System Mode Changes - General and Criticality-Based

A. Burns
Department of Computer Science,
University of York, UK.
Email: alan.burns@york.ac.uk

***Abstract*—In this paper we summarise, and attempt to unify, the many descriptions that have been published on general mode changes. We then use this summary to position the criticality mode change. We conclude that a criticality mode change (from low to high) is closest in nature to a (graceful) degradation mode change following (partial) system failure. However, a criticality mode change (from high to low) has more in common with a (exceptional) functional mode change. The paper also addresses systems that may have both criticality and general mode changes.**

I. Introduction

Many real-world applications involve systems that operate in a number of clearly defined *modes*. Aircraft flights progress through phases (e.g. taxiing, take-off, climbing, level flight, etc) and automotive systems have modes to cover start-up, cruise control, driver control, 'limp home' etc.

If a system has more than one mode then there must be a *mode change protocol* to control how the system moves between modes. Such (general) protocols have been the subject of considerable study over a number of years [26], [10], [30], [20], [29], [13], [23], [21], [1], [11].

The more recent literature on supporting mixed criticality systems has identified situations in which the system must move from one criticality level to another [31], [6], [5], [12], [24], [15], for example, in a dual criticality system (low and high) a move from the low criticality mode to the high criticality mode. As a consequence of this move some low criticality work is abandoned (either temporally or permanently).

This type of criticality mode change has a number of similarities with the more general mode changes, but there are some important differences. The abandonment of work (even of a lower criticality) is clearly unacceptable in a fully functioning system, but it may be acceptable as part of a response to a (partial) system failure. In this paper we argue that a criticality mode change is equivalent to a particular form of general mode change (providing graceful degradation).

The paper is organised as follows; next we review the models and forms of analysis for general mode changes. In Section III, we use this context, to position the definition of a *criticality mode change*. Systems with both general and criticality mode changes are considered in Section IV. Conclusions are provided in Section V.

II. General Mode Changes

To formalise what it means for a system's behaviour to be described in terms of modes, a number of aspects need to be covered:

- Type – what are the different classes of modes.
- Trigger – what causes a mode change.
- Protocol – how is the mode change managed.
- Attributes – properties of modes.
- Definition – the software, and its operational parameters, that constitute a mode.
- Analysis – in particular the scheduling analysis used to verify the timing properties of the system during a mode change.

We look at each of these in turn. We assume a standard system model in which periodic and sporadic tasks, characterised by their minimum inter-arrival time, deadline and one or more measures of their worst-case execution time, give rise to a potentially unbounded sequence of jobs.

A. Type

In the literature on mode change protocols (cited above) three distinct types of mode can be identified. These provide a natural partitioning of the functionality of the system:

1) Normal Functional Modes – the application moves through a number of different phases. These phases are planned and are entered regularly. An example would be moving from driver control to cruise control in a standard family car.
2) Exceptional Functional Modes (sometimes called *Operational mode*) – rare events that will cause code to be executed that is not otherwise required. The response is planned, but the resulting mode change may never occur. An example would be 'prepare for crash' mode contained within a car – when the on-car monitoring system detects that an impact may be about to happen, it winds up the windows, tightens the seat belts, applies the brakes so the pads are just in contact with the disks (making them more responsive if applied by the driver) and prepares to deploy the airbag.
3) Degraded Functional Modes (sometimes called simply *Graceful Degradation*) – errors require load to be shed and priority given to issues of safety and minimum functionality. General responses to mode change events

are planned but the full set of error conditions may not be known in advance. An example would be a 'limp home' mode following engine sensor failure.

A system might have ten or more normal modes that are progressed through in a statically defined sequence. It might also have a small number of exceptional modes, and perhaps one or two degraded modes. With more complex systems, modes might be organised hierarchically.

B. Trigger

The mode change *event* (sometimes called *request* or *trigger*) is typically related to the state of the system or the system's environment as indicated via an input reading or an internal state change. For example, a driver touching the brake peddle will generate an event that will move the engine control system from 'cruise control' to 'driver control'.

An event could be *timed-trigger* if the mode change is coordinated to the 'time' of the environment. So a power generation system may switch modes at midnight. Also air traffic control systems have day and night modes (which switch over at a particular time – or at least should do, failure to return to 'day' mode being a cause of system failure in the past (see http://www.bbc.co.uk/news/uk-25278163). An example of a relative time trigger is a data collection mode that executes for just 10 minutes before returning to some previous mode.

