A Conditional Scheduling Approach to Designing Real-Time Systems

Lloyd Greenwald
Department of Computer and Information Science
University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104-6389
lgg@linc.cis.upenn.edu

Thomas Dean
Department of Computer Science
Brown University
Box 1910, Providence, RI 02912
tld@cs.brown.edu

Abstract

We present an approach to designing real-time systems based on dynamically sequencing condition-specific task-execution schedules. A system that dynamically alters its real-time execution component provides flexibility in the face of changing on-line conditions. For domains in which real-time response and safety must be guaranteed at design time, achieving this flexibility requires the introduction of new modeling and analysis tools. We provide analytical techniques for validating the behavior of a real-time system designed under our conditional scheduling approach. We demonstrate the approach through the detailed design and analysis of a real-time avionics scheduling solution. This solution involves architectural changes to existing avionics system hardware and makes use of predictive models of in-flight dynamics to provide real-time behavior guarantees at design time. The approach described in this paper is an example of the use of a general framework that we have developed for analyzing tradeoffs when designing systems in which an agent with limited computational resources is required to respond in a timely manner to situations arising in a dynamic environment.

Introduction

Flexibility in the face of changing conditions is a common goal of intelligent planning and scheduling systems. To attain this goal many systems have been designed that modify plans incrementally on-line (Georgeff & Lansky 1987; Lyons & Hendriks 1994). A typical architectural design of these systems is to combine fast reactive components with slower deliberative components (Gat 1991; Simmons 1991).

Applying intelligent planning and scheduling solutions in real-time environments requires carefully modeling and analyzing the impact of on-line deliberation on behavior guarantees. Due to safety considerations real-time control must be strictly guaranteed during any on-line deliberation. In the planning and scheduling literature (Musliner, Durfee, & Shin 1995) present a planner that constructs provably safe task-execution schedules, and (Pell *et al.* 1997) discuss safe execution during planning.

Conditional scheduling makes use of a predictive model of the dynamic environment to replace on-line deliberation with the conditional sequencing of provably safe task-execution schedules constructed at design time. Contrasted with on-line planning, this approach permits tight design-time time and space guarantees. Conditional scheduling is an example of the use of our general framework for analyzing tradeoffs when designing systems in which an agent with limited computational resources is required to respond in a timely manner to situations arising in a dynamic environment (Greenwald & Dean 1997).

We develop the conditional scheduling approach in the context of real-time avionics scheduling. A real-time avionics schedule allocates the resources of a distributed multiprocessor system to the execution of aircraft flight operations. Providing provably safe schedules is complicated by the combination of uncertain in-flight conditions and system resources that are tightly limited due to power and weight considerations.

The traditional approach to real-time avionics scheduling is to design a single static schedule that samples flight operations at fixed frequencies throughout the flight. For solutions based on static schedules, behavior and timing guarantees are dictated by worst-case in-flight conditions.

A conditional scheduling solution replaces the static schedule with a collection of situation-specific schedules. Each schedule is designed to provide guaranteed behavior (*e.g.* safety) over a subset of in-flight conditions. A higher level of control is provided to dynamically switch schedules as conditions change.

The intention of conditional scheduling is to extend the capabilities and flexibility of existing systems without sacrificing real-time behavior guarantees. Conditional schedules allow more effective use of limited resources by taking advantage of situation-specific conditional relationships between flight operations.

For example, the thrust-management operation may require frequent execution during take-off and landing, while stabilization may require more frequent execution during cruising. Additionally, climate control may need more monitoring during warm clear weather conditions, while the navigation control may require more frequent

monitoring during foggy conditions. The flight conditions that dictate worst-case sampling frequencies may differ for different operations. Simultaneously sampling all flight operations at worst-case frequency may never be necessary in practice. Conditional scheduling mitigates worst-case assumptions and provides effective use of limited resources by targeting resources toward the most currently critical flight operations.

A conditional scheduling solution must include a strategy for switching schedules that guarantees that, at any given time, the active schedule controlling the avionics hardware implements sampling frequencies that are appropriate for any in-flight condition that may occur before a new schedule is activated. All computational resources required to execute this strategy must be explicitly modeled in order to provide real-time behavior guarantees. To design this strategy we require a model of in-flight dynamics that can be used to predict changing in-flight conditions.

Conditional scheduling is more flexible than traditional static scheduling and provides guarantees difficult to obtain for solutions that include on-line deliberation. One form of on-line deliberation is a fully dynamic scheduling solution that reasons about arbitrary dynamic changes to the flight operation sampling frequencies. It is difficult to provide design-time methods that guarantee real-time behavior under such general solutions. Conditional scheduling provides a direct path for extending traditional deterministic methods.

