An Architecture for Intelligent Task Interruption

D.D. Sharma and Srini Narayan
Corporate Technology Center
FMC Corporation
1205 Coleman Ave
Santa Clara, CA 95052

Abstract

In the design of real-time systems the capabiltiy for task interruption is often considered essential. In this paper we examine the problem of task interruption in knowledge-based domains. We propose that task interruption can be often avoided by using appropriate functional architectures and knowledge engineering principles. Situations for which task interruption is indispensable we describe a preliminary architecture based on priority hierarchies.

1 Motivation

Real-time systems require that decisions be made dynamically in response to incoming sensor data with desired responsiveness within the specified time deadlines. Often real-time solutions should degrade gracefully. Knowledge-based real-time systems perform several functions such as sensing information, processing information to determine the situation status, evaluating the situation, plan for needed actions, and executing the planned actions while monitoring the status of the plans, the system, and the situation. In such systems time stress arises from sources external (in terms of high data rate, unanticipated demands on problem solving, and unexpected deadlines) and internal (knowledge based processing is computationally intensive and the computation time is often unpredictable) to the knowledge based system. All practical systems have limited resources and given the unexpected demands requires intelligent resource management schemes.

Resource management involves making resources available to tasks that need them and reclaiming resources from tasks that do not need them. To accomplish resource management, amongst other things, the capability to interrupt tasks, suspend execution, and later resume execution may be needed. Given the tasks to be interrupted, real-time operating systems provide efficient low level mechanisms to implement task interruption. In knowledge-based systems the determination of the task to be

interrupted has to be inferred in runtime and is based on the specific context. Runtime reasoning to determine which task to interrupt and is computationally expensive and alternative solutions are desired.

2 Issues of Task Interruption in Real-Time Knowledge-Intensive Architectures

An hospital is something all of us can relate to, therefore, following [Hayes-Roth 1987] we use a hospital situation as an example to describe issues of task interruption. Hospitals are well designed functional structures, they have finite resources (heterogeneous), perform various tasks, and respond to time critical situations. The function of a hospital is to provide health maintenance services to the community it serves while controlling the cost of providing such services. A hospital situation has the following characteristics relevant to task interruption.

1. Finite Resources: Resources of an hospital are its physicians, surgeons, nurses, pharmacy, operating theaters, operating budget, etc. The resources in each category are finite and often cannot be interchanged.
2. Environment Demands: The hospital receives a continuous stream of patients. Different patients have different treatment requirements. During certain seasons the demand for one type of treatment may increase significantly. During certain calamities the demand on certain facilities may suddenly increase. The hospital are required to adapt to such external changes.
3. Internal Demands: Apart from the constraints of finite resources the internal systems are subject to failure (e.g., strike by nurses). Thus the available internal resources are dynamically changing.
4. Emergency Situation: Not all patients are attended to within uniform time interval. Some patients make appointments. However, there is often a stream of critical patients who need immediate care. The way a hospital solves this problem is by providing an architectural solution; i.e., an emergency ward. An emergency ward is staffed and operated under a different set of resource allocation policies.
5. Categorized External Demands: Given the inflow of patients to a hospital a single receptionist desk will quickly overflow. The problem is solved by recognizing that incoming patient have different needs. Often patients themselves know the general nature of problem. Again a hospital provides an architectural solution: it has several speciality wards and screening nurses. An incoming patient can directly go to a speciality ward or go to the screening nurse from where he is directed to an appropriate speciality ward.
6. Resource Contention: At any given point in time there are only a limited number of doctors, nurses, operating theaters, and anesthesiologists available. Since hospitals attempt to minimize operating costs the resources are scheduled based on an expected resource demand profile. Whenever external demands exceed the average demand the available resources fall short of the need. This leads to contention of resources. Also internal failures can cause less than allocated resources to be available thus causing resource contention even during a normal demand situation.
7. Resource Facilitators: Certain resources such as Nurses are like resource

facilitators - they make sure that resources needed to attend to a patient are available to a doctor when needed. Also in situations of resource contention nurses or interns can relieve a surgeon for a limited period of time. Thus having resource facilitators helps in achieving graceful interruption.