For graceful degradation the trigger may come from the hardware platform or some health monitoring subsystem.

C. Protocol

The mode change event requires a protocol to manage the actual mode change. Such protocols can also be characterised in three ways:

1) Immediate – the mode change event causes an immediate mode change with the old mode jobs being *suspended* or *aborted*, and new mode jobs starting immediately.
2) Bounded (sometimes called *synchronous* [21]) – within a bounded time from the mode change event a point in time is reached in which there are no active jobs from the old mode and hence a clean switch of modes is then possible.
3) Phased (sometimes called *asynchronous*) – following the mode change event old mode jobs are allowed to complete, and new mode jobs are started within a bounded time.

With Immediate and Bounded, the system is only ever in one mode; with Phased there is a (limited) interval of time in which the modes are overlapping. Some jobs from the new mode have started while other old-mode jobs are yet to complete.

Overlapping can also happen in a distributed or multiprocessor system in which a phased change is necessary as it is not possible to simultaneously inform the entire system of the need to change mode. The propagation of the mode change event will inevitable take time.

Phased changes are the most difficult to analyse as the load on the system is typically higher during the change than it is in either mode [30]. Analysis can however be used to start new tasks as soon as possible commensurate with all deadlines being met during transition [20]. Here a worst-case scenario is assumed, with all old-mode tasks releasing a job just before the mode change occurs. Each of these jobs is allowed to terminate. Each new-mode task has a temporal offset (from the time of the change event) that the scheduling analysis has furnished. So this offset is the minimum possible that will not undermine schedulability during the mode change.

The completion of the old mode is usually defined to be when all current jobs, released in the old mode, have completed. However in some situations a number of old-mode jobs may need to be executed to complete the work of that mode. For example, a buffer of sensor input values from the old mode may need to be cleared before the mode change can occur [21]. And in distributed systems a series of old-mode messages may be in transit and need to be delivered and processed before the mode can be considered complete.

A single processor system implemented via a cyclic executive can easily support a Bounded mode change by waiting until the end of the current major cycle and then switching the (pre-computed) scheduling tables. A system using fixed priority scheduling or EDF can also support a Bounded change by waiting for the next idle tick and then changing the set of eligible tasks [29]. An idle tick is an instant in time when there is no work to undertake apart from new jobs released at that instant. Clearly there can be no causal effect from before to after an idle tick. For multiprocessor systems the coordination of the mode change across the entire platform is more of a challenge [25].

The definition of the mode change protocol must be closely tied to the form of analysis used to verify the system's behaviour (as indicated below).

D. Attributes

In this subsection we define a number of attributes that have been used to define properties of modes and mode change protocols.

A mode is *re-entrant* if it can be returned to at some time after it was left. Other modes can be termed *one-shot* (or *single-shot*) if they can only be entered once, *sink* if the system never leaves this mode once it has entered it, or *initial* if the system always starts in that mode. The full set of modes is *cyclic* if the system systematically and repeated moves through the modes. Alternatively the set of modes is *connected* (sometime called *strongly connected*) if the system can move from any mode to any other mode. Obviously a connected set does not have any one-shot or sink modes.

A mode that has aborted jobs is not usually re-entrant. A mode which can give rise to suspended jobs can, however, be re-visited with the suspended jobs continuing from the state they were in.

These definitions are useful but do not cover all interesting cases. For example, a system with four modes, A to D, might have the behaviour that it can start in A or B; it moves backwards and forwards between these modes unless some event occurs that moves the system to mode C, A and B are never returned to but the system then moves between C and D. The descriptive terms can be assigned to the pairs of modes but not to the individual modes.

Although the above classifications are independent there is some common coupling: Functional mode changes tend to be Bounded and often re-entrant, Operational changes can be Immediate or Phased and may lead to the use of one-shot modes, and Graceful Degradation may require Phased or Immediate changes, and the degraded mode may well be a sink mode.

A particular case of Graceful Degradation concerns execution time overruns. If a task, due to a software error, enters an infinite loop then the only recovery strategy is to abort the task (its current job must be abandoned). This will require an Immediate change. Later the task could be restarted (*cold restart*) or an alternative task introduced in the new 'mode'. This task could start *cold* (no relevant internal state) or *warm* (if it has access to state updated by the aborted task). A *hot* standby would most probably be present in the old mode, but be deemed more important once it had taken over from the aborted primary task [22].