Dynamically altering schedules has been discussed in work on *mode changes* (Sha *et al.* 1989; Fohler 1992) and *parametric dispatching* (Gerber, Pugh, & Saksena 1995). A mode change consists of swapping static schedules at execution-time in response to a specific event that triggers the change. Modes and mode change events are specified as part of the design problem. The focus is on safe operation of the system during the transition from one mode to the next. This work is complementary to conditional scheduling in that our focus is on providing design-time methods to generate swapping strategies that guarantee real-time behavior.

In parametric dispatching, tasks are statically ordered at design time but, the actual start time and execution time of each task are determined dynamically. Parametric dispatching consists of a design-time component to determine the static ordering, an on-line scheduler to dynamically determine upper and lower bounds on start times for each task, and a fast dispatcher to implement the schedule, taking into account any non-real-time tasks. Parametric dispatching provides guarantees with respect to the static, deterministically known components of the schedule. Given nondeterministic execution times, a parametric dispatching solution can not guarantee at design time any improvement over the worst-case static solution, though the expectation is that it will adapt to varying execution times. Given a predictive dynamic model, the analytical techniques we have developed for conditional scheduling may prove suitable for analyzing the on-line components of parametric dispatching solutions as well.

Another approach that replaces on-line deliberation with design-time reasoning is *just-in-case* scheduling (Drummond, Bresina, & Swanson 1994). In this work a stochastic model of action duration is used to generate contingent schedules that address likely breaks in a nominal static schedule. Just-in-case scheduling optimizes reactive behavior for a given error model. Conditional scheduling anticipates errors by modeling the dynamic conditions that predict errors. The analytical techniques of conditional scheduling may be used to determine the level of achievable guaranteed behavior. Furthermore, conditional scheduling techniques may be used to reason about the bounded time and space available for executing a just-in-case scheduling solution.

After presenting an existing real-time avionics system, we develop the architectural changes to the avionics hardware required to implement our conditional scheduling solution. We then discuss how predictive dynamic models are employed in our solution and develop analytical methods for guaranteeing real-time behavior. Finally, we briefly discuss the results of a limited empirical study on the application of these techniques.

Real-Time Avionics

In *fly-by-wire* aircraft all flight operations are computer controlled. Since there are no mechanical connections between pilot and airplane, it is critical that the electronics designed to control these aircraft adhere to strict FAA validation. As an example, we describe a real-time avionics system similar to the Airplane Information Management System (AIMS) designed by Honeywell for the Boeing 777 (Carpenter *et al.* 1994; Hoyme & Driscoll 1993). Subsequently, we extend this system to accommodate conditional scheduling.

AIMS consists of two redundant rack-mounted hardware cabinets. Each contains a 2-bit parallel *SAFEbus* and a number of Line Replaceable Modules (LRMs). Processor LRMs process flight operations, while I/O LRMs process flight data. Multiple LRMs compose a distributed multiprocessor hardware system. Each processor LRM contains bus interface circuitry, processor, intermodule memory and a schedule table implemented in eprom. Figure 1 illustrates a distributed multiprocessor avionics hardware system similar to AIMS.

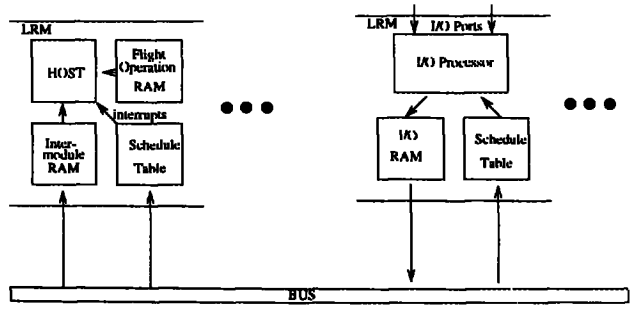

Figure 1: Distributed multiprocessor avionics hardware

Schedule tables provide synchronization for processing and access to the shared bus. A schedule table has a fixed number of entries that compose a cyclic schedule. Each entry consists of either a data transmit, data receive, or processor interrupt command. Data transfer commands hard-code storage locations in intermodule memory for transmission or reception. Interrupt commands are used to signal process swaps to the real-time OS in the host processor of the given LRM. The flow of data across the SAFEbus and the processing activity of each parallel processing LRM are deterministically synchronized.

Real-time avionics scheduling, in this context, refers to the generation of schedule tables for all LRMs, such that the resulting sequences of transmit, receive and interrupt commands satisfy I/O and flight operation processing requirements. Each LRM has limited processing and memory capacity. Flight operations might include thrust management, flight-data acquisition, flight and navigation display systems, flight management, climate control, data communications management, and central maintenance. Schedule tables are generated at design time and must be guaranteed to provide safe, time-critical behavior, consistent with FAA regulations, under uncertain in-flight conditions.

