MODELLING REAL-TIME CONSTRAINTS

S.J. Berryman and I. Sommerville

Lancaster University, UK

ABSTRACT

The objective of the work described here is to provide a software tool to assist real-time system specifiers and designers to predict, at an early stage of the development process, the timing behaviour of the system developed. Our tool (Simulation of Real-time systems (SRT)) is used to model the timing aspects of a real-time system and then simulate the system to predict its behaviour.

INTRODUCTION

A real-time system is expected to interact with the environment within certain timing constraints and the software designers must produce a system which can guarantee to meet these constraints. A realistic real-time system will be composed of many interacting modules which could be executed with real or virtual concurrency. Real concurrency means each module executing on its own processor and virtual concurrency is where many modules (processes) are executed on one processor, Allworth [1]. There can also be a combination of these, where many processes are distributed between many processors. The smallest real-time systems can, therefore, present considerable timing problems which could be greatly reduced by using systematic methods supported by software tools, Orr et al [13].

Our tool SRT (Simulation of Real-time systems), allows a model of a real-time system to be constructed and then evaluated by simulation. The construction of the model is achieved by using a graphical user interface (Figure 1). Icons from the control panel are copied onto the design panel and then joined together with lines that represent the databuses. Each icon has a number of attributes which can be specified by selecting the form option on the icon menu.

This paper presents an overview of the system which is implemented using Smalltalk-80, an object-oriented programming language. The language was chosen because of its rapid prototyping ability, thus ideas can be implemented and tested very quickly.

The paper is split into five sections. The first section looks at current tools and methods for modelling timing constraints. The second and third sections explain a real-time system whose timing characteristics can be modelled using SRT. The fourth section describes how SRT evaluates the model constructed in the previous two sections and analyses the output data. The fifth section looks at future versions of the tool and draws some conclusions about our work.

RELATED WORK

There are various tools and methods for prototyping, formally specifying and implementing real-time systems. This section looks briefly at each area and concludes by showing where SRT fits in.

Rapid prototyping tools can be seen by the designer and user as a way of viewing the product at the design stage. The user can determine whether or not the prototype is appropriate and, if not, the appropriate alterations can be made, thus avoiding wasted effort in the implementation of incorrect requirements. The computer aided prototyping system, proposed by Luqi et al, [8, 9, 10, 11] is a rapid prototyping tool for real-time system prototyping. The system is composed of a software base which contains a number of components. These components are explicitly given a set of timing constraints. The tools provides help in constructing a prototype and determines its real-time performance by the constraints placed on each component.

Formal specification of real-time systems should enable the system designer to verify mathematically that a system will meet all its deadlines. However, formal methods are still immature for software systems and introducing the notion of time makes the issue of verification even more complex.

Various specification languages for real-time systems are emerging but major restrictions are placed on the real-time systems to reduce the complexity. For example, the specification language RT-ASLAN, Auernheimer and Kemmerer [2], makes the assumption that each process is on a dedicated processor, thus avoiding any scheduling issues.

Lauber's [5] approach is based on the fact that the design of large complex real-time systems will use a specification language. If timing constraints can be expressed within the specification then it can be used to test the behaviour of the real-time system. The specification model represents a rough prototype which is evaluated. The methods used are based on dynamic testing of the specification with data being fed in by the system's designer or a simulation of the real world.

Some real-time languages support facilities for the representation of time. Euclid, Kligerman and Stoyenko [4], is capable of expressing timing constraints but at the cost of restricting other features which are unpredictable, such as dynamic process creation and recursion. The language supports exception handling, so that the appropriate action can be taken when deadlines are missed. The language FLEX, Lin and Natarajan [6], supports facilities for imprecise computing, which is a method of arriving at an approximate value and then improving the result as much as time permits.

All the approaches described above have a particular place in the software development cycle. The prototyping approach concentrates on modelling all the requirements. To build a complete prototype the design would need to be decomposed into modules and then these modules implemented in the prototyping language. Thus, to determine the feasibility of the timing constraints from this method would require a complete prototype of the system to be built.