8. Anticipatory Scheduling of Resources: Apart from providing architectures for efficiently applying resources, hospitals deliberately schedule resources to reduce the effects of unanticipated upsets in resource demands. Thus instead of being reactive to resource demands, the demands and the readiness of resources are anticipated based on past models and and activities are scheduled accordingly. For example, major surgeries are often scheduled in morning hours. Blood transfusion and anesthesiology is not scheduled during weekends [Hester, 1989].

9. Taxonomy of Tasks from Interruption viewpoint: A patient (i.e. a task) needs a wide range of heterogeneous services. In a hospital situation these services are provided by various specialists. To look into the problem of interruptibility instead of thinking in terms of surgeons (i.e. resources) being interruptible it is useful to think of activities being interruptible or non-interruptible. Typically, most of the activities of a physician might be interruptible. In other cases the activity need not be interrupted while in other cases the activity cannot be interrupted.

10. Context Dependency: The above example gives a default classification of activities. In reality the interruptability determination is made with respect to the context. For example, contrary to expectations a surgeon's activity is often interruptible. In fact the interruption is facilitated by the presence of numerous interns and highly skilled nurses. Depending upon the stage of the surgery a surgeon can be pulled out to assist on a relatively more complex and urgent case.

11. Task Interactions: A certain patient is undergoing a course of treatment. The patient develops a new abnormality requiring administration of new drugs. Do we stop the ongoing treatment completely? Can the new drugs unfavorably interact with the new treatment? Such issues need careful examination.

12. Task Interdependency: A patient is undergoing an intricate surgical procedure. Can the anesthesiologist be pulled away for another case? This will depend upon the stage of the surgery and the future needs of this procedure.

In summary:

1. Responsiveness and resource contention problems do not necessarily require interruption. Architectural and appropriate resource scheduling solutions are preferred over task interruption.

2. In situations of emergency (time critical situations) where the demand considerably exceeds available resources task interruption may be needed.

3. Determining interruptability of a task is highly context dependent.

3 Intelligent Task Interruption

The Task Interruption problem can be defined as follows: Given n concurrently active tasks T1, T2, ..., Tn (where each task is characterized by its resource needs and completion deadlines) and a new task T waiting for execution the following question need to be answered: Should we interrupt any of the existing tasks and which one?

Task Interruption in realistic situations requires use of situation and domain specific knowledge to determine whether and who to interrupt. To perform efficient interruption we need two things:

1. A Problem Solving model to enable deciding whether, who, and where to interrupt. The model should take into account logical and temporal dependencies between the on going activities.
2. An architecture to support efficient interruption thus addressing how to interrupt.

In this paper the architecture support will be not be discussed in detail. Various architectural designs will be briefly mentioned where appropriate.

4 Why Interrupt? or The Goals for Interruption

The problem of deciding whether to interrupt a task should be put in context of the objectives of the situation. Some of the objectives and possible solutions are discussed below.

1. **Responsiveness.** In real-time systems improving responsiveness for a certain a category of tasks is important. In conventional software systems responsiveness is achieved by sending an interrupt signal to the appropriate hardware. In AI architectures responsiveness can be improved without interruption. Following designs lead to improved responsiveness;

- Task Grain Size Control: In agenda based architectures (Figure 1a) tasks are posted on a agenda and scheduled for execution. A task is executed only after the previous task was completed. The waiting time of a task on the agenda is equal to the product of number-of-tasks-ahead in the task queus and the average-task-execution-time. The agenda gets a chance to deallocate a task every average-task-execution-time seconds. Thus responsiveness can be increased by decreasing the task granularity. Large tasks can be decomposed and expressed as a sequence of such micro-tasks. For example, hospitals have time-based agendas for nurses. By appropriately decomposing the size of the tasks they can do (such as administor medicine, take temperature) their responsiveness is increased.
- Priority Channel Architectures. In priority channes architectures different channles are provided for events of different priority (Dodhiawala et al., 1989). A channel defines computation through all the stages of problem solving (very much like a thread) (Figure 1b). Tasks on a higher priority channel are given prcedence over tasks on lower priority channel. Thus the responsiveness for high priority tasks is much better. For example, an

accident victim will probably be put on an emergency channel.

- Multiple Priority Task Queues: Instead of having a single task queue one can define multiple task queues based on the priority of tasks. In such a scheme a cpu time slice will always be first given to the highest priority task queue. In multiprocessor architectures instead of a cpu time slice we will allocate adequate number of processors to appropriate task queue. The allocation of processors can be either static or dynamic. In dynamic allocation the responsivenss requirement can be changed to meet the external demand and processors be allocated appropriately. The QP-Net architecture provides the dynamic allocation feature [Sharma and Sridharan, 1988]. As an example certain wards in a hospital, such as intenisive care, get a high priority.

