Embry-Riddle Aeronautical University

From the Selected Works of Sam B. Siewert

Spring March 10, 2001

Experiments with a real-time multi-pipeline architecture for shared control

Sam B Siewert

Available at: https://works.bepress.com/sam_siewert/10/

Experiments with a Real-time Multi-Pipeline Architecture for Shared Control

Sam Siewert

Department of Electrical and Computer Engineering
University of Colorado, Boulder, CO 80309-0520
siewerts@thinker.colorado.edu

Abstract—The concept of shared control distributes control between embedded systems, remote automation, and operators in order to create a system with optimized allocation of control functions to meet mission objectives. For example, a highly autonomous system will require much more embedded functionality compared to a teleoperated system that allocates more functionality to the operator. This paper reports research completed over the past six years at the University of Colorado with experimental systems including a Space Shuttle payload flown on STS-85 and an optical navigation test-bed. In both cases, the research has been focused upon defining a framework for real-time shared control. Initially, a top-down end-to-end approach was taken – this work culminated in the experimental Shuttle payload shared-control architecture known as DATA-CHASER. Experience with this system showed that further research was needed on the embedded side to create a better integration of hard and soft real-time control functions allocated between the embedded, ground and operator control segments. In addition, the challenges encountered in the embedded system segment showed that an implementation framework for the architecture was needed that would work equally well in any segment. This framework was implemented as an extension to the Wind River VxWorks kernel and evaluated on a test-bed called RACE. RACE optically navigates an air-powered vehicle semi-autonomously based on high-level operator commanding. This paper summarizes results from both the hard real-time RACE optical navigation experiment and the soft real-time DATA-CHASER Shuttle demonstration project and presents an integrated architecture for both hard and soft real-time shared control. The results show significant performance advantages of the shared-control architecture and greatly simplified implementation using the derived framework. Lessons learned from both experiments and the implementation of this evolving architecture are presented along with plans for future work to make the framework a standardized kernel module available for VxWorks, Solaris, and Linux.

TABLE OF CONTENTS

1. INTRODUCTION
2. SHARED CONTROL ARCHITECTURE
3. REAL-TIME EXECUTION FRAMEWORK
4. DATA-CHASER EXPERIMENTAL RESULTS
SUMMARY
5. OPERATOR-IN-THE-LOOP CONCEPT
6. RACE TEST-BED OPTICAL NAVIGATION
EXPERIMENT RESULTS
7. FUTURE WORK
8. CONCLUSION
9. REFERENCES

1.0 INTRODUCTION

In 1996 a joint effort between the University of Colorado Space Grant College and the NASA Jet Propulsion Laboratory was undertaken to implement a shared-control architecture for semi-autonomous operation of a Space Shuttle payload called DATA/CHASER [SiHan96]. The experimental system was flown on STS-85 in the summer of 1997 and while challenges with meeting all of the original objectives were encountered, the major objectives of incorporating flight and ground autonomous closed-loop control, on-line re-planning, and high-level operator control were all achieved [SiHan97]. Many of the problems encountered with realizing this system stemmed from the lack of a common embedded and ground system environment for real-time tasks. The shared control architectural concept requires building processing pipelines between sensors and actuators on the flight system, through the ground link, and through an operator control loop. Furthermore, the range of timing requirements and device interface characteristics make shared-control systems a challenging software engineering task. This experience became the motivation for development of a framework for quickly implementing processing pipelines with hard real-time, soft real-time, and best-effort timing requirements. The multi-agent shared control concept for space systems has been successfully used for a number of applications including deep space probes [NayKu99], telerobotics

[Bru93], and for mostly autonomous rovers [WeiRod99]. Furthermore, many new applications proposed for semi-autonomous space system architectures require more data intensive sensor processing [Dec99] and intelligent fault detection [WyShe99], more microprocessor resources for intelligent on-line planning [RabKni99], and yet still require traditional hard real-time digital control capabilities for typical functions like attitude control [Tör95]. Given this experience and motivated by emerging applications which require mixed hard and soft real-time processing and higher throughput, the RT EPA (Real-Time Execution Performance Agent) framework was proposed to simplify the task of implementing processing pipelines between source and sink devices with unambiguous specification of timing deadlines and reliability [SiNuHu97]. This framework has undergone extensive testing and evaluation in an optical navigation test-bed and has been shown to not only simplify implementation of complex real-time shared control multi-pipeline architectures [Si00], but also to solve difficult real-time scheduling problems for data intensive applications such as the SIRTf space-based telescope [SiNu00]. This paper reviews the concept for semi-autonomous shared control architectures requiring real-time processing pipelines, and then presents a summary of results obtained in the optical navigation test-bed. Finally, plans for future work to extend the RT EPA framework are presented.