Current formal methods are not mature enough to represent the complexities of realistic real-time systems. Lauber's method involves a detailed specification, much of which is irrelevant when dealing with timing issues. The real-time languages offer the facility to express and evaluate timing constraints but determining the feasibility of these constraints is necessary well before the implementation stage.

All these methods are concerned with modelling both the functionality and timing aspects of the systems, whereas SRT deals only with the timing aspects. The system designer can therefore model the timing behaviour before significant implementation effort is devoted to system prototyping.

MODELLING TIMING CONSTRAINTS

Real-time systems can be considered, at an abstract level, to consist of three basic entities namely sensors, processes and actuators. SRT uses these *real-time entities* to model the process structure and proposed hardware devices. The process structure is made up of a fixed number of processes, because dynamic process creation causes unpredictability. The time constraints, which the software designers must adhere to, will be defined by the system designers. These timing constraints will be determined by the environment, so that the systems designer has to calculate how quickly the system must respond to changes in the environment.

This section presents an example of a temperature monitoring system with real-time constraints, to illustrate the capabilities of SRT.

Temperature Monitoring System

A real-time system is required to monitor and control the temperature of a piece of equipment. This system must respond to any change of temperature which is outside the allowed range within the defined timing limits.

The proposed system requires three processes, each with a number of associated sensors and actuators connected to them. The sensors measure the temperature, convert the analogue signal to digital and relay the information to the process. The actuators are cooling devices which can be switched on or off by the process.

ProcessA maintains the average temperature of the whole piece of equipment. The process has three sensors and two actuators (cooling devices) connected to it. When the average temperature goes past a particular point then the actuators change state (on or off). The system designers have determined that this process is required to sample the sensors 400 times a second, so that the required temperature is maintained. ProcessB and C are lower priority processes which monitor a subsection of the equipment. Table 1 defines the specification of all three processes.

TABLE 1 - Temperature Monitoring System's process requirements

Process Label	Priority	Rate	Success Rate
ProcessA	1	400	100%
ProcessB	2	250	80%
ProcessC	3	250	50%

ProcessB and C must succeed to meet their deadlines 80% and 50% of the time respectively because processA can deal with any serious temperature changes. ProcessA must always meet its deadlines, failure to do so could cause serious damage to the equipment.

The sensors are all of the same type, that is, they all take 100µs to respond to a sampling signal. All the actuators take 1 000µs to achieve the transition from one state to another. It has been determined that the actuators on average switch 20 times per second. ProcessA has a rate of 400, thus on average the actuator is activated 1 in 20 samples. This can be modelled by having a probability which has a uniform distribution of 1 to 20. ProcessesB and C on average send a signal to the actuator every 1 in 12.5 samples. This is rounded down to 1 in 12 as it is better to overestimate and leave space for error, than underestimate.

The buses all take one unit of time to transmit and receive the required information. The scheduler has an overhead of 10 units. The currently proposed processor has a clock speed of 0.1 MHz (100 000 units per second).

SRT can model this timing information and evaluate the prototype by simulation to determine whether or not these processes can meet the set deadlines with the proposed hardware.

Constructing a Prototype

This section describes how a real-time system prototype can be constructed using the SRT tool. The first part of this section describes the types of timing attribute associated with the entities. The second part looks at each entity type and the attributes associated with them.

Timing Attributes. Each of the three Real-Time entities; sensors, processes and actuators, have a number of attributes. These define the timing requirements of the prototype and are of two types:

1) Actual Time
This timing attribute represents the actual measurement of time in the real world which is typically used to simulate a delay for an IO device. The actual time in SRT is specified in micro seconds.

2) Unit Time
This is the number of units (clock cycles) a process requires from the processor to simulate the execution of a piece of code. This type of timing is processor dependent, so that changing the processor speed will result in the unit time taking different values of actual time.