2. Redundant Tasks. AI tasks often have unpredictable execution times. In some designs multiple solution tracks are simultaneously investigated and the best solutions obtained within the deadline is accepted. For example, in some emergencies both the hospital and the local fire department can get a call. This causes the problem of deleting no more desired solution tracks. One way is to keep track of all of the redundant tasks and then delete them later. Another possibility is to make these tasks timed tasks with an allocated time budget. If the tasks do not complete within the time budget then they self terminate.

3. Lack of Resources. In some situations it is possible that resources necessary for a task may no longer be available (Dynamic reallocation of resources will cause this situation). Thus it will be desirable to remove the task from the list of active tasks. This problem is of importance in control systems tasks where certain actions may continue ignorant of the status of resources. A solution is to check for the resources periodically and self suspend/terminate when such resources will no longer be available.

4. Remove Undesired/Harmful Tasks. In some contexts certain tasks if allowed to continue will cause unacceptable consequences. In such cases these tasks have to be identified and interrupted. Task interruption will also be needed in case of conflicts between the tasks.

5. Free-Up Resources. If the resources needed for a task cannot be made available by rescheduling and the tasks cannot wait then certain number of tasks will have to be interrupted.

In summary:

1. For most of the traditional reasons for interruption it is possible to avoid task interruption by utilizing appropriate architectural and task model designs. Given the cost of interruption such alternatives are preferable.
2. Situations where task interruption is needed are: task conflicts (logical), removal of potentially harmful tasks, and for freeing-up resources for more critical tasks.