Real-time avionics scheduling is complicated by subtle dependencies between flight operations, and competition for limited data bus capacity and limited LRM processing and memory capacity. The design task is to allocate the limited hardware resources to satisfying the processing requirements.

An important requirement is that each flight operation must be scheduled according to its predetermined *sampling frequency*. The sampling frequency specifies the number of times per second a flight operation must be sampled on the distributed hardware in order to ensure time-critical response. For example, in AIMS these frequencies range from 5-80 hertz. Other requirements include the execution time per sampling, data size, and dependencies between operations and across samplings.

Sampling frequency requirements are determined according to traditional real-time repetition robustness criteria. For each flight operation this minimum *worst-case* frequency requirement is dictated by the worst-case environmental conditions the aircraft might encounter. Schedules are statically constructed to adhere to these worst-case requirements.

These processing requirements determine a cyclic scheduling problem. Within each cycle, each operation must be scheduled its required frequency of instantiations on the distributed multiprocessor hardware; with dependencies and other requirements satisfied. In the AIMS system, the static cyclic schedule consists of thousands of jobs that must interact with the asynchronous external environment through shared I/O ports. Figure 2 depicts a worst-case static schedule implemented in schedule table eprom. Each flight operation is sampled at its worst-case frequency, regardless of in-flight conditions.

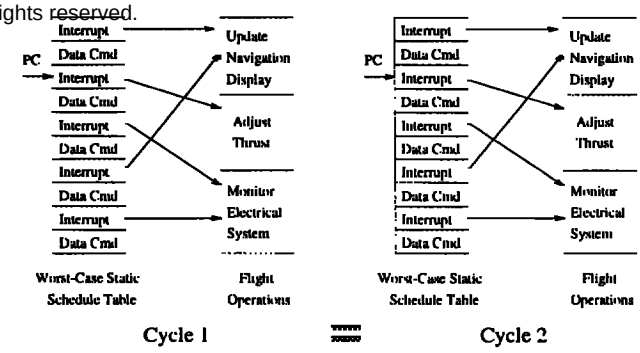

Figure 2: Worst-case static schedules are fixed at design time *i.e.* they do not change from one cycle to the next

A Conditional Real-Time Avionics Scheduling Architecture

A first step in developing a conditional scheduling solution is to adopt an architecture suitable for switching schedules in response to sensed flight conditions. In this section we develop an architecture that extends the distributed multiprocessor architecture described above.

The three main architectural changes required to accommodate conditional scheduling are (1) additional storage for conditional schedules, (2) sensors and circuitry to sense changing in-flight conditions, and (3) circuitry to implement the schedule switching strategy.

To store multiple schedules we introduce an additional LRM to serve as a *schedule server*. The schedule server LRM is packed with eproms that contain precompiled conditional schedules, each targeted toward a different set of in-flight conditions. The schedule server LRM also contains a closed-loop *paging circuit* that is designed to "eavesdrop" on all flight data that is transmitted by the I/O LRMs, and use this data to select the most appropriate conditional schedule. Figure 3 depicts the augmented hardware.

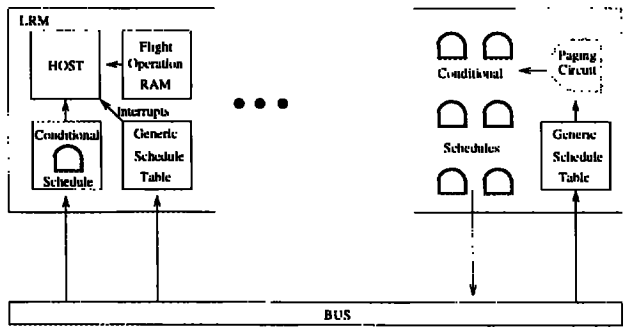

Figure 3: Distributed multiprocessor avionics hardware augmented with a schedule server LRM

All conditional schedules must be consistent with a fixed synchronization pattern. To implement this synchronization, we design a *generic schedule table* with pre-determined interrupt and transmit/receive com-

mands. This generic schedule table is used to synchronize flight operation executions and data transfers, as well as paging circuit execution and data transfers.

In this paper we assume a specific synchronization pattern for paging conditional schedules. At fixed intervals of time, the paging circuitry determines the current situation from the flight data and uses this situation to select a conditional schedule. A *situation* s, in this context, is an assignment of values to a set of variables corresponding to in-flight conditions such as altitude, inclination, cabin pressure, and fuel level. Let $\mathcal{S}$ be the space of situations, suitably defined over sensor readings, and $\mathcal{O}$ be the space of flight operations.