2.0 SHARED-CONTROL ARCHITECTURE

The basic concept of shared-control architecture comes from robotics research and constitutes a compromise between totally autonomous systems and completely teleoperated systems. The operator-in-the-loop concept for semi-autonomous shared control has been applied to robotics, intelligent flight control systems, deep space probes, and planetary rovers. While it is clear this architecture has many uses, the timing for closing the loop in each of these example systems varies dramatically. Furthermore, a single system may include processing loops with tight timing requirements (deadlines on the order of microseconds or less) as well as loose timing requirements (deadlines measured in hours or days). This wide range of timing requirements was investigated in detail prior to the DATA/CHASER experiment [SiHan96]. The range of timing horizons for a shared-control semi-autonomous system is depicted in Figure 1. A full survey of application domains which include a mixed set of processing pipeline requirements was also completed [Si00], including application domains such as virtual reality [NuBra99]. Through this survey, it was found that frameworks similar to the RT EPA have been developed in the past as far back as 1991 [Gov91] and are currently under development as well [McCart00]. However, it is not clear that any of the related frameworks address the combined aspects of efficient processing pipelines and real-time performance.

Figure 1: Range of Timing Horizons for Semi-Autonomous Shared-Control Architecture

The RT EPA framework was designed to simplify the task of implementing shared-control architectures and to ensure real-time reliability in these systems. Such systems may include complex processing

pipelines and high bandwidth data sources, so in order to evaluate the framework well, an optical navigation pipeline experiment was devised. The experiment test-bed called RACE (Rail-guided, Air-powered Controls Experiment), shown in Figure 2, includes an air-powered vehicle which uses optical navigation to fly to a target location on overhead rails.

Figure 2: RACE Test-bed

The optical navigation shared-control architecture includes: autonomous navigation with on-board image processing, operator target commanding, operator telemetry monitoring, and on-board digital control. The experiment is fully documented in [Si00].

3.0 REAL-TIME EXECUTION FRAMEWORK

The RT EPA is a framework for developing reliable real-time digital control and data processing pipelines. The RT EPA has been described in detail previously [Si00] [SiNuHu07] [SiNu00]. The framework is based upon two new real-time processor resource management theories:

- 1) Confidence-based Deadline Monotonic Scheduling, and
- 2) Multi-Epoch Scheduling.

CBDM (Confidence-based Deadline Monotonic) scheduling extends the traditional DM (Deadline Monotonic) scheduling feasibility test [Au93] as follows:

eq 1: From probability theory for a normal distribution,

$$C_{\text{low or high}}(i) = C_{\text{expected}}(i) + Z_{p_{\text{low or high}}}(i) \left(\frac{\sigma(i)}{\sqrt{N_{\text{trials}}(i)}} \right),$$

where C is execution time for thread i and Z_p is the execution time distribution quantile.

eq 2: EPA-DM admission test: $\forall i: 1 \leq i \leq n$:

$$\left(\frac{C_{\text{low or high}}(i)}{D_{\text{soft or term}}(i)} \right) + \left(\frac{I_{\text{max}}(i)}{D_{\text{soft or term}}(i)} \right) \leq 1.0 \text{ ?},$$

where D is the service deadline relative to service release, provides the feasibility test for scheduling a service given processor resources.

$$\text{eq 3: } I_{\text{max}}(i) = \sum_{j=1}^{i-1} \text{ceiling} \left(\frac{D_{\text{term}}(i)}{T(j)} \right) D_{\text{term}}(j) \text{ ,}$$

where $I_{\text{max}}(i)$ is the interference time by higher priority threads $j=1$ to $i-1$ which preempt and run up to the ceiling term number of times during the period in which thread i runs.