Real-time entities. SRT consists of five RT entities, which are all represented as icons on the simulation control panel (Figure 1). Sensors, processes and actuators are used as the basis for constructing the real-time prototype. Two other processes, scheduler and background are special types of process that must always be present. Entities are connected by databuses, which are represented as arrows on the simulation control panel.

Each real-time entity has a label attribute, so that they can be distinguished. The following explains other attributes associated with each entity and relationship (databus).

1) Sensors
There are two types of sensors, event driven and polling.

The polling sensor device receives a signal from a process and then replies with its current state. As shown in figure 2, the sensor has two timing attributes, sensor time and minimum polling rate. The sensor time attribute is the actual time in micro seconds that a sensor takes to receive a signal from the bus, convert the sensor reading from analog to digital and put the result back onto the bus. The second attribute sets out how often the sensor should be polled per second.

Event driven sensors are modelled by using probability distributions. Both timing attributes are left blank and the arrival rate of events to a process is specified on the incoming databus.

2) Actuators
A process can trigger an actuator in one of two modes, suspended or continuous. In suspended mode the process sends a signal to the actuator and waits for a reply (suspends) before it continues. The continuous mode does not wait for a reply from the actuator but signals to the actuator and then continues. This mode may result in a queue building up on the actuator if the process puts out more signals than the actuator can consume. The importance of this varies depending on the device. For example, if the actuator is a display, then losing a few signals may be acceptable but in other situations a build-up of signals would be disastrous.

The actuator has two timing attributes (Figure 3), actuator time and code time after action. Actuator time is the actual time it will take the device to complete its task. Currently this time is static, so that the actuator always takes a set amount of time. Realistically, a device may vary in time depending on its current state, therefore a time range based on a probability distribution should be used. The 'Code time after action' attribute is expressed in unit time and will be explained in the process entity section.

3) Buses
The databuses that connect the various entities also have timing attributes. The direction of the arrow indicates the flow of control. In figure 1 the sensors are being polled. Figure 4 shows the form for the databus. The first two attributes specify the unit time it takes the information to travel to and from the two connecting entities.

The buses are also used to specify any probability distributions between two entities. This can only be set between event sensor and process or process and actuator. The bus from the event driven sensor can be used to signify the arrival of events into the process. Figure 4 shows the different distributions available. The other values are the parameters for the distributions; for example normal distribution will have two parameters, mean and deviation.

4) Processes
A process can be of two types depending on the type of sensors connected to it, periodic or sporadic. A periodic process is one which occurs at regular intervals to sample some devices and, if required, send a signal to the actuator. Within SRT a periodic process is one which only has polling sensors connected to it. A sporadic process has a number of event driven sensors associated with it. This type of process only comes into action when an event occurs.

Either process when executing will receive data from its sensors (either by polling, or interrupt), do some processing and finally, if required, send a signal to the actuator. A process will suspend when waiting to receive data from a sensor. During the execution of code it is possible for another process with a higher priority to pre-empt it.

Figure 5 is a process activity view, which enables the software designer to define the process's characteristics. The top panel contains the process's attributes. The rate attribute defines how often this process should be executed per second. The priority attribute determines the importance of the process. The middle panel contains a number of selection list views which are used to describe the process's minor cycles. Figure 5 shows that processA firstly executes some code to start up, then sensors1,2 and 3 are polled with a small amount of code executed in between them. The following code simulates the process to calculate the average temperature of the three sensors. Finally the actuators are activated.

There may be situations when a process need not send a signal to an actuator every time. In the temperature monitoring system, ProcessA changes the actuators state once in every 20 samples. The bus running from the process to the actuator can be used to specify a particular probability distribution which determines how often the process sends to the actuator a signal. The actuator attribute 'code time after action' specifies in unit time any extra code that may have to be run before the actuator is kicked off.

The information on the bottom of the panel is calculated from the minor cycle of the process. The figures show the amount of actual time a process will take, the amount of processor units required and finally the number of times this process could feasibility be executed per second if it had sole access to the processor. This figure should be significantly greater than the rate specified at the top of the panel.