A selected conditional schedule is transmitted to the processor LRMs and loaded into the intermodule memory of each LRM. The schedule specifies a new pattern for flight operation execution over the next fixed time interval. Conceptually, the schedule server LRM is a black box that "delivers" an execution schedule that provides guaranteed real-time behavior for the set of situations that may arise during its execution.[1]

The flight operation that is launched during a given interrupt depends upon the conditional schedule currently loaded into intermodule memory. Data transfers are treated similarly. In other words, the host processor of the LRM interprets the commands of the fixed generic schedule table differently depending upon the conditional schedule currently in intermodule memory. In this way, synchronization may still be guaranteed deterministically at design time. Note that bus traffic due to paging must be taken into account in the generic schedule table.

To simplify the presentation, we take a restricted view of conditional schedules. In particular, we define a conditional schedule in terms of the flight operations represented and guaranteed sampling frequency for each flight operation in the schedule.

Static schedules must satisfy requirements determined by worst-case sampling frequencies. However, in a conditional schedule, a flight operation may be sampled at a lower rate than the worst-case frequency requirement. A conditional schedule must satisfy the worst-case frequency requirements only with respect to the set of in-flight conditions it is intended to address. The following conditional frequency model expresses situation-specific frequency requirements.

Definition 1 *Let* $\nu_o(s) \in \mathbf{Z}$ *be defined as the* instantaneous frequency requirement *of flight operation* $o \in \mathcal{O}$ *in situation* $s \in \mathcal{S}$. *For any set of possible situations* $S \subseteq \mathcal{S}$, *the frequency requirement of a given flight operation* $o \in \mathcal{O}$ *is defined as the maximum frequency over all situations in that set, namely* $\nu_o(S) = \max_{\{s \in S\}} \nu_o(s)$.

Note that worst-case sampling frequencies can be derived as a special case of Definition 1, in which we take the maximum frequency over all possible situations, $\nu_o = \nu_o(\mathcal{S}) = \max_{\{s \in \mathcal{S}\}} \nu_o(s)$. This is consistent with the traditional static approach to real-time avionics scheduling, that is invariant to in-flight conditions and must be defined over any possible situation in the space of situations.

The development of conditional frequencies for a given flight environment must be approached similarly to the traditional task of determining worst-case frequency requirements. Such procedures are problem-specific. An illustrative example of conditional sampling frequencies for real-time avionics systems is given in Table 1. This example consists of six different weather-related situations, and sampling frequencies for three different flight operations. Each flight operation is sampled either once or twice per cycle as a function of the in-flight situation.

Situation	Nav	Thrust	Elec
clear/calm (c/c)	1	1	1
clear/low winds (c/l)	1	1	1
clear/high winds (c/h)	1	2	1
cloudy (cy)	1	1	1
fog (f)	2	1	1
lightning storm (ls)	1	1	2

Table 1: Situation-specific sampling frequencies

Figure 4 depicts the change of execution from worst-case static schedules to conditional schedules. Under static schedules, the schedule table commands point directly to hard-coded flight operations, and remain constant over time. Under conditional schedules, the generic schedule table points to hard-coded addresses in intermodule memory that store the commands of the current conditional schedule. As shown, the commands in one cycle of processing may differ from the previous cycle as new conditional schedules are transmitted from the schedule server LRM. Each conditional schedule contains the same flight operations as the static schedule, but the frequency with which any given flight operation is sampled changes over time.

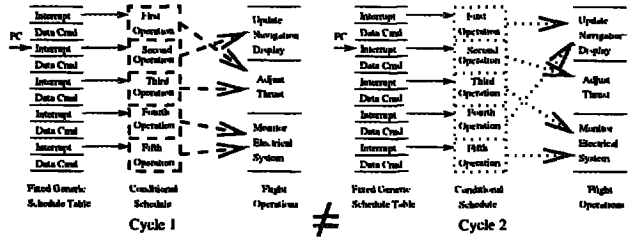

Figure 4: Conditional schedules alter scheduling of flight operations over time

[1] Note that an alternative design would be to include conditional schedules and paging circuitry on each processor LRM, instead of requiring a distinct schedule server LRM. This alternative design represents a different tradeoff between LRM space requirements, bus utilization, and schedule switching delay.

Next we develop conditional sampling frequency requirements using a predictive model of in-flight dynamics. These requirements may be compared to the sampling frequency guarantees of conditional schedules. We then demonstrate methods for guaranteeing real-time behavior that take into account all computational resources required to implement conditional scheduling.

Requirements of a Predictive Model of In-Flight Dynamics

When designing a conditional scheduling solution we must answer two questions concerning the in-flight dynamics: (1) what situations are possible at the points in time at which paging decisions are made? and (2) for each time interval between paging decisions, what situations are possible during the interval? We answer these questions with the aid of a *predictive model* of in-flight dynamics.

