since waiting for a message to arrive may introduce an unpredictable delay that may cause a task to miss its deadline. Non blocking primitives with overwrite capability have to be used in the communication among critical tasks whenever the most recent data is of primary importance, as in the case of sensory data collection.

Asynchronous primitives allow the sender to deliver messages without waiting for delivery confirmation. The message is read when present, otherwise a specific code is returned to indicate message absence. To preserve the last arrived message for every receiver request, at any time, messages are not consumed by the receivers but remain "stuck" to the port until they are overwritten. The implementation of this semantics is realized providing new primitives that allow all the features described so far. Non real-time tasks can use remote procedure calls to require services to other tasks, for example to inquire local database servers, or to request services to remote nodes.

Allowing each device to interrupt the cpu at any time, may introduce unpredictable delays at run-time, which may cause a critical task to miss its deadline. The drivers associated to the I/O devices of the Real Time Console, are written and scheduled using the mechanism that CHORUS provides to activate user tasks when interrupts are detected by the system. To guarantee deadlines for time critical I/O operations, I/O handling processes must be scheduled as ordinary threads with the assigned deadlines or priorities, as for sporadic tasks, according to the application requirements.

To avoid unpredictable delays on memory accesses, a fixed memory management scheme is used within hard type tasks. Such policy requires that the maximum amount of memory used by tasks is bounded and defined off-line. A pre-paging allocation policy can be used to guarantee memory requirements for all time critical tasks locking in main memory their pages, while soft and non real-time tasks share the remaining memory space in a conventional way.

4. Conclusions

In this paper, the application requirements of a vessel traffic control system have been described. An architectural solution is then discussed. In particular, we have focused our attention on the real-time aspects of the system, especially those concerning the scheduling algorithms of time critical tasks and inter process communications. Some modifications of an existing priority based kernel (the Chorus OS) are proposed to achieve a predictable and reliable system.

In our description all real-time activities run on a single real-time console. The implementation of a full real-time distributed system would need the development of ad hoc scheduling algorithms and network protocols because of the lack of standardized commercial systems. However, the adopted solution seems to be reasonable in order to investigate the proposals of the TRACS project.

5. References

[Cho89] M. Rozier, et al: "CHORUS Distributed Operating Systems", Technical Report CS/TR-88-7.8, Chorus Systemes, Saint Quentin en Yvelines Cedex, France.

[Liu73] C. L. Liu, and J. W. Layland: "Scheduling Algoritms for Multiprogramming in a Hard-Real-Time Environment", *Journal of ACM*, Vol. 20, No. 1, pp. 46-61, January 1973.

[Sha90] L. Sha, R. Rajkumar, and J. P. Lehoczky: "Priority Inheritance Protocol: An Approach to Real-Time Synchronization", *IEEE Transactions on Computer*, Vol. 39, No. 9, pp. 1175-1185, September 1990.

[Sta88] J. A. Stankovic: "A Serious Problem for Next-Generation Systems", *IEEE Computer*, pp. 10-19, October 1988.

[Spr89] B. Sprunt, L. Sha, and J. P. Lehoczky: "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", *Technical Report*, CMU/SEI-89-TR-11, EDS-TR-89-19, Software Engineering Institute, Carnegie Mellon University, Pittsburg, Pennsylvania, April 1989.