The minor cycles for ProcessB and ProcessC follow the same lines as ProcessA.

5) Background process
This process is executed only when nothing else can be run. Generally this type of process in real-time systems will carry out checks on the hardware. No attributes are associated with this process.

6) Scheduler
Currently only one scheduling policy has been built into the system, which is a fixed priority scheduling algorithm (rate-monotonic) , as described by Liu and Layland [7]. Each process is given a priority, 1 being the highest. Priorities should be assigned using the rate-monotonic priority assignment rule, in which a higher request rate determines a higher priority.

The scheduler decides what process to run by the priority and state of the process. A process can be one of four states: Ready, Running, Suspended (blocked) and Dormant. Most operating systems will support the first three, the dormant state being associated with real-time operating systems. The dormant state is when a periodic process has completed its major cycle and is now waiting for its next occurrence.

Figure 6, shows the two timing attributes associated with the scheduler. The first attribute is to set the speed of the processor, that is, how many units or clock cycles can run in a second. This value sets a relationship between the actual time values and the unit time. This following formula is used by SRT to convert actual time into unit time.

unit time = (10^6 / actual time (μs)) /
processor speed.

For example, the following calculates the unit time from an actuator which has a delay time of 1 000μs and a processor running at 0.1 MHz.

1 000 000 / 1 000μs = 1 000 per second
1 000 / 100 000 = 100 units
therefore 100 units = 1 000μs

The overhead attribute specifies the amount of processor time the scheduler will take up when swopping two processes. The overhead value is in unit time and this code cannot be pre-empted.

Representing Timing Constraints

SRT has two types of timing constraints, process deadline and device deadline.

Process Deadline

Each process has a frequency rate which implicitly defines its deadline. The process must be completed before it is next scheduled otherwise an overload will occur. The scheduler deals with overloads by aborting the process so that it can be restarted on its next occurrence. Euclid [4] provides an exception handling mechanism for determining the action of a process when it fails to meet its deadline. As other scheduling polices are implemented, so mechanisms for representing timing delays for exceptions raised by missed deadlines will be considered.

When making timing estimates of the processing required, the type of process should also be considered. Hard time processes which are required to always meet their deadline should be calculated using the worst-case run times, whereas soft processes should be calculated with an average run time, Burns and Wellings [3].

Device Timing Constraints

These are constraints placed on the system by the system designer when determining the sampling and action rate for the system to function within the requirements. The computer system must be able to meet these constraints for the system to function correctly.

The sensor has the attribute minimum polling times. After the simulation the summary information will show whether or not the sensors have been polled enough times.

The actuator constraint determines how often, in actual time, the actuator must be updated. Currently SRT displays for each actuator the amount of times an action has been completed and whether or not a queue exists

EVALUATING TIMING CONSTRAINTS BY SIMULATION

Once a model of the real-time system has been constructed then the timing constraints can be tested by simulation. The simulation is executed for the unit time specified by the user. If the simulation is executed in the trace mode then every process reports its progress at every stage.

Every process (including scheduler and background) has a text window which displays its actions and the processor time at which the action was started. Figure 7 shows the scheduler, background and ProcessA text windows. The scheduler reports to the text view every time processes are swopped and the background process reports the range of times it was being executed. In figure 7, the process view shows part of the activities of processA during the simulation. On the start up of the 400th cycle the processor executes for 15 units, 5 units for process code time and 10 units for the scheduling overhead (99751-99765 inclusive). This is followed by the process suspending while sensor1 is being polled. Sensor1 causes ProcessA to suspend for 12 units, 2 for data transfer and 10 for the sensor delay. This is then repeated for the other two sensors. The final piece of code runs for 50 units, 40 units for the code and 10 units for scheduling. This cycle does not need to activate the actuators. After completion of the cycle the process will lie dormant until its next occurrence.

The simulation is terminated when the processor time reaches the amount defined on the simulation panel.