Developing a predictive model is complicated by the fact that in-flight conditions and control decisions are not independent. In other words, the frequency at which a flight operation is sampled in the current time interval may affect the in-flight conditions of future time intervals. A detailed discussion of these implications may be found in (Greenwald 1997).

Another way to look at the traditional approach to real-time avionics scheduling is to note that, as long as the specified worst-case frequency requirements are achieved by the static solution, safe behavior is guaranteed. The worst-case frequency model provides a way to achieve safe behavior without explicitly modeling the interaction of avionics system and external environment.

A conditional scheduling solution must be more careful in understanding the interaction between avionics system and external environment. The predictive model must account for this interaction.

Define an execution cycle to span n ticks. Once every n ticks, the current situation is determined from on-line flight data and is used as input to the paging circuitry. The paging circuitry must select a conditional schedule to transmit to the processor LRMs. It takes n ticks to select and dispatch a conditional schedule. This conditional schedule will then apply during the next cycle (that again spans n consecutive ticks). Thus, there is a time delay between making a paging decision and being able to execute the corresponding conditional schedule. During this time interval the previously paged schedule is executed. A paging decision must be based not only on the situation at the time a conditional schedule is selected but, also on the potential intervening situations and flight operation executions prior to the switchover.

The process of developing a predictive model is simplified in a hard real-time environment. In this environment we can assume that any active schedule has been guaranteed to meet the sampling frequency requirements for all situations that are possible during its activation. Using this knowledge, we can model the situations that are reachable in the next interval by focusing solely on the external sources of uncertainty and the known real-time behavior.

For example, consider changing weather conditions over time. We assume that the avionics system does not have any special ability to alter future weather conditions.[2] Figure 5 depicts a reachability model for weather conditions. The reachability graph is augmented with the minimum sampling frequency requirements for each flight operation for the given condition. This figure shows that, if the current situation includes the clear/low winds condition, the situations that are possible in the next cycle include clear/low winds, clear/calm, clear/high winds, and lightning storm. Additionally, depending upon the actual situation, the sampling frequency of either the adjust thrust operation or the monitor electrical system operation may need to be doubled to satisfy safe behavior requirements.

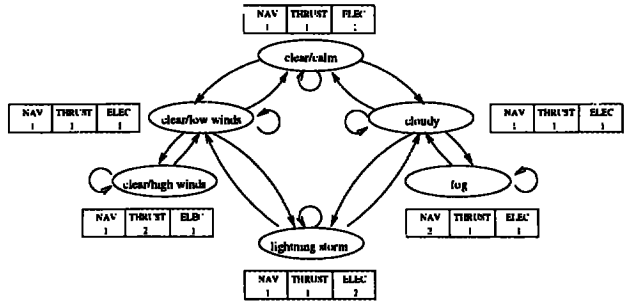

Figure 5: Example flight environment reachability model and situation-specific frequency model

In general, a reachability model can be represented by a set of situations that are possible at the start of execution, P^0, and a function capturing the transitions between situations, P_s^k. P_s^k describes the set of situations possible at tick $t+k$ given situation s at tick t, under safe behavior. The arcs in Figure 5 represent a single-tick reachability model P_s^1.

Define F^n to be the set of situations that may occur at time points at which paging decisions are made, and define $R = \{R_s^n | s \in F^n\}$ to be the collection of sets of situations that may occur during each execution cycle. Where $R_s^n = \cup_{\{n \leq k < 2n\}} P_s^k$. In other words, R_s^n includes all situations that may occur between n and $2n$ ticks after sensing situation s.

It is reasonable to assume that the flight environments of real-time avionics problems are *ergodic*. In other words, in the long run any situation can occur at any time. Thus, we typically define F^n to be the

[2] This assumption is not as trivial as it may appear. The avionics system may be designed with very early detection of weather hazards. Flight operations may then alter the course of the aircraft to avoid these hazards. In this example, we assume a short duration interval during which these maneuvers are no longer possible. In general, the predictive model must account for this interaction between controller and environment.

set of all possible situations $\mathcal{S}$. Therefore, the primary predictive model we require is the collection of sets $R = \{R_s^n | s \in \mathcal{S}\}$.