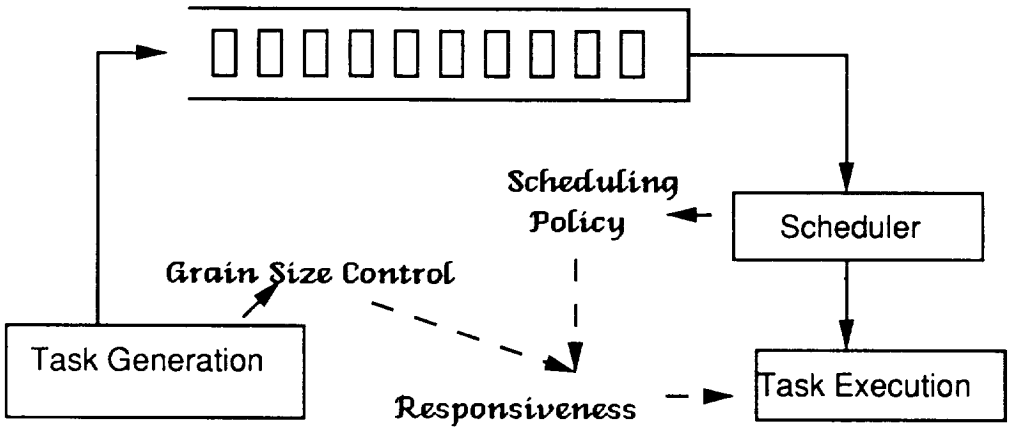


Figure 1 a. Agenda Control Architecure ~~(RT-1)~~

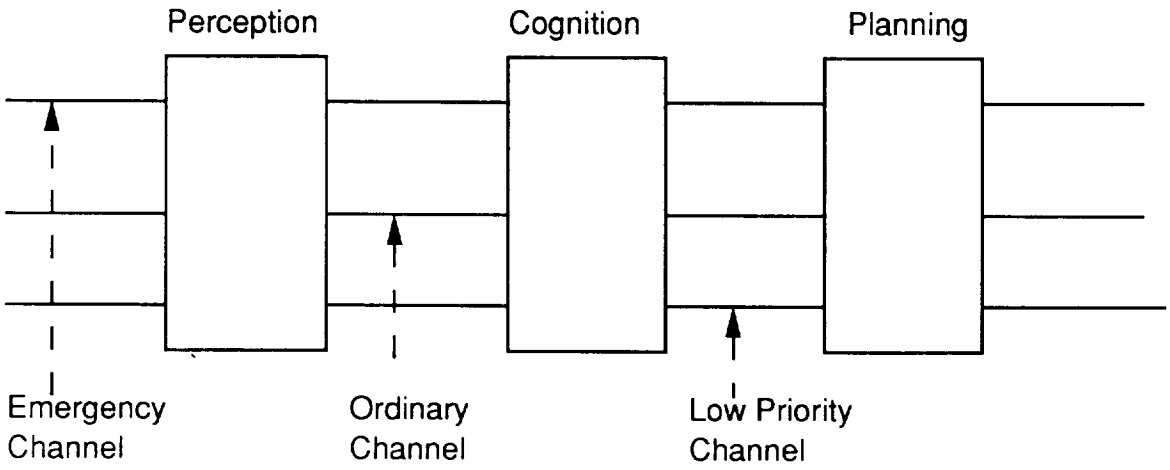


Figure 1 b. Priority Channel Based Architecture

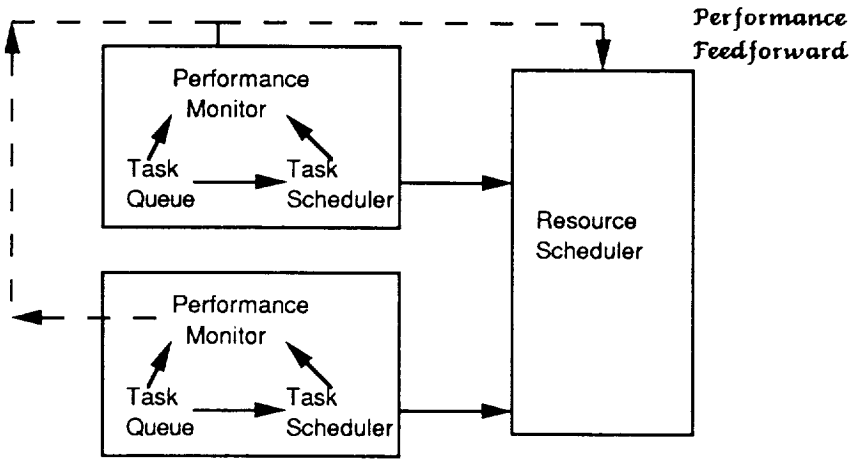


Figure 1c. Performance Controlling Architecure (QP-Net)

5 Who to Interrupt?

The question of who to interrupt needs to be answered in basically two cases: (1) There is a logical conflict between tasks, and (2) There is a need to free-up resources for more critical tasks. To solve this problem we will have to develop an approach for selecting one set of tasks over other. Such a selection needs to based on the significance of the tasks to the overall goals of the system. We thus need to develop certain approaches for expressing and reasoning about the significance.

Task Significance Principle: The information on task significance can be used as follows: If there are n active tasks (T1, T2, ..., Tn) and a new task T is awaiting execution then interrupt enough number of tasks of significance less than T so as to release sufficient resources for T.

5.1 Representation Issues of Task Significance

The significance measure of a task should express the following types of knowledge about the tasks:

1. Certain domain specific preferences or priorities. For example, in most domains the safety issue will have a precedence over productivity issues.
2. Sensitivity to real-time factors such as task deadlines. Thus even though two instances of same task (defend against an enemy missile) may have same preference but the one with a shorter deadline takes precedence over the other.
3. Sensitivity to Context. Most real-time systems operate in different modes. For example a certain fighter plane can be either in attack mode or egress mode. Depending upon the mode the tasks may have different preferences.
4. Task Interactions. In complex systems the effect of execution or termination of a task can propagate various side effects. In real-time systems the unpredictable dynamics of the environment makes it infeasible to anticipate and compile all possible interactions. The exact nature and the extent of the impact of these side effects has to be determined during run-time.
5. Task Dependencies. The basic principle here is that if a significant task depends upon the result of a specific task then that task is significant too. For example, if we interrupt the supporting tasks then the context and resources for several other tasks may be lost and they may have to be interrupted as well. In this dependency analysis we need to examine the criticality with respect to the new_task T as well. Moreover, the task dependencies can be time dependent. For example, given three tasks Ti, Tj, and Tk task Tj may require that task Ti be executed by some time t1, and task Tk may require that Ti be executed within the window (t2, t3).

In summary:

1. Significance of a task depends upon the runtime environment of a system.

Determining task significance requires extensive runtime analysis. The problem being that the computational cost of such analysis may be prohibitive to real-time performance".