Analysing a process's characteristics

A summary of the process activities is displayed in each text view. This information can be used to determine which constraints might be modified to improve performance.

During the simulation each process keeps a record of the time spent in the four states, which are displayed graphically (Figure 8). Each process calculates how often it misses its deadline. All hard real-time processes should have a 100% success rate.

The background process summary displays the percentage of time it was executed. This figure can give an idea on the amount of processing time that could still be used. The scheduler summary displays the amount of actual processing performed; the rest of the time was taken up by the scheduler process.

The process summary includes a list of all the sensor and actuator devices. Each device supplies information on how

often it has been activated, so that the figure can be used to check if the device timing constraints have been met. Each sensor connected to the process displays how many times it was polled and whether or not it satisfied its device timing constraint. The actuator displays how often it was activated and how many signals, if any, were waiting on the queue. The summary information shown is after one second of simulation time (100 000 units); because of the cyclic nature of real-time systems, running it for any longer period of time would probably not make any significant difference.

From the summary information each processA, B and C met their deadlines 100%, 98.8% and 58.8% of the time respectively, which is within the requirements. If any changes were made to the requirements then the software designer would need to know if all the timing constraints could still be met. If, during the development cycle, it was realised that another sensor connected to processB was required, then the initial SRT model could easily be adapted and then re-evaluated. After re-evaluating the system processesA, B and C met their deadlines 100%, 86.4% and 45.2% of the time respectively. This change in requirements has caused processC to fail to met its required 50% performance level defined in the requirements. This information is immediately available, thus allowing the designers to make early decisions regarding the timing behaviour of the system. The decisions may involve the use of faster hardware, reduction of software requirements or the use of additional processors.

A small change in the requirements can radically change the timing behaviour of the system. Thus, without the aid of software tools it becomes very difficult for software designers to predict the effect of change on the timing behaviour of a real-time system . Especially if the real-time system has many processes which are distributed over many processors.

FUTURE WORK

The initial system is being used to test and develop new ideas. This next section looks at some of the current ideas under development and also considers the long term objectives of SRT.

Short term

Currently processes only suspend when involved in IO operations, but a process may actually have to wait for the completion of another process before it can continue. Process-to-process communication is being implemented using a semaphore notation, so that one process will wait for a signal from another.

The system designer must make estimates on the execution times of processes, which will probably be far from accurate. Project managers, even after years of experience, don't seem to make much better estimates. The estimates are generally made and then multiplied by some factor determined by previous underestimates. SRT does enable changes to be made very easily, so that the consequence of improved estimates can be seen instantly. One problem with these cyclic processes is that each cycle may not take an equal amount of time, for example a process may have to do some extra calculations after some determined number of cycles. This flexibility could be modelled using a probability distribution mechanism.

Sporadic processes are to be implemented into the fixed priority scheduling policy. A sporadic process would be given a priority, as with the periodic process, but not an occurrence rate. When an event occurs the scheduler will service the interrupt and then execute the highest priority process. All emergency interrupts could be modelled in one highest priority process.

Long Term

A longer term aim is to provide modelling of multiple processors. The simulation would provide information about each processor, such as its average load balance. This would allow designers to manipulate processes between processors to determine the best configuration.

CONCLUSION

The correctness of real-time systems involves not only the verification of the logical steps of the software but the completion of the steps inside the given time limits. If the time limits are not met then the work carried out will probably be of limited use to the system and economic, human, and ecological catastrophes could result, Stankovic [14].

Tools are required which help system designers make the correct decisions by having the information available at the design stage rather than after the implementation stage, where finding a fault would be far more costly, Malcolm [12].

SRT can be used throughout the development of the real-time system, so that the consequences of improved timing estimates and additional requirements can be reflected instantly. SRT has also shown that it could be used to investigate various scheduling policies to determine the appropriate policy for a particular process configuration. SRT should help to provide information which will assist in determining the feasibility of the timing constraints at an earlier stage in the development than previously would have been possible.