The second fundamental theory for applying the RT EPA is ME (Multi-Epoch) scheduling. A

technique similar to ME scheduling has been used in a limited fashion on the Space Shuttle flight software [Carlow84]. The ME formulation used in the RT EPA has been employed to solve a challenging data processing performance problem on the SIRTf space-based telescope [SiNu00]. The idea was inspired by the Shuttle flight software design which includes major modes of real-time processing and operation that are mutually exclusive and independently evaluated as far as scheduling feasibility. The Shuttle flight software, however, only operates in each major mode once during a mission (e.g. ascent, on-orbit, entry). ME scheduling takes the concept used in the Shuttle flight software further by providing on-line automatic switching between memory-resident tasks in multiple software modes. The theory and initial application to SIRTf is described fully elsewhere [SiNu00] [Si00].

In addition to CBDM and ME scheduling, the RT EPA provides a practical API (Application Programmer's Interface) with on-line performance monitoring and control to enforce deadlines, provide execution models, and to handle execution faults.

In order to describe the API, an example application employing the RT EPA API is provided here. This example shows overall initialization of the system with a system safing routine for missed hard deadlines and then admission of a single pipeline service. The admission test guarantees reliable real-time execution with either a hard guarantee or a soft reliability guarantee. Execution faults are detected and handled such that failure of one pipeline does not adversely impact others – services are fire-walled from deadline over-runs by other unrelated services (e.g. failure in a re-planning application will not impact attitude control). Once all pipeline services have been admitted, the system is activated and is continuously monitored using previously reserved resources. The following code example is typical of a main application making use of the RT EPA framework.

First, an application program must initialize the framework itself by defining how hard deadline misses will be handled, how tasks not under control of the RT EPA will be handled, whether performance data will be computed on-line, and if it will, what the period for computation will be. The initialization call is:

```
rtepaInitailize
( (FUNCPTR) service_hard_deadline_fault_callback,
  (PERFORM_MON | DEMOTE_OTHER_TASKS |
   CREATE_IDLE_TASK),
  active_monitoring_period);
```

Since the RT EPA has been designed to handle event-driven processing, the system requires definition

of a data ready event semaphore for each data source in the system, which must in turn be associated with a device interface, typically through an interrupt service routine.

```
data_ready_event =
  semBCreate(SEM_Q_FIFO, SEM_EMPTY);

rtepaPCIx86IRQEventInitialize
(data_ready_event, irq, (FUNCPTR)service_isr);
```

After initialization and specification of the event source which releases the data processing or digital control service, the execution model for the service must be initialized and the new service tested for scheduling feasibility in the system where other services may already have been admitted and in operation.

```
service_execution_model_initialization();

if((rtepaTaskAdmit
  (&rtid[0], service_type_is_reliable,
   interference_assumption_is_worstcase,
   execution_model_is_normal_distribution,
   hard_deadline_miss_policy_is_restart,
   &service_execution_model[0],
   Dsoft[0], Dterm[0], T[0],
   &SoftConf, &TermConf, "tService1")
  ) != ERROR )
  printf("Service %d can be scheduled\n", rtid[0]);
else
{
  printf("Service admission error\n");
  return ERROR;
}
```

Once a new service has been admitted to the system, then the service must then be associated with a release semaphore and activated in order to be executed. At this time it is also possible to specify isochronal output from the service such that the RT EPA framework buffers results for output to the next processing stage or sink device interface on a strictly periodic basis.

```
event_release_type_info.release_sem =
  data_ready_event;

rtepaTaskActivate
(rtid[0], (FUNCPTR) service_entry_point,
```

```

(FUNCPTR) service_soft_deadline_miss_callback,
(FUNCPTR) service_release_complete_callback,
service_complete_type_is_isochronal,
service_output_period,
service_stack_size,
event_release_type_info,
online_model_size);

```

Performance monitoring for the system must also be specified if the application needs this data to handle execution faults or to re-negotiate service:

```

if((rtepaRegisterPerfMon
    (rtid[0],
    (FUNCPTR) service_renegotiation_callback,
    (ACT_EXEC | ACT_RESP | ACT_FREQ |
    ACT_HRD_CONF | ACT_SFT_CONF)
    ) == ERROR)
    printf("Service performance monitoring error\n");

service_source_activate();

```

As noted earlier, the RT EPA framework also makes specification of advanced real-time requirements such as isochronal output and pipeline configuration simple. The RT EPA API provides three major functions for pipeline control configuration.