2. Even if the computation of significance can be done in real-time the determination of the impact of task interaction and dependencies will be imprecise. This is due to the unpredictable and dynamic nature of the real-time environments.

5.2 Reasoning Issues Related to Task Significance

Given significance measures of various tasks one can then use the Task Significance Principle as a reasoning guideline. In dynamic systems where the significance of tasks can change rapidly the recomputation of the significance of all the tasks can be prohibitive. To appreciate the implication of the last point let us consider two measures of significance: a cost/benefit measure and a utility measure.

Cost/Benefit Measure. Assume a new task T on the waiting queue and N tasks on the active queue. Should we interrupt an active task in favor of new task T? In a cost/Benefit decision making framework it will be argued that this question should be answered based on the following considerations:

(1) What is the benefit of executing T?
(2) What is the penalty of not executing T?
(3) If T will be swapped with an active task Ti then
what is the penalty of terminating Ti.

T should replace a Ti if there exists a Ti such that:

(1) - (3) > (2)

This requires computing at least N such relationships. The bigger problem arises in computing each of the terms (1), (2), and (3). Since the benefit and penalty depends upon the interaction between the task and the resources needed for T may be obtained by interrupting more than one active tasks the complexity of the problem becomes quite hairy. Apart from the complexity arising from the combinatorics of the problem there is also complexity due to interactions based on phenomenological knowledge.

Utility Measure

One way to escape the problem of performing runtime analysis is to define some measure of utility of allocating resources to tasks. Thus if the system has say a maximum of N tasks then we can define utility of allocating computational resources to these tasks. At any given moment let us assume there will be M (< N) tasks on the scheduling queue. Let us assume that we can only allocate resource quantity R. Then we will select the subset of tasks from M such that the total resources is less than or equal to R and the utility is maximized. Every time we reschedule resources we will recompute the utility. Apart from the problem of how we define these utility functions we

have the following problems:

1. We will have to evaluate 2**N configurations to find out the maximum utility subset.
2. The utilities typically will not be static but context dependent. Thus we may have to compute utility functions themselves. In a tightly coupled system these utilities may depend on several parameters thus creating a modelling nightmare.

In practical systems another problem arises. Not all tasks are equal and representing them on a uniform scale can lead to anamolus results. For example, suppose we have the following preference rule for a passenger plane:

Ground Landing is preferred to Crash Landing is preferred to Total Crash

Using utility theory we can get the following undersired result:

```
Ground landing with probability p1
or Total Crash with probability p2
is preferred to
Crash Landing with probability p3]
```

In summary:

1. We need a measure of significance which is easy to model, compute, and change dynamically. The significance measure will have to be sensitive to contexts, task interactions, and dependencies.
2. To efficiently search for preferred tasks we need a computational architecture.

6 TIPS: A Preference Based Architecture for Task Interruption

TIPS is an agenda-based architecture. It consists of a waiting queue (see Figure 2), an active tasks queue, a tasks scheduler to schedule tasks on the waiting queue, an interrupt scheduler, and a resource scheduler. In general, there can be more than one waiting and active tasks queue for each task category. For example, one can have an emergency waiting and active task queue similar to channels design in RT-1 [Dodhiawala et al., 1989]. In such a case the resource scheduler will first always check the emergency queues for work to do. In general, resource scheduler allocates cpu time slices (in multiprocess systems) or processors (in multiprocessor systems). The task scheduler is same as the scheduler in agenda based systems. It prioritizes tasks on the waiting queue. The interrupt scheduler basically solves the problem of determining whether an interrupt is needed and who to interrupt.

Interrupt scheduler will not solve the problem of responsiveness and other aspects of real-time systems for which interruption is really not needed. It is assumed that for responsivenss appropriate functional architectures will be used such as RT-1. Also the computation of deadlines and resource requirements are solved by the problem solving

architecture.