E. Definition

In terms of the code contained within a mode, a mode change may involve:

- Tasks that run unaffected in both modes.
- Tasks that run only in the old mode.
- Tasks that run only in new mode.
- Tasks that run in both modes but have their defining parameters changed.

In the latter case, a task could have its period and/or deadline altered, and in a fixed priority scheme its priority. A suspended job may actually be allowed to execute at a background priority; hence there is some overlap in these definitions between tasks that only execute in the old mode, and those that run in both modes but with diminished urgency in the second mode.

A task that has the same release characteristics, but which undertakes altered functionality in the new mode may have a different worst-case execution time in the new mode.

Finally, once 'criticality' becomes a task parameter then it is possible for a task to remain unchanged during a mode change, but for its designated criticality to alter. The hot standby introduced above is an example of such a change.

F. Analysis

From a schedulability point of view, different modes have different code requirements. So schedulability in one mode does not imply schedulability in another mode or in any Phased mode change. All modes must be checked, and all Phased changes.

For Immediate mode changes there is no specific scheduling problem, but there is an obligation on the RTOS and/or real-time programming language to facilitate immediate task suspension and/or aborting (which may be quicker). Suspension is needed for re-entrant modes, abort for non re-entrant. If a suspended or aborted job could be holding a resource that is used in the new mode then action must be taken to recover the resource or to allow a 'suspended' task to continue to execute until it has released the resource. From a scheduling point of view the mode change may therefore not be truly immediate.

For Phased changes there has been scheduling analysis produced [20] that computes a set of minimum release offsets for each new mode task. Whenever the mode change event occurs these offsets will ensure that all old mode jobs complete by their deadlines, but new mode tasks start as soon as possible. The scheduling of Phased changes is complicated by tasks that change their periods. A seemingly simple change of a task that moves from requiring 6 ticks of computation every 20 to 3 every 10 (or visa versa) can cause deadline misses on other non-changing tasks.

A final complication with Phased changes comes from the possibility of overlapping phases; e.g. during the move from mode A to mode B, a move to mode C is required. Systems tend to avoid this difficult to analysis situation by not allowing a further change until the current change has been completed. However, it may again be necessary to wait until there is a system idle tick before a Bounded or Phased change can be guaranteed to be complete.

A complex system with a large number of modes and possible mode changes can be modelled using state and state transitions formalisms [21]. Formal analysis can be used to verify that a system always remains within safe modes [1].

III. Criticality Mode Changes

In this section we review the literature on mixed criticality systems (MCS) that has utilised the notion of criticality modes and mode changes.

Consider a system with N criticality levels, $L_0 \dots L_{N-1}$, executing on a uniprocessor and using priority based scheduling of constrained tasks. Perhaps up to five levels of criticality may be identified in a system (see, for example, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards). Typical names for the levels are ASIL (Automotive Safety and Integrity Levels) and SIL (Safety Integrity Level). It should be noted that not all papers on MCSs assign to 'criticality' the same meaning, an issue explored by Graydon and Bate [14].

The standard MCS's model [31], [6], [5], [12], [24], [15] has the following properties:

- Each task in the system is characterised by the minimum inter-arrival time of its jobs (period denoted by T), deadline (relative to the release of each job, denoted by D) and worst-case execution time (one per criticality level), denoted by $C(L_0) \dots C(L_{N-1})$. A key aspect of the standard MCS model is that $L_x > L_y \rightarrow C(L_x) \geq C(L_y)$.
- The system starts in the L_0 mode, and remains in that mode as long as all jobs execute within their low criticality computation times ($C(L_0)$).
- If any job executes for its $C(L_0)$ execution time without completing then the system immediately moves to the

next criticality mode, L_1.

- As the system moves to the L_1 mode all L_0 criticality tasks are abandoned. No further L_0 criticality jobs are executed.
- The system remains in the L_1 mode unless a job executes for its $C(L_1)$ execution time without completing, the system then immediately moves to the next criticality mode; jobs with criticality level L_1 are dropped.
- This process continues (potentially) until the top criticality mode is reached (L_{N-1}) with only tasks of this criticality level executing.
- Tasks are assumed to be independent of each other (they do not share any resource other than the processor).