ACKNOWLEDGEMENT

The authors wish to thank Paul Kearney, Bill Jones and Morag Kellaway from British Aerospace for their constructive ideas in the development of the current version of SRT.

REFERENCES

[1] S. T. Allworth, 1981, "Introduction to real time software design", MacMillan

[2] B. Auernheimer and R. Kemmerer, 1986, "RT-ASLAN: A Specification Language for Real-Time Systems", IEEE Transactions on Software Engineering, 12(9), 879-89

[3] A. Burns and A. Wellings, 1990, "Real-Time Systems and their programming languages", International Computer Science Series

[4] E. Kligerman and A. D. Stoyenko, 1986, "Real-Time Euclid: A Language for Reliable Real-Time Systems", IEEE Transactions on Software Engineering, 12(9), 941-949

[5] R. J. Lauber, 1989, "Forecasting Real-Time Behaviour During Software Design using a CASE environment", The Journal of Real-Time Systems, 1(1), 61-76

[6] K.-J. Lin and S. Natarajan, 1988, "Expressing and Maintaining Timing constraints in FLEX", IEEE, 96-105

[7] C. L. Liu and J. W. Layland, 1973, "Scheduling Algorithms for Multiprogramming on a Hard-Real-Time Environment", Journal of the Association for Computing Machinery, 20(1), 46-61

[8] Luqi, 1988, "Knowledge-Based Support for Rapid Software Prototyping", IEEE Expert, 9-18

[9] Luqi and V. Berzins, 1988, "Rapidly Prototyping Real-Time Systems", IEEE Software, 25-36

[10] Luqi, V. Berzins and R. T. Yeh, 1988, "A Prototyping Language for Real-Time Software", IEEE Transactions on Software Engineering, 14(10), 1409-1423

[11] Luqi and M. Ketabchi, 1988, "A Computer-Aided Prototyping System", IEEE Software, 66-72

[12] B. Malcolm, 1989, "A Large Embedded System Project Case Study", Software Engineering for Large Software Systems, Elsevier Applied Science,

[13] R. A. Orr, M. T. Norris, R. Tinker and C. D. V. Rouch, 1988, "Systematic method for realtime system design", Microprocessors and Microsystems, 12(7), 391-395

[14] J. A. Stankovic, 1988, "A Serious Problem for Next-Generation Systems", IEEE Computer, 10-18