First, a function to specify source device interrupt-based release as shown previously:

```

int rtepaPCIx86IRQReleaseEventInitialize
    (int rtid, SEM_ID event_semaphore,
    unsigned char x86irq, (FUNCPTR) isr_entry_pt);

```

Second, a function for specifying release of processing stages between the source and sink interfaces:

```

void rtepaPipelineSeq
    (int src_rtid, int sink_rtid,
    int sink_release_freq, int sink_release_offset,
    SEM_ID sink_release_sem);

```

Third and finally, a function for specifying whether a service should provide isochronal output:

```

void rtepaSetIsochronousOutput(int rtid, r_time Tout);

```

The remaining details of the API and much more detailed usage examples are given in previous publications [Si00].

4.0 DATA-CHASER EXPERIMENTAL RESULTS SUMMARY

The DATA-CHASER system was designed to provide three levels of automation within a distributed shared-control system. The system design included three basic elements:

1. On-board flight system remote reactive agent automation.
2. Ground system agent automation.
3. Ground teleoperations.

All three elements were incorporated into the system and demonstrated on STS-85. Section 5, which follows, provides a summary of how the system was implemented and which aspects were successful and which were not. Overall, the goals for shared control were met, but difficulties in implementation, especially in the embedded flight segment of control led to limited success in proving the architecture. These challenges were the motivation for later development of the RT EPA framework discussed in section 3.

5.0 OPERATOR IN-THE-LOOP CONCEPT

This high-level design is depicted in Figure 3. The embedded reactive agent design was based on having on-board telemetry monitoring provided by SELMON (Selective Monitoring) and a rule-based fault/protection and an autonomous reactive agent for real-time automated instrumentation operations according to SCL (Spacecraft Command Language) rules, scripts, and constraints executed on an FRTE (Flight Real-Time Executive). The ground system agent automation included both a reactive agent provided by SCL with a GRTE (Ground Real-Time Executive) and a longer-term deliberative agent capable of on-line re-planning to optimize mission objectives. The on-line planner called DCAPS (DATA-CHASER Automated Planning and Scheduling) was an adaptation of the JPL Plan-it II Lisp-based iterative repair planning system which was interfaced to SCL such that it generated scripts for activities and updated rule-set activation for operational modes. Finally, operators were provided a high-level graphical interface for monitoring system status and updating rules and plans on the flight remote reactive agent, the ground reactive agent, or on the

ground deliberative agent. Overall, the DATA-CHASER end-to-end system provided semi-autonomous operation where personnel only need intervene to handle unexpected events which ultimately could be handled automatically through

addition of a ground rule or flight rule in the system. Likewise, mission objectives could be updated through an operator planning interface to the deliberative agent. The system was interfaced through I/O servers on the ground and on the flight system.

Figure 3: Goal Architecture for the DATA/CHASER Experiment

The actual DATA-CHASER system implemented and flight tested on STS-85 met all of the semi-autonomous system objectives except for full implementation of the FRTE and SELMON on the flight system. This was largely due to problems encountered with porting the GRTE from a Solaris operating system environment to the embedded RTEMS environment on the flight 68040 microprocessor. The final system is shown in Figure 4.

In addition to software engineering challenges related to implementing and testing the DATA-CHASER architecture, the ability to evaluate the system's real-time performance before and even during/after the mission was lacking. The difficulty in

interfacing device interfaces to the flight reactive agent and the lack of real-time performance assurance was a key motivation for developing the RT EPA. So, overall, DATA-CHASER was a success in demonstrating the distributed multi-agent concept and viability of the architecture as well as providing lessons learned that could be directly incorporated into on-going real-time automation research at the University of Colorado. The problem was that a test-bed similar to DATA-CHASER was needed now to test the new framework for a similar system and one with more opportunity to experiment more fully, so the RACE test-bed was designed.