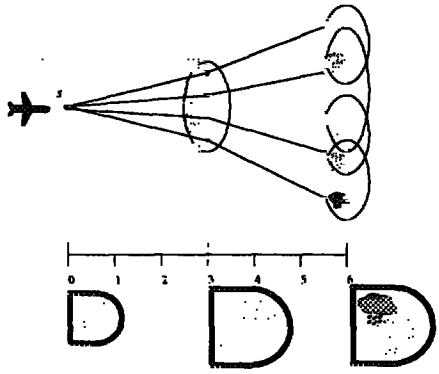

Figure 6: Controlling aircraft avionics hardware with weather-specific conditional schedules

In Figure 6, three different weather-specific conditional schedules are illustrated, corresponding to sunny conditions, combinations of sun and wind, and combinations of wind and rain. As the aircraft travels, a paging circuit monitors current weather conditions and reasons about future weather conditions. The paging circuit uses this information and a model of paging delay to determine which schedule should be controlling the avionics hardware at any given time.

Guaranteeing Real-Time Behavior with a Conditional Scheduling Solution

In order to guarantee real-time behavior under a conditional scheduling solution, two sets of guarantees must be satisfied. First, we must guarantee that the schedule server contains conditional schedules suitable for each set of situations that might occur during any in-flight execution interval. Second, we must guarantee enough space on the server to store these schedules, and that the time delay required to perform the switching of schedules does not exceed the fixed time interval n.

Definition 1 provides a direct way to combine the predictive model R with a situation-specific sampling frequency model in order to re-express R in terms of frequency requirements. This is captured in the following definition.

Definition 2 *For any set of situations r, let $\nu(r) = \{\nu_o(r) | o \in \mathcal{O}\}$ describe the set of frequency requirements for all flight operations over all situations in r. Given a collection of sets of situations $R = \{R_s^n | s \in \mathcal{S}\}$ and a frequency model ν, define the* coverage requirement *to be the collection $R_\nu = \{\nu(R_s^n) | s \in \mathcal{S}\}$. Denote the member of R_ν corresponding to situation $s \in \mathcal{S}$, by $R_\nu(s)$ i.e. $R_\nu(s) = \nu(R_s^n)$. The sampling frequency requirement of any given flight operation, $o \in \mathcal{O}$, is given by $\nu_o(R_s^n)$.*

Table 2 provides the coverage requirement for the example conditional frequency and reachability models of Figure 5. In the example problem, $n = 1$. Thus, $R_s^n = P_s^1$, for each $s \in \mathcal{S}$. The table depicts the single-transition neighbors of each situation, the instantaneous frequency requirement of each flight operation for each neighbor, and the coverage requirement due to the situation. The coverage requirement over all situations is $R_\nu = \{111, 122, 121, 212, 211, 112\}$. Note that the coverage requirement can be further summarized by the *flattened* collection $R'_\nu = \{122, 212\}$.

s	P_s^1	$\nu(s')$ for each $s' \in P_s^1$	$R_\nu(s)$
c/c	$\{c/c, c/l, cy\}$	$\{111, 111, 111\}$	111
c/l	$\{c/c, c/l, c/h, ls\}$	$\{111, 111, 121, 112\}$	122
c/h	$\{c/l, c/h\}$	$\{111, 121\}$	121
cy	$\{c/l, cy, f, ls\}$	$\{111, 111, 211, 112\}$	212
f	$\{cy, f\}$	$\{111, 211\}$	211
ls	$\{c/l, cy, ls\}$	$\{111, 111, 112\}$	112

Table 2: Deriving coverage requirements given a predictive model of in-flight dynamics based on single-tick cycles, and a corresponding frequency model

If we suppose that only five total flight operation samplings are allowed per cycle and that the lightning storm situation requires two navigation samplings per cycle (instead of one), the problem becomes infeasible. In particular, after sensing a clear/low winds situation, we would require a worst-case schedule (222) requiring six samplings in order to guarantee safety.

To evaluate the ability of a conditional scheduling solution to cover the requirements of a predictive model, R_ν, we must define the sampling frequencies provided by the collection of conditional schedules stored on the schedule server, C'. The following definition describes these sampling frequencies.

Definition 3 *The* domain *V_S of conditional schedule $c_S \in C'$ is the minimum sampling frequency $\bar{\nu}_o$ that it guarantees for each flight operation, $o \in \mathcal{O}$. $V_S = \{\bar{\nu}_o | o \in \mathcal{O}\}$, $\bar{\nu}_o \in \mathbf{Z}$.*

Since Definitions 2 and 3 express coverage requirements and conditional schedule domains, respectively, in terms of sampling frequencies, it is straightforward to compare these collections as shown in the following theorem.