This abstract behavioural model has been very useful in allowing key properties of mixed criticality systems to be derived, but it has been necessary to extend the model to allow for more realistic characteristics such as allowing some lower criticality work to execute in the higher criticality modes and for the lower criticality modes to be reinstated when conditions are appropriate. This is covered in the following papers [5], [28], [27], [18], [9], [4], [16], [17].

So the standard model (SM) defines a path from L_0 to L_{N-1}. The adaptive model (AM) allows movements in the opposite direction.

Note that work has also been focused on criticality-aware resource control protocols that will allow resource sharing between tasks [7], [32], [19], [33]. This work does not however directly impact mode changing unless resources can be used by tasks of different criticality.

A. *Characteristics of a criticality mode change*

Using the terms introduced in the Section II we can define the above SM criticality mode change protocol as follows

- L_0 is the initial mode.
- L_{N-1} is a sink mode.
- All modes are one-shot.
- Mode transitions are Immediate (or Phased in some models where executing lower criticality jobs are allowed to complete – though usually their deadlines are not guaranteed).
- Following a mode change some tasks only execute in the old mode.
- Some tasks execute in both modes, but their execution times are increased[1].
- There are no 'new mode' tasks.

As discussed above the more expressive and adaptive mode (AM) allows systems to regain functionality and move back towards the initial (fully functional) mode [5], [28], [27], [18], [9]. AM is therefore characterised as follows:

- L_0 is the initial mode.
- There are no sink modes.
- Mode transitions are typically Bounded.
- All modes are re-entrant.
- Some tasks execute in both modes, but their execution times (and periods) are deemed to vary.
- There are new-mode tasks when moving mode in the direction of L_0.

But what type of mode change are these? First for the standard model (SM). Early papers on MCS [31], [6] were clear that the initial L_0 mode is the only expected state for the system to be in. Other criticalities were only introduced so that scheduling analysis can be used to reduce the resource needs of the system. This is done by leveraging the pessimistic execution times assumed for high criticality tasks in the higher criticality modes.

In a two criticality system (LO and HI), these pessimistic values (the $C(HI)$ values) are *not* expected to be experienced at run-time. Indeed the $C(LO)$ values are most likely to also be pessimistic (though less so of course).

Therefore, a task executing for longer than expected (beyond $C(LO)$) can be deemed to be at fault. And hence a criticality mode change should be described as a form of Graceful Degradation. If one accepts this view then of the N modes, only one reflects normal functionality, all the other $N-1$ are forms of degraded service – as increasing levels of functionality are being dropped.

For the adaptive model (AM) mode changes are better defined as exceptional (operational). They are planned but may not occur.

All protocols and forms of analysis that have been developed for general mode changes are directly applicable to criticality mode changes (albeit often in a simpler form as a criticality mode change does not have all the characteristics of the more general protocol). So, for example, in the standard model where L_0 is the initial mode and L_{N-1} is the sink mode, there are no new-mode tasks. But in the more adaptive scheme where lost work can be returned to (i.e. L_{N-1} is not a sink mode) then new-mode tasks will need to be supported.

In the general literature on fault tolerance, recovering from an error (or partial failure) can either be: degraded service followed by active recovery, or degrading service followed by 're-boot' (e.g. channel re-initialisation in an avionics system). With a 're-boot' the system, in effect, moves from the sink mode to the initial mode, but this is done outside the model of the software. With active recovery the system recovers by moving away from the degraded modes, there are no longer sink modes.

For mixed criticality, the standard model (SM) assumes that the software cannot return to L_0. Active recovery requires an adaptive protocol (AM).

IV. SYSTEMS WITH BOTH GENERAL AND CRITICALITY MODE CHANGES

Having established that the main SM criticality mode change is usefully defined as a form of graceful degradation, it seem perfectly reasonable for a large system or system of systems to have both general and criticality mode changes. Some points of interest are:

[1]Some models for MCS have period as well as execution time being criticality dependent [8], [2], [4], [3]; in these models a task's period may reduce (as well as computation time increase) during a criticality mode change.

- Assume the system consists of a set of applications, of potentially different criticality levels.
- A General Mode Change may impact on just one or a subset of applications and therefore criticality levels.
- Graceful Degradation, in general, is most likely to be influenced by criticality.
- A General Mode Change Protocol may involve some tasks changing their criticality designation.