Figure 4: Actual DATA/CHASER Shared-Control System Flown on STS-85

6.0 RACE TEST-BED OPTICAL NAVIGATION EXPERIMENT RESULTS

The RT EPA made implementation of the RACE optical navigation experiment simple. The full source code is not provided here, but the overall driver for the multi-pipeline system was less than 4 thousand lines of C source code, much of it the NTSC decoder DMA micro-code for the video driver and the entire multi-pipeline system shown in Figure 5 was implemented in several hundred lines of code. While it is clear that the RT EPA affords software engineering advantages, the real-time performance advantages it provides are perhaps even more important.

A very difficult aspect of implementing and testing real-time pipelines is predicting and verifying real-time performance. In order to evaluate the RT EPA framework, five execution performance monitoring and control goals were defined for the RACE experiment:

- 1) Demonstration of the admission and execution control of marginal thread sets with deadline reliability specification.
- 2) Demonstration of on-line kernel monitoring and use of on-line models as a basis to re-negotiate for a new service level based on actual performance.
- 3) Demonstration of RT EPA pipeline phasing control (goal 3a) and control to meet isochronal output requirements (goal 3b).
- 4) Establish viability of ME theory by showing scheduling feasibility for a real-world system that otherwise cannot be scheduled.
- 5) Demonstrate protection of services from deadline over-runs (i.e. fire-walling of services from each other).

All five goals were met and demonstrated with RACE [Si00]. The results for goals 1, 3b, and 5 are reviewed here since these results demonstrate the execution monitoring and control capabilities of the

RT EPA well and goal 4 is reported on in detail elsewhere [Si00] [SiNu00] as are goals 2 and 3a [Si00].

The expected execution times in RACE lead to a loading of approximately 95%, but given the low confidence required on many of the threads, this is the reason that the thread set can be admitted by the RT EPA despite the high average loading. No matter how one interprets execution time, in all cases the loading is above the RM least upper bound of 72.05% for 9 threads [LiuLay73], showing that the RT EPA can schedule such systems as goal 1, a primary goal. Furthermore, the RT EPA overrun control also makes this otherwise marginal thread set feasible since overruns are terminated and therefore interference controlled when such an execution fault occurs (goal 5). The S array in Table 1, taken from the RT EPA on-

line admission test that accounts for utility and interference over each thread deadline is provided here. Intuitively, the threads with the largest S values have the highest probability of missing a deadline – a high S value thread with high confidence is the most likely point of failure in maintaining negotiated service. Clearly, from a worst-case execution standpoint, the RACE experiment is a marginal task set for which the scheduling feasibility is questionable. Ideally such marginal task sets would be avoided, but given aerospace computing system constraints on power, radiation hardening, and mass/size, combined with the high-value of data return this may be unavoidable – the SIRT/MIPS space telescope payload processing system is a clear example of why being able to deal with marginal task sets may in fact greatly increase mission value [SiNu00].

Figure 5: RACE Semi-Autonomous Multi-Pipeline Architecture

Table 1: RACE Source/Sink Pipeline Task Set Description

id	Name	Low Conf	High Conf	T (msec)	Cexp (μsec)	Exp Util	Clow (μsec)	Low C Util	C wc (μsec)	WC Util	S
0	tBtvid	1.0	1.0	33.333	64	0.002	0	0	1200	0.036	0.02
1	tFrmDisp	0.5	0.9	100.00	38772	0.388	36126	0.361	40075	0.400	0.82
2	tOpnav	0.9	0.99	66.67	20906	0.314	19545	0.293	23072	0.346	0.96
3	tRACECtl	1.0	1.0	66.67	190	0.003	0	0	1272	0.020	0.97
4	tTlmLnk	0.2	0.5	200.00	384	0.002	0	0	1392	0.007	0.77
5	tFrmLnk	0.5	0.8	500.00	55362	0.111	50083	0.100	58045	0.116	0.91
6	tCamCtl	1.0	1.0	200.0	610	0.003	317	0.001	1530	0.008	0.89
7	tNet	0.5	0.25	100.00	8000	0.080	4000	0.040	10000	0.100	N/A
8	tExc	1.0	1.0	1.0	50	0.050	25	0.050	50	0.050	N/A
						0.953		0.845		1.073	