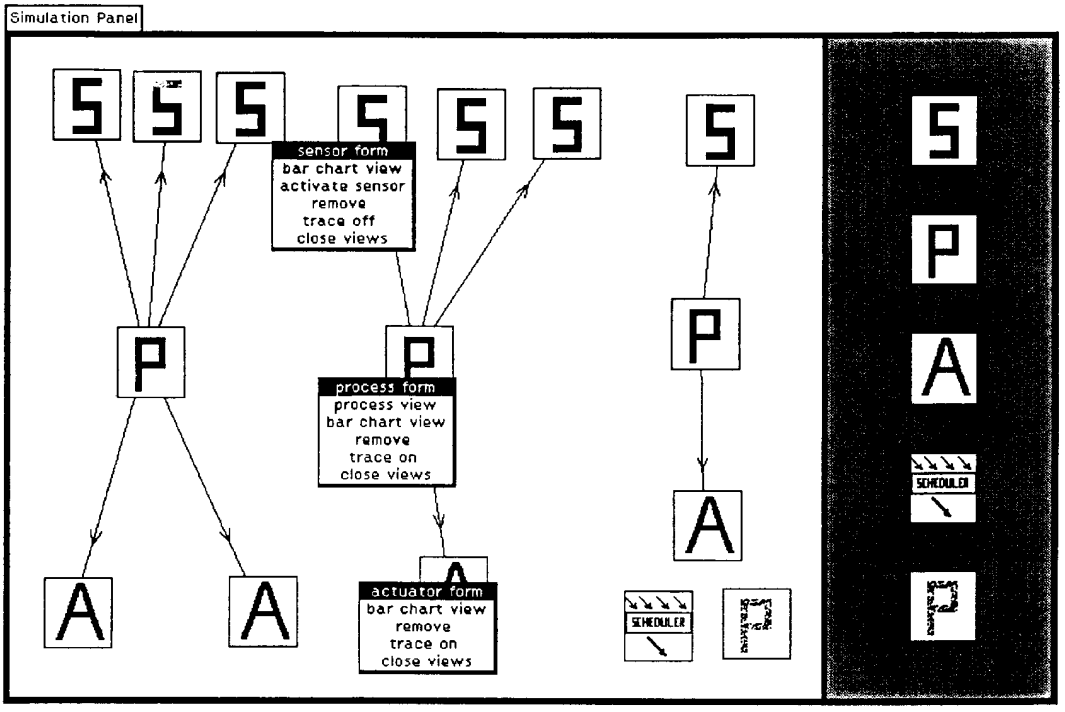


Figure 1 Simulation Panel

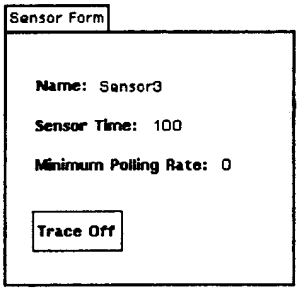


Figure 2 Sensor Form

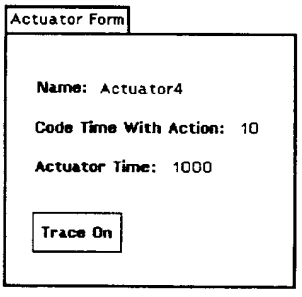


Figure 3 Actuator Form

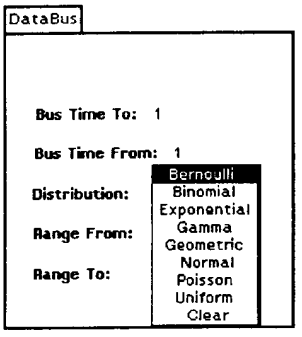


Figure 4 Databus Form

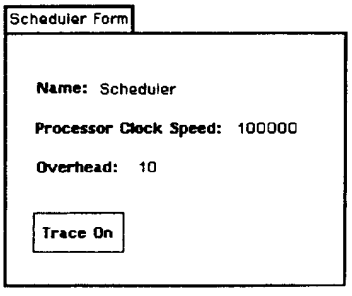