In the latter case a set of tasks may be more critical, say, during take-off than during taxiing. So the same tasks are executing, but are deemed to have different worst-case execution times. Fortunately this is equivalent to the tasks having added functionality and therefore modified worst-case execution times.

If any system uses mode changes in response to component failure then they are bound to use 'criticality' to decide which code to abandon and which to retain. One of the common forms of error detection is to use a watch-dog timer. If some event has not occurred by a fixed time then switch mode and protect the key computations. A task executing for longer than assumed during system verification can be identified via timers; the fault that causes the error could be in hardware or software. Here a criticality mode change and a general mode change are essentially the same thing.

A. Example of a system with both forms of mode change

Consider as an example a simplistic cruise control system that has just three modes: two normal modes, standby (SB) and speed control (SC), and one exceptional, collision avoidance (CA). The following point appertain:

- The system starts in SB with the driver in control of the vehicle.
- Movements between the SB and SC modes are normal.
- The transition to CA is operational.
- Movements between SB and SC are Bounded or Phases.
- The trigger for transition to CA is, however, Immediate.
- In all three modes a task that undertakes proximity analysis executes, this task has a reduced period in the CA mode.

The system software is partitioned between two levels of criticality: SIL4 for the safety critical functions, and SIL2 for the rest. The standby (SB) mode contains mainly SIL2 code. The collision avoidance (CA) mode has predominantly SIL4 code and the speed control (SC) mode has both SIL4 and SIL2 code in approximately equal amounts. All SIL2 code has a WCET based on extensive measurement. All SIL4 code has WCET based on pessimistic static analysis. In addition all SIL4 code also has a SIL2 estimate based only on measurement.

If one focuses on the SIL4 code, as a certification authority might, then there is a three mode system with varying amounts of SIL4 code. Similarly, from the fully functional point of view there is the same three mode system but with both SIL4 and SIL2 code.

From a mixed criticality point of view the system must be schedulable when SIL2 values are used for all code, and the system moves between the three functional modes. Additionally, the system must be schedulable in the SC and CA modes when only SIL4 code is executing and SIL4 WCET values are used.

If only Bounded or Immediate modes changes are used then the system is, at any time, only in one of three normal functional modes. This leads to explicit tests to:

- Check SB in SIL2 mode and SIL4 mode.
- Check SC in SIL2 mode and SIL4 mode.
- Check SC during transition to SIL4 mode
- Check CA in SIL2 mode and SIL4 mode.
- Check CA during transition to SIL4 mode

If however Phased changes are part of the functional design then one would have to (in addition):

- Check Phased changes in SIL2
- Check Phased changes in SIL4
- Check Phased changes with transition to SIL4

This latter case might be difficult to formulate in terms of identifying the worst-case scenario.

What this simple example indicates is that a system has orthogonal functional and criticality modes. And a system can move between functional modes, criticality modes and both at the same time. So with this example, the system could move from SC in SIL2 to CA in SIL4. But it could not move in the opposite direction. All realistic possibilities must therefore be checked as part of the system's verification.

As indicated earlier, simultaneous general mode changes are often prohibited due to the complexity they introduce. Unfortunately the introduction of orthogonal criticality mode changes has re-introduced simultaneous changes.

V. Conclusions

We have surveyed existing mode change models to provide a framework in which:

- Mode change protocols are defined to move a system between Functional modes (normal, exceptional or degraded).
- Mode change events are Immediate, Bounded or Phased.
- Each mode is defined by its tasks, and attributes such as being re-entrant, the initial mode, a sink mode or a one-shot mode (or a combination thereof).
- Tasks can exist in more than one mode, though parameters may be mode specific.
- Some tasks are mode specific.
- During a mode change, tasks may be suspended or aborted.

In the standard model of a criticality change, the proposed protocols are closest in behaviour to:

1) Graceful degradation; i.e. reduced functionality after the change.
2) Immediate or Bounded triggers, with aborted or suspended tasks.
3) Some tasks exist in both modes, but some only in the earlier mode; there are no new-mode tasks.

4) Tasks that exist in both modes may have their (worst-case) computation times increased and/or their periods decreased, and/or their criticality changed.

For papers that have attempted to define a more adaptive criticality mode change protocol, the behaviours are different:

1) The initial mode is normal, others are considered exceptional.
2) Bounded triggers are used, with suspended tasks.
3) Some tasks exist in both modes, new-mode tasks are present when changing mode in a direction toward the initial mode.
4) Tasks that exist in both modes may have their (worst-case) computation times, periods or criticality levels changed.