If we now look closely at plots of the RACE RT EPA kernel monitoring results in Table 2, we see that in fact even with a worst-case loading somewhere between 0.845 and 0.953, there is still pessimism in the critical instant assumption and the actual loading is much less. This example shows how observed worst-case execution times are pessimistic and how actual reliability is maintained over large sample sizes (2300 33.33 msec periods for this data). So, the RT EPA provides not only an interface for ensuring not only that deadlines can be met given worst-case execution

and desired deadline reliability, but that the actual on-line reliability meets or exceeds the service specification as it does in these results. Finally, more importantly, if a timing worst-case situation does evolve, the RT EPA will preserve reliability by fire-walling services from each other at their negotiated levels of service and reliability for the given worst-case model provided. The RT EPA provides the fire-walling by terminating services that would otherwise over-run hard deadlines and create unaccounted for interference.

Table 2: RACE Source/Sink Actual Performance

rtid	Name	Soft Rel.	Hard Rel..	Dsoft (msecs)	Dterm (msecs)	T (msecs)	Cexp (μsec)	DM Utility	RM Utility
0	tBtvid	1.0	1.0	40	50	33.333	77.9	01.56	02.34
1	tFrmDisp	1.0	1.0	50	50	100.000	9840	19.68	09.84
2	tOpnav	1.0	1.0	66	66	66.67	21743	32.94	32.94
3	tRACECtl	1.0	1.0	66	66	66.67	1200	01.80	01.80
4	tTlmLnk	1.0	1.0	150	200	200.00	2000	01.00	01.00
5	tFrmLnk	1.0	1.0	400	500	500.00	55028	11.01	11.01
6	tCamCtl	1.0	1.0	1000	1000	1000.00	583	00.06	00.06
								68.5	58.99

Before discussing the fire-walling results, given normal operation the RT EPA provides pipeline phasing control to better meet timing constraints. The RT EPA stage-to-stage isochronal release phasing control (goal 3b), is demonstrated by Figures 6a and

6b. In Figure 6a, the video link response jitter is uncontrolled such that the jitter from this stage will result in sink output or next stage release period jitter. However, in Figure 6b, the isochronal hold output control feature of the RT EPA was enabled and the result is that response jitter is greatly minimized.

Figure 6 A and B: Before and After Phasing Control

In order to protect pipeline services from expected over-runs by services with less than full reliability, the RT EPA bounds the interference due to an overrun to the specified termination deadline (goal 5). Over that period it is possible to assume that the release will attempt to use either the full resources of the period (worst-case assumption) or its typical resource demands, but either way it has not completed by the termination deadline due to one of the following conditions:

- 1) lack of I/O resources
- 2) lack of CPU resources
- 3) lack of both I/O and CPU resources
- 4) atypical execution jitter due to algorithmic or architectural variance

In order to demonstrate the RT EPA ability to protect the system from occasional or malfunctioning service termination deadline overruns, the RACE test-

bed was run and an artificial interference introduced by requesting a color frame to be dumped without going through task admission. This unaccounted for interference to the frame link and camera control task resulted in overruns for both of those services. Figure 7 shows that two missed deadlines occurred due to the unaccounted for interference, but that the system continued to function after those isolated misses. The assumed interference by the RT EPA over the miss was configured for expected execution time for the thread. Looking carefully at the releases around the miss we see that while the execution time of the miss was much higher than normal, due to the nature of TCP/IP packetization and interference by the dump to the same channel, the dropout due to restarting caused the overall interference to average out close to expected execution time. This test was more complicated than simple CPU interference since both the frame link thread and the interfering frame dump request not only were competing for CPU, but also for the *ethernet* interface. Despite this complication, the RT EPA was able to control the overrun.