Figure 6 Scheduler Form

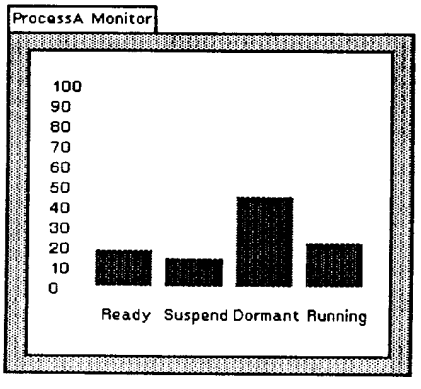


Figure 8 Summary of ProcessA

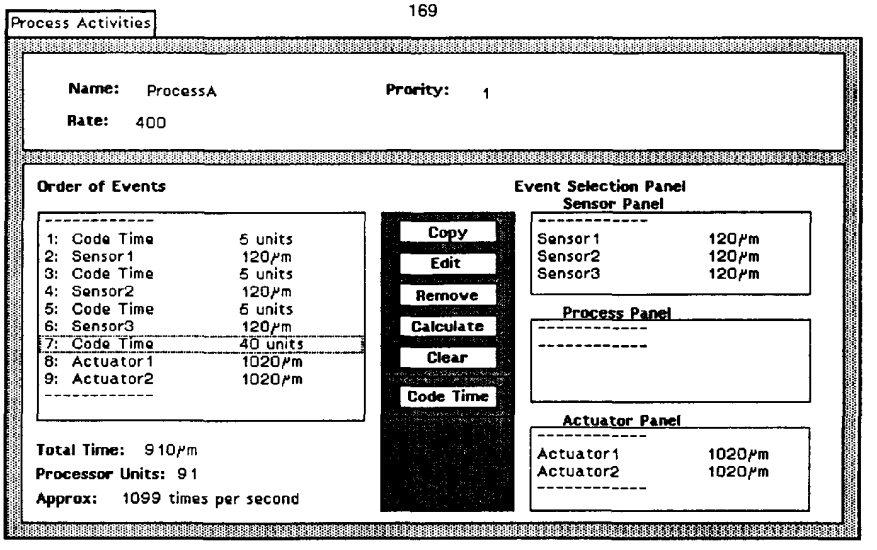
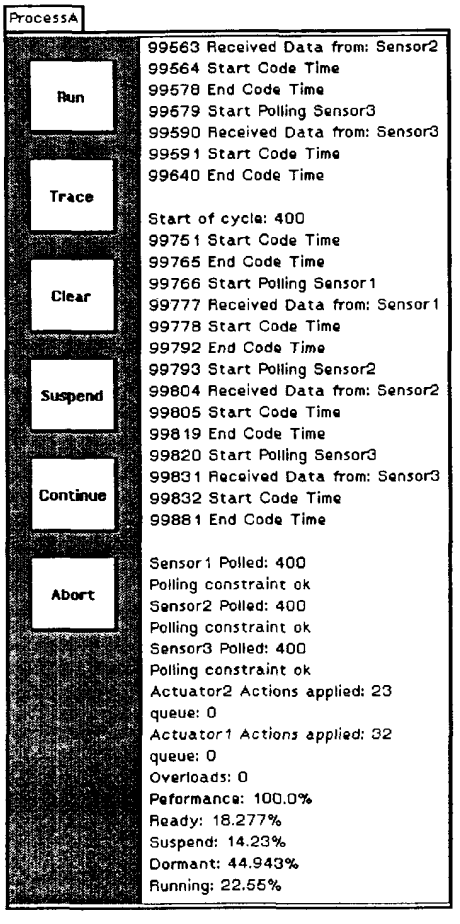
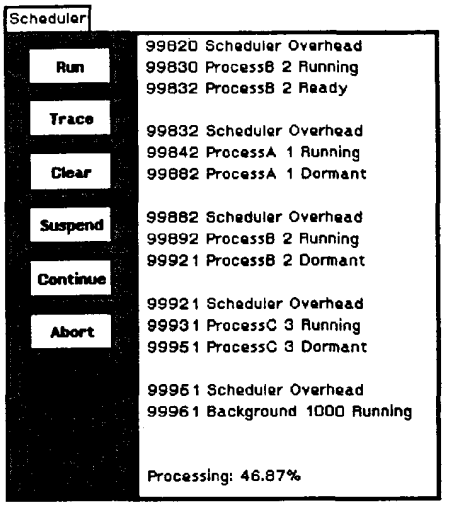


Figure 5 Process Activity View





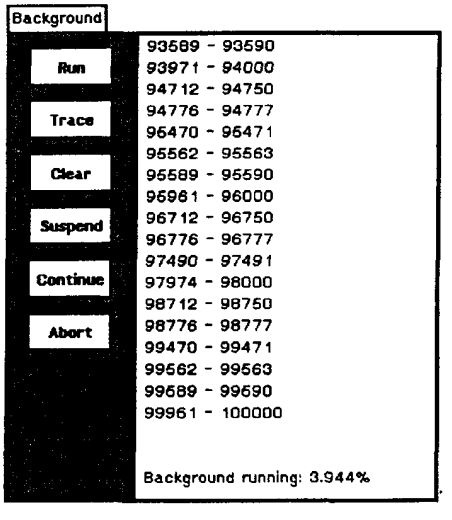


Figure 7 Process Text Views