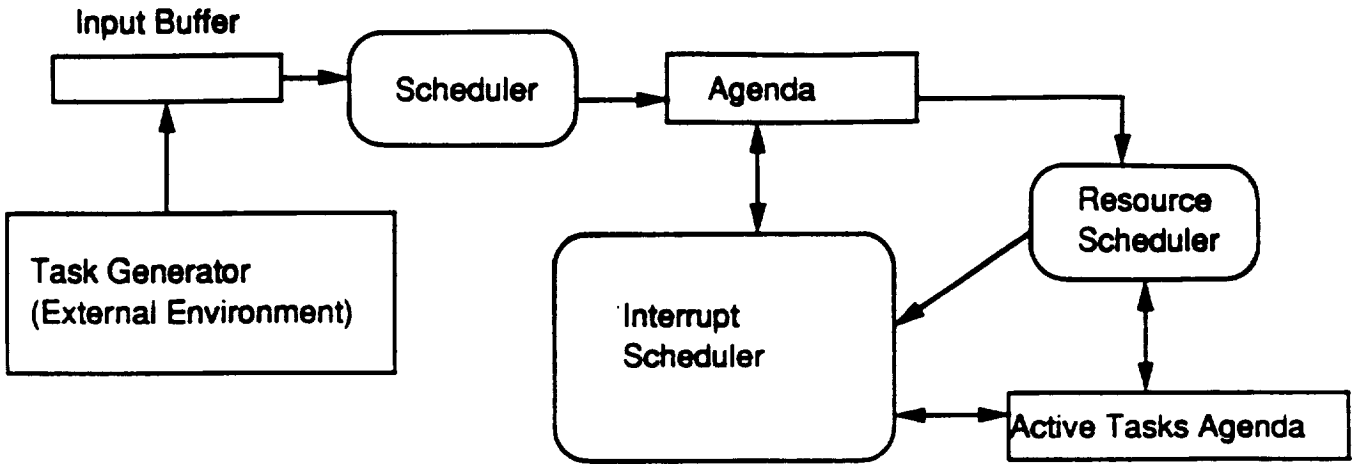


Figure 2. A Preliminary Architecture for TIPS

6.1 Design of Interrupt Scheduler

Interrupt scheduler uses a model of domain contexts, goals, and tasks defined at compile time. The model is hierarchical and consists of at least three layer as shown in Figure 3. Figure 4 shows an example from the Howitzer domain. There are two aspects to the hierarchical model:

1. Preferences of priorities of the tasks are defined with respect to the goals. In general priorities of elements in layer k are defined with respect to the elements in layer (k-1).
2. The hierarchical model is also used as a control structure for determining who to interrupt.