Figure 7: Frame Link Termination Deadline Miss Control

Similarly, if a particular task were to malfunction or be misconfigured, then the RT EPA protects other services from the misconfigured task, which rather than occasionally missing deadlines, may continually miss deadlines while misconfigured. It should be noted that the deadline confidence for the termination deadline dropped below the requested 0.9 to 0.85 due to the period of misconfiguration. Furthermore, the actual reliability in the deadline was 0.71. If the misconfiguration had been allowed to continue, eventually the confidence would have dropped to zero if all actual execution times exceeded the desired deadline. The reliability is based on number of missed deadlines over all samples taken and the confidence is based on the number of samples out of all on-line samples that are within the deadline. Since the misconfiguration was allowed to persist for

approximately 150 releases with an on-line model size of 100, the computation of the confidence is straightforward. Furthermore, since the number of samples was less than the on-line model size (523 samples) and the initial model was a normal model instead of distribution-free, this explains why the reliability was lower than the confidence (all initial values in the model are set to zero unless a distribution free model is loaded). What is also very interesting is that after the misconfiguration, there is an execution and response time hysteresis. This is most likely due to a newly evolved L1/L2 cache reference stream and/or dispatch context after the period of higher loading since the hysteresis exists in both the execution time and response time. This particular task has almost no interference since it is one of the shortest deadline and therefore highest priority tasks in the system.

Figure 8: Misconfiguration Execution Variance Example

In this case, a useful extension to the RT EPA would be to provide for a restart policy on occasional misses with a secondary dismissal policy for miss trends. This is not currently a feature of the RT EPA, but would be a simple extension.

7.0 FUTURE WORK

Future work for the RT EPA framework includes:

1. Extension of the RT EPA for generalized ME scheduling as discussed in [SiNu00].
2. Implementation of the RT EPA in typical real-time ground systems such as Solaris and Linux/RT.

3. Extension of the RT EPA to include I/O resource monitoring and control.
4. Extension of the RT EPA for end-to-end performance optimization in distributed real-time systems.

Future work goal 1 is in progress. Goal 2 feasibility has been assessed as has 3 [Si00]. Goal 4 has only recently been reviewed, but quality of service in distributed systems has been widely researched and it is envisioned that the RT EPA will provide node performance control in networks such that services can be better matched to real-time transport protocols.

8.0 CONCLUSION

The implementation of shared-control semi-autonomous system architectures is complicated due to the need to interface many source devices, processing, and sink devices which all have to function within timing deadlines. The RT EPA framework provides a method for more directly mapping processing and timing requirements into an application implementing a real-time shared control architecture. Furthermore, it provides execution fault tolerance and performance monitoring with small overhead (less than 1% in RACE experiments) with fire-walling so that hard and soft real-time services can safely coexist on one microprocessor system. It is envisioned the application of the RT EPA to a wide range of mixed hard and soft real-time service systems is possible and will greatly enhance reliability and reduce implementation time and risk.

9.0 REFERENCES

- [Au93] Audsley, N., Burns, A., and Wellings, A., "Deadline Monotonic Scheduling Theory and Application", *Control Engineering Practice*, Vol. 1, pp 71-8, 1993.
- [BriRoy99] Briand, Loïc and Roy, Daniel, Meeting Deadlines in Hard Real-Time Systems – The Rate Monotonic Approach, IEEE Computer Society Press, 1999.
- [Bru93] Brunner, B., Hirzinger, G., Landzettel, K., and Heindl, J., "Multisensory shared autonomy and tele-sensor-programming - key issues in the space robot technology experiment ROTEX", *International Conference on Intelligent Robots and Systems*, Yokohama, Japan, July, 1993.
- [Carlow84] Carlow, Gene, "Architecture of the Space Shuttle Primary Avionics Software System", *Communications of the Association for Computing Machinery*, Vol. 27, No. 9, September, 1984.
- [Dec99] Decoste, D., "Adaptive Resource Profiling", *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.
- [Gov91] Govindan, R., and Anderson, D., "Scheduling and IPC Mechanisms for Continuous Media", *13th ACM Symposium on Operating Systems Principles*, 1991.
- [LiuLay73] Liu, C., and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the Association for Computing Machinery*, pp. 46-61, Vol. 20, No. 1, January 1973.
- [McCart00] McCartney, C., "DirectX Display Drivers – DirectX 7.0 and Beyond", *Windows Hardware Engineering Conference*, New Orleans, April 25, 2000.
- [NayKu99] Nayak, P., Kurien, J., Dorais, G., Millar, W., Rajan, K., Kanefsky, R., Bernadr, D., Gamble, E., Rouquette, N., et al, "Validating the DS-1 Remote Agent Experiment", *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.
- [NuBra99] Nutt, G., Brandt, S., Griff, A., Siewert, S., Berk, T., Humphrey, M., "Dynamically Negotiated Resource Management for Data Intensive Application Suites", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12, No. 1 (January/February 2000), pp. 78-95.
- [RabKni99] Rabideau, G., Knight, R., Chien, S., Fukunaga, A., Govindjee, A., "Iterative Repair Planning for Spacecraft Operations Using the ASPEN System," *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.
- [Si00] Siewert, S., "A Real-Time Execution Performance Agent Interface for Confidence-based Scheduling", *Ph.D. Dissertation*, Dept. of Computer Science, Univ. of Colorado Boulder, 2000.
- [SiHan96] S. Siewert and Elaine Hansen, "A Distributed Operations Automation Testbed to Evaluate System Support for Autonomy and Operator Interaction Protocols", *4th International Symposium on Space Mission Operations and Ground Data Systems*, Munich, Germany, September 1996.
- [SiHan97] S. Siewert, E. Hansen 1997. "Lowering the Cost of Mission Operations Through End-to-End Automation", *International Symposium on Artificial Intelligence*,