Theorem 1 *Given a collection of conditional schedules C' and a coverage requirement R_ν, a conditional scheduling solution guarantees safe behavior only if for each $R_\nu(s) \in R_\nu$, there exists some conditional schedule $c_S \in C'$ such that, for each $o \in \mathcal{O}$, $\bar{\nu}_o \in V_S$ and $\nu_o(R_s^n) \leq \bar{\nu}_o$.*

The conditions of Theorem 1 can also be written as a system of *set covering* constraints. Note that Theorem 1 is defined with respect to a specific collection of conditional schedules C'. Namely, the collection that

has been implemented on the schedule server. This collection may be a subset of a larger space of possible schedules C. The theorem does not say anything about the existence of such a subset. Proving that a solution exists in general is NP-complete (Greenwald 1997). Since C' includes only those schedules satisfying the resource tradeoffs of the given architecture, the trivial solution of choosing the worst-case schedule (covering all requirements) is not necessarily feasible.

In order to guarantee safe behavior, we must show that Theorem 1 is satisfied *and* that the computational mechanisms to perform paging can be accommodated within the limited time and space resources. To evaluate the latter constraint we set up a system of *executability* equations relating computational resources to paging requirements. In general these equations do not have any specific structure. However, in this paper we develop a simplified conditional scheduling solution that allow us to combine the executability equations and set covering constraints into a single mixed integer linear program (MILP). We may then use standard techniques to determine whether or not real-time behavior may be guaranteed (Greenwald 1997).

Let M define the total space available on the schedule server for storing conditional schedules and let $B_\delta(c_S)$ define the amount of space required to store conditional schedule $c_S \in C$ on the schedule server. In order to employ a generic schedule table, it must be the case that paging can be synchronized using the same schedule of deterministically specified bus transmissions, regardless of the actual conditional schedule being executed. Thus, we assume that all conditional schedules on the schedule server require as much space as the largest schedule on the server.

Further let D^* describe the overriding maximum number of schedules that can be stored on the schedule server. Assume that there is enough space on the schedule server to implement a paging circuit mapping any possible situation $s \in S$ to at most D^* conditional schedules. For example, we might implement the paging circuit as a table with $|S|$ entries of $\log D^*$ bits each, requiring the current situation to be coded into $\log |S|$ address bits. Further assume that the paging interval n is long enough to allow paging of the largest available conditional schedule. Given that the generic schedule is linearly ordered, we can synchronize the paging of slices of the conditional schedule for the next cycle with the execution of the conditional schedule for the current cycle. This eliminates the need for any additional buffering resources.

These simplifications allow us to focus on the design-time tradeoff between the number of schedules stored on the schedule server and the size of the largest schedule. Note that this simplified conditional scheduling solution does not model multiple processors, I/O requirements, or ordering of flight operation samplings.

A collection of schedules $C' \subseteq C$ may be executed within the conditional scheduling architecture as long as the following executability equations are satisfied.

$$|C'| \leq D^* \quad (1)$$

$$|C'| * B^*_\delta(C') \leq M \quad (2)$$

Equation 1 states that the total number of schedules is no greater than the maximum that may be mapped by a paging circuit. Equation 2 states that the space requirements of the collection C' do not exceed the space on the server. Where space requirements are a product of the number of conditional schedules and $B^*_\delta(C')$, the maximum number of bits of any conditional schedule $c_S \in C'$.

For each possible conditional schedule, $c_S \in C$, we can use the executability equations to compute the maximum number of schedules, d_S, allowed in any solution containing c_S. Namely

$$d_S = \min\left(\frac{M}{B_\delta(c_S)}, D^*\right).$$

This construction leads to the following theorem.

Theorem 2 *Given a collection of conditional schedules $C' \subseteq C$, maximum server space M, and overriding maximum schedules D^*, C' is* executable *if and only if*

$$|C'| \leq \min_{\{c_S \in C'\}} d_S,$$

where $d_S = \min(\frac{M}{B_\delta(c_S)}, D^*)$, for each $c_S \in C$.

Theorems 1 and 2 capture covering and executability constraints that guarantee real-time behavior for a specific collection of conditional schedules $C' \subseteq C$. We may combine these constraints within a single MILP in order to search the space of possible solutions for a conditional scheduling solution that guarantees real-time behavior. The following MILP captures this combined set of constraints using standard formulation techniques.

Definition 4 *Let variable $Z_S = 1$, if conditional schedule $c_S \in C$ is in solution C', and 0, if not. Furthermore, let constant $W_{r,S} = 1$, if for each $o \in \mathcal{O}$ $\bar{\nu}_o \in V_S$ and $\nu_o(r) \leq \bar{\nu}_o$, and 0, if not, $\forall r \in R, c_S \in C$.*

$$\begin{aligned}
Z^I &= \min d \\
&\textit{such that} \\
\textstyle\sum_{\{c_S \in C\}} Z_S &\leq d \\
&\textit{for each } r \in R\textit{:} \\
\textstyle\sum_{\{c_S \in C\}} W_{r,S} Z_S &\geq 1 \\
&\textit{for each } c_S \in C \textit{ and } d_S < D^*\textit{:} \\
d - d_S - D^* + D^* Z_S &\leq 0 \\
0 \leq d &\leq D^* \\
&\textit{for each } c_S \in C\textit{:} \\
Z_S &\in \{0,1\}
\end{aligned}$$