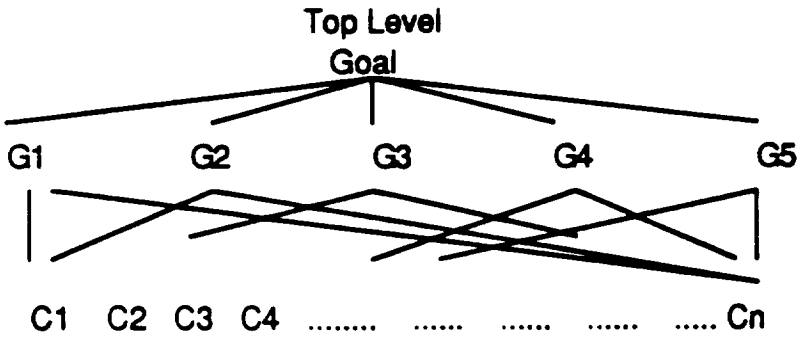


Figure 3. A Preference/Priority Hierarchy

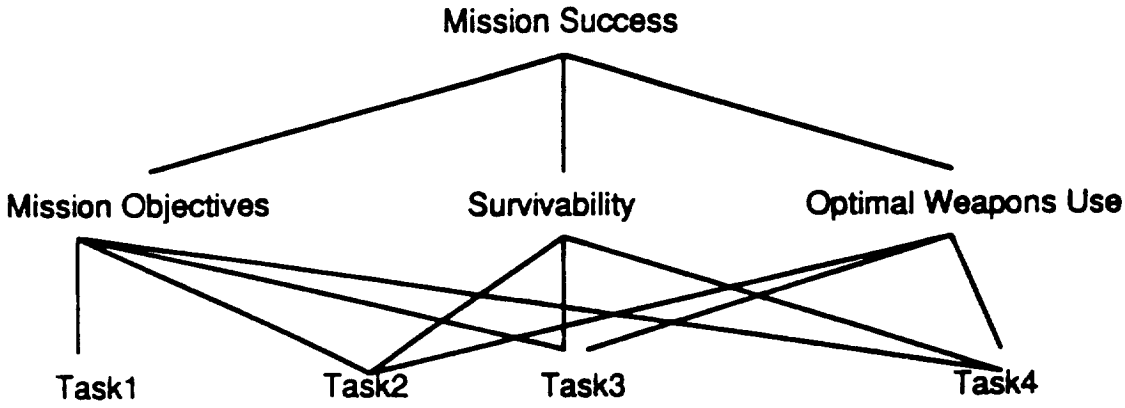


Figure 4. Goals-Task Priority Hierarchy Example

Modeling Task Priorities

The following discussion is based on Saaty's treatment of priorities in hierarchical systems [Saaty 1980]. We will not review Saaty's theory here but simply illustrate it by examples. Some of the basics are as follows:

1. Let C1, C2, C3, ..., Cn be n tasks or goals. Then priority of Ci is defined as a function which maps C -> [0, 1]. If wi is priority of Ci with respect to a high level goal then

$$w1 + w2 + w3 + ...+ wn = 1$$

2. Let xPy denote that x is preferred over y. xIy implies x and y are equally pereferable. Then following relations hold:

```
if xPy then wx > wy
if xIy then wx = wy
if xPy, yPz then xPz and also that wx > wz
```

In case of hierarchical structure shown in Figure 3, the priority of the first task with respect to the top node is given by:

$$WC1 = WC1(G1)*WG1 + WC1(G2)*WG2 + ... +WC1(G5)*WG5$$

Where,

WC1 = Priority of C1 with respect to Top Level Goal
WC1(Gi) = Priority of C1 with respect to goal Gi
WGi = Priority of Goal Gi with respect to Top Level Goal

Task Priorities under Change of Context

If the context changes then the preferences of goals changes with respect to the overall objective. Given new goal level priorities new task level priorities can be

computed.

Example

Consider the aircraft landing example shown in Figure 5. Under the current context the priorities are shown in Figure 5a. In this context it is more important to find an airport with adequate landing facilities. Expedient landing will help passengers feel better but that is less important. In this case the priorities computation lead to the choice of airport D. The task for performing landing at airport A is initiated.

After a while the pilot gets a low fuel level alarm and changes his priorities in favor of expedient landing. He only needs to change the priorities at the goal level. This new prioritization leads to selecting airport D.

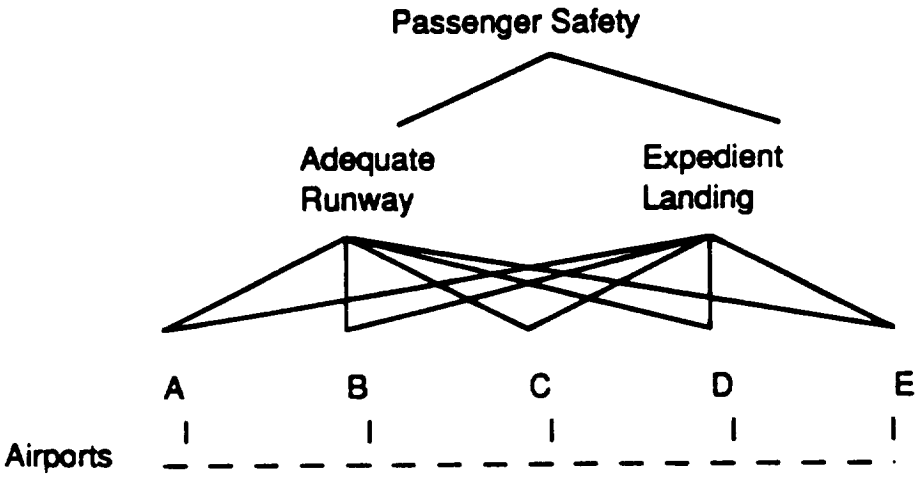
6.2 Task Models

We are essentially considering two types of tasks: tasks that are interruptible and the tasks that are not interruptible. Tasks can be composed of other tasks or micro tasks. Micro tasks once started execute to completion. The size of the micro tasks is such that their execution time is less than the overhead time associated with task interruption.

```
(Task new-task
 :preconditions
 :body
 :sponsor
 :goal
 :context
 :priority
 :children-tasks
 :Dependents
 :Supporters
 :Exclude-List
 :deadline
 :execution-time
 :resources-needed
 :status)

)

(Micro-task new-micro-task
 :precondition
 :body
 :sponsor
 :goal
 :context
 :priority
 :deadline
 :execution-time
 :resources-needed
```