Robotics, and Automation in Space, Tokyo, Japan, June 1997.

- [SiNu00] Siewert, S., Nutt, G., “Multi-Epoch Scheduling Within the Real-Time Execution Performance Agent Framework”, *IEEE Real-Time Systems Symposium*, Orlando Florida, November, 2000.
- [SiNu96] Siewert, S., and Nutt, G., “A Space Systems Testbed for Situated Agent Observability and Interaction”, *Second ASCE Specialty Conf. on Robotics for Challenging Environments*, Albuquerque, New Mexico, June 1996
- [SiNuHa99] Siewert, Sam, Gary Nutt, and Elaine Hansen, "The Real-Time Execution Performance Agent: An Approach for Balancing Hard and Soft Real-Time Execution for Space Applications," *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.
- [SiNuHu97] S. Siewert, G. Nutt, M. Humphrey, An Execution Performance Agent Interface to Parametrically Controlled In-kernel Pipelines, *IEEE Real-time Technology and Applications Symposium*, Montreal, Canada, copyright IEEE, 1997.
- [Tör95] Törngren, M., “On the Modelling of Distributed Real-Time Control Systems”, *Proceedings of 13th IFAC Workshop on Distributed Computer Control Systems*, Toulouse-Blagnac, France, Sept. 1995.
- [WeiRod99] Weisbin, C., Rodriguez, G. Schenker, P., Das, H., Hayati, S. Baumgartner, E., Maimone, M., Nesnas, I., Volpe, R., “Autonomous Rover Technology for Mars Sample Return”, *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.
- [WyShe99] Wyatt, J., Sherwood, R., Sue, M., Szijjarto, J., “Flight Validation of On-Demand Operations: The Deep Space One Beacon Monitor Operations Experiment”, *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, June, 1999.

ABOUT THE AUTHOR

Dr. Sam Siewert is a member of technical staff with the Optical Networking Group at Bell Labs – Lucent Technologies and also teaches as adjunct Electrical and Computer Engineering faculty at the University of Colorado. Dr. Siewert has published numerous papers on space system operational architectures, real-time theory, embedded applications, and distributed systems. During 1997-2000, Dr. Siewert worked at Ball Aerospace in Boulder on SIRTf (Space Infrared Telescope facility) developing software for the MIPS (Multi-band Imaging and Photometer for SIRTf) instrument being built by the University of Arizona and Ball Aerospace. SIRTf launches in 2002 and will be put into an Earth following orbit about the Sun. As a computer science Ph.D. student at the University of Colorado, Dr. Siewert was a graduate research assistant with the Colorado Space Grant College where he was the software lead on the End-to-End Mission Operations Systems Software for a Space Shuttle Hitchhiker Payload that flew on STS-85 in August 1997. Prior to graduate studies at the University of Colorado, Dr. Siewert worked for McDonnell Douglas Astronautics Corporation on Guidance, Navigation & Control software for Space Station, the Space Shuttle, and the Aeroassist Flight Experiment. Dr. Siewert received a B.S. in Aerospace Engineering from the University of Notre Dame in 1989.