This difference underpins discussion that have occurred at workshops and seminars on mixed criticality. Low criticality work is still 'critical' and so cannot be abandoned lightly. The standard model appears to happily abort mission critical work. This has lead researchers to focus on adaptive schemes that minimise the harm done to this work. But the standard model does not advocate abandonment; rather it gives structural support to a form of graceful degradation following a timing error. It ensures that following a timing error the higher critical work can still be guaranteed. The more adaptive models should be seem as providing fault tolerance and error recovery.

In general, a system will be in both a functional mode and a criticality mode. But there will be some functional modes that have only one criticality; and some modes will be the target of graceful degradation both because of functional failures and execution time overruns.

Acknowledgements

The research described in this paper is funded, in part, by ESPRC (UK) grant, MCC (EP/K011626/1). The contents of this paper have benefited from fruitful discussions with Sanjoy Baruah.

References

[1] R. Alur, A. Trivedi, and D. Wojtczak. Optimal scheduling for constant-rate multi-mode systems. In *Proc. of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 75–84. ACM, 2012.

[2] S.K. Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *Proc. 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 31–38, 2012.

[3] S.K. Baruah. Response-time analysis of mixed criticality systems with pessimistic frequency specification. Technical report, University of North Carolina at Chapel Hill, 2013.

[4] S.K. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky, editor, *Proc. of Reliable Software Technologies - Ada-Europe 2011*, pages 174–188. Springer, 2011.

[5] S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[6] S.K. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[7] A. Burns. The application of the original priority ceiling protocol to mixed criticality systems. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 7–11, 2013.

[8] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011.

[9] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. WMC, RTSS*, pages 1–6, 2013.

[10] A. Burns and T.J. Quiggle. Effective use of abort in programming mode changes. *Ada Letters*, 1990.

[11] P. Ekberg, M. Stigge, N. Guan, and W. Yi. State-based mode switching with applications to mixed criticality systems. In *Proc. WMC, RTSS*, pages 61–66, 2013.

[12] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic task systems. In *ECRTS*, pages 135–144, 2012.

[13] P. Emberson and I. Bate. Minimising task migrations and priority changes in mode transitions. In *Proc. of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 07)*, pages 158–167, 2007.

[14] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *Proc. WMC, RTSS*, pages 19–24, 2013.

[15] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *IEEE RTSS*, pages 13–23, 2011.

[16] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. Technical Report 350, ETH Zurich, Laboratory TIK, 2013.

[17] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Singapore, Jan 2014.

[18] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. In *Proc. WMC, RTSS*, pages 43–48, 2013.

[19] K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *IEEE RTAS*, pages 47–56, 2011.

[20] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *10th Euromicro Workshop on Real-Time Systems*, pages 172–179. IEEE Computer Society, 1998.

[21] L.T.X. Phan, I. Lee, and O. Sokolsky. A semantic framework for mode change protocols. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 91–100, 2011.

[22] D. Powell. Failure mode assumptions and assumption coverage. In *Proc. 22nd Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, pages 386–95. IEEE Computer Society Press, 1992.

[23] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new protocol. *Journal of Real-Time Systems*, 26(2):161–197, 2004.

[24] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 155–165, 2012.

[25] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In *Proc. RTNS*, pages 183–192. ACM, 2013.

[26] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven premptive scheduling. *Journal of Real-Time Systems*, 1(3):244–264, 1989.

[27] H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE, pages 147–152, 2013.

[28] H. Su, D. Zhu, and D. Mosse. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *Proc. RTCSA*, 2013.

[29] K. Tindell and A Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politecnica de Madrid, 1996.

[30] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority preemptive scheduled systems. In *Proc. Real Time Systems Symposium*, pages 100–109, Phoenix, Arizona, 1992.

[31] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

[32] Q. Zhao, Z. Gu, and H. Zeng. Integration of resource synchronization and preemption-thresholds into EDF-based mixed-criticality scheduling algorithm. In *Proc. RTCSA*, 2013.

[33] Q. Zhao, Z. Gu, and H. Zeng. HLC-PCP: A resource synchronization protocol for certifiable mixed criticality scheduling. *Embedded Systems Letters, IEEE*, 6(1), 2014.