Goals Priority Matrix

Goals	Contexts: Default	Contexts: Low Fuel Alarm
Adequate Runway	.9	.2
Expedient Landing	.1	.8

Airports Priority Matrix

Airports	Goals: Adequate Runway	Goals: Expedient Landing
A	.4	
B	.3	.2
C	.2	.2
D	.1	.4
E		.2

Figure 5. Airport Prioritization for Passenger Safety

```
:status)
```

6.3 Control Scheme

The control scheme works on the following principles:

1. Resolve Conflicts Locally First. If a task appears on the waiting list and was created to support goal Gi then compare the priority of the new task with respect to tasks currently active for supporting Gi. If necessary, swap the new tasks with one or more of the active tasks for Gi.

2. Fallback to the next higher level in the hierarchy. If the conflict cannot be resolved locally then it is necessary to resolve conflicts at the higher goal level.

3. Observe Task dependency and interactions constraints to prune the alternatives. The task dependency and interaction constraints are defined as a part of the task models (see slots dependents, supporters, exclude-list in task model).

4. Use the most comprehensive priority value. It is possible for the same tasks to exist on the active tasks list/agenda/schedule/queue multiple times based on the different goals it needs to serve. In that case the total priority of the task should be used.

This is how the scheme will work. In the beginning the system starts with a goal G. In fact the system may start with several goals G1, G2, G3, ... For each of the goals priority is defined - the interpretation of preferences being the relative resources to be allocated to pursuing these goals. New tasks are either spawned by the goals or are created in support of the goals. The priorities of these tasks are defined at the time of task creation. These priorities can either be default priorities or may be based on the context at the creation time. We then use four principles defined above to solve the interruption problem.

7 Conclusion

In this paper we have described an analysis of task interruption and a paper design. We believe in the richness of task interruption process. It is our premise that task interruption should be avoided for time critical situation by using appropriate architectural and data structural features. For situations where interruption is essential we have argued in favor of minimizing runtime reasoning. We have described knowledged-based architecture (TIPS) which attempts to explore various features to reduces runtime reasoning. These features are: preference models for tasks, a domain specific task context tree, and a hierarchical control mechanism. Table 1 summarizes major points of this paper.

Acknowledgements

We wish to thank Dr. T. Hester and Dr. N.S. Sridharan for providing critical review of this paper and for many stimulating discussions.

References

1. [Dodhiawala et al., 1989] Dodhiawala, R, Sridharan, N.S.,Pickering, C., Raulefs, P., "Real-Time AI Systems: A Definition and an Architecture." Proceedings of IJCAI, 1989.

2. [Hayes-Roth, 1987] Hayes-Roth, B., "A Multi-Processor Interrupt-Driven Architecture for Adaptive Intelligent Systems". Report # KSL 87-31. Department of Computer Science, Stanford University.

3. [Hester, 1989] Hester, T. Personal Communication.

4. [Saaty, 1980] Saaty, T.L.,"The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation". McGraw-Hill International Book Company, 1980.

5. [Sharma and Sridharan, 1988] Sharma, D.D. and Sridharan, N.S., "Knowledge-based Real-Time Control: A Parallel Processing Perspective". Proceedings of AAAI, 1988.

6. [Sharma et al., 1989] Sharma, D.D.; Huang, S.R.; Bhatt, R.; Sridharan, N.S.; "A Tool for Modeling Concurrent Real-Time Computation". Proceedings Workshop on Real-Time AI Problems, IJCAI, 1989.

Table 1. Designs Issues and Solution Options for Interrupt Based Architectures

Design Issue	Design Options
Why Interrupt?	
Responsiveness	No Interrupt, Control of Task Grainsize, Special architectures (RT-1, QP-Net)
Free-up Resources	Reschedule tasks, Interrupt in favor of Critical tasks
Tasks Not Needed	Soft/Hard Interrupt
Who to Interrupt?	Preference/Priority Based Selection, Context Tree, Hierarchical Task Selection (TIPS)
Where to Interrupt?	Special Task Models Predefined logically safe states (between two micro-tasks) On demand (special cases only)
How to Interrupt?	
No Interrupt	Task Size, Micro-tasks, Dynamic Allocation
Soft Interrupt	Remove from Agenda or Active Tasks Queue
Hard Interrupt	Utilize RTOS support