MILP capturing constraints of Theorems 1 and 2

Given that d describes the number of schedules in a given solution and D^* is the largest possible number of schedules in any solution, Definition 4 specifies if-then constraints of the form $d - d_S - D^* + D^* Z_S \leq 0$ for each $c_S \in C$. If $Z_S = 0$ this constraint is always

satisfied. However, if $Z_S = 1$, this constraint may only be satisfied if $d \leq d_S$ *i.e.* if the number of schedules in the solution does not exceed d_S.

The following theorem directly follows from Theorems 1 and 2.

Theorem 3 *Given conditional schedules C, predictive model R, sampling frequency model ν, and the conditional scheduling architecture described above, a conditional scheduling solution with guaranteed real-time behavior exists if and only if the MILP of Definition 4 has a feasible solution.*

Discussion

This paper describes a new approach to designing and analyzing flexible real-time systems and demonstrates the approach by developing a conditional scheduling solution in the context of real-time avionics scheduling. This solution involves architectural changes to existing avionics system hardware and relies on predictive models of in-flight dynamics to provide real-time behavior guarantees at design time. Conditional schedules allow more effective use of limited resources by taking advantage of situation-specific conditional relationships between flight operations. Our conditional scheduling approach may potentially be applied to any domain in which worst-case load requirements can be weakened by conditioning on on-line situations.

The conditional scheduling approach additionally has the potential to provide new capabilities to real-time avionics systems. For example, by allowing sensor-based switching of schedules, we can achieve emergency-specific responses that differ from normal flight responses in cases for which these responses are dissimilar.

A fundamental difficulty with developing an empirical study of a new problem-solving approach is that data for evaluating the approach may not exist. With respect to applying the conditional scheduling approach to real-time avionics scheduling, there exist neither conditional frequency models, conditional schedules nor schedule servers.

While it is difficult to evaluate the practical benefits of conditional scheduling over traditional static scheduling without such real-world models, in (Greenwald 1997) we provide a limited empirical study of these claims. Our results demonstrate that our analytical techniques for evaluating conditional scheduling solutions may be practically applied at design time. Furthermore, we provide evidence that there exist a space of avionics problems for which conditional scheduling solutions can be effectively applied. However, these results are based on randomized studies. More work is required to demonstrate the benefits of this approach in real-world avionics scheduling or other hard real-time scheduling problems.

Acknowledgments

This work has been supported in part by the Air Force and the Advanced Research Projects Agency of the Department of Defense under grant No. F30602-95-1-0020, by the National Science Foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under grant No. IRI-9312395, and by the Veterans Health Administration through a post-doctoral fellowship in medical informatics.

References

Carpenter, T.; Driscoll, K.; Hoyme, K.; and Carciofini, J. 1994. Arinc 659 scheduling: Problem definition. In *Proceedings of the 15th IEEE Real-Time Systems Symposium.*

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proceedings AAAI-94.*

Fohler, G. 1992. Realizing changes of operational modes with a pre run-time scheduled hard real-time system. In *Proceedings of the Second International Workshop on Responsive Computer Systems.*

Gat, E. 1991. Integrating reaction and planning in a heterogeneous asynchronous architecture for mobile robot navigation. *SIGART Bulletin* 2(4).

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings AAAI-87.*

Gerber, R.; Pugh, W.; and Saksena, M. 1995. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers* 44(3).

Greenwald, L., and Dean, T. 1997. Tradeoffs in the design of on-line systems. In *Working Notes of AAAI-97 Workshop on On-Line Search.*

Greenwald, L. G. 1997. *Analysis and Design of On-line Decision-Making Solutions for Time-Critical Planning and Scheduling Under Uncertainty.* Ph.D. Dissertation, Brown University, Providence, RI.

Hoyme, K., and Driscoll, K. 1993. Safebus(tm). *IEEE Aerospace Electronics and Systems Magazine* 34–39.

Lyons, D. M., and Hendriks, A. J. 1994. Testing incremental adaptation. In *Second International Conference on AI Planning Systems.*

Musliner, D.; Durfee, E.; and Shin, K. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74:83–127.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *Proceedings IJCAI 15,* 1234–1239.

Sha, L.; Rajkumar, R.; Lehoczky, J.; and Ramamritham, K. 1989. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems* 1(3):243–265.

Simmons, R. 1991. Coordinating planning, perception, and action for mobile robots. *SIGART Bulletin* 2(4).