Modeling Mixed-critical Systems in Real-time BIP

Dario Socci, Peter Poplavko, Saddek Bensalem, Marius Bozga

Modeling Mixed-critical Systems in Real-time BIP

Dario Socci, Peter Poplavko, Saddek Bensalem and Marius Bozga
UJF-Grenoble 1
CNRS VERIMAG UMR 5104,
Grenoble, F-38041, France
{*Dario.Socci* | *Petro.Poplavko* | *Saddek.Bensalem* | *Marius.Bozga*}*@imag.fr*

***Abstract*—The proliferation of multi- and manycores creates an important design problem: the design and verification for mixed-criticality constraints in timing and safety, taking into account the resource sharing and hardware faults. In our work, we aim to contribute towards the solution of these problems by using a formal design language – the real time BIP, to model both hardware and software, functionality and scheduling. In this paper we present the initial experiments of modeling mixed-criticality systems in BIP.**

I. Introduction

The introduction of many-cores and multi-cores is leading to an increasing trend in embedded systems towards implementing multiple subsystems upon a single shared platform. However, in most applications, not all the subsystems are equally critical. Especially this observation is important when human lives depend on correct functionality, e.g. in avionics systems. In mixed criticality systems different degrees of failures, from minor, hazardous to major, need to be distinguished [1]. The previous work mostly assumes time or space isolation of subsystems having different levels of criticality. However, when integrating different subsystems on a single multi-core die there is a need to share hardware resources (processors, on-chip memory, and global interconnect) between different subsystems. Also, handling safely the hardware failure is another design problem, that will only increase as multi-core will become more and more commonplace. This problem already manifested itself in popular many-core systems – the GPUs – so it is relatively well studied how to manage the resource sharing and safety when all subsystems have the same level of criticality. However, adding the mixed criticality assumption may easily boost the complexity from tractable to intractable [2], and a general lack of design methodology can be stated.

A popular language for programming safety-critical systems is Ada, and a great interest exists today to express multi-core and especially mixed-critical applications in this language [3], [4]. However, for verification of safety properties (*i.e.,* automatic check for absence of bugs), any program has to be *translated* to a formal model, which is a non-trivial task even for Ada, which was designed to facilitate an easier static analysis of code [5]. Formalization is required not only for analysis of safety properties, but also for timing [6]. Formal models play important role in the analysis of hardware faults and fault correction [7] and of the shared resource conflicts in multicores [8]. Therefore in our mixed-criticality project we target many-core systems addressing the technological challenges by using a formal design language – BIP. The design input can be either provided in BIP or obtained by translation from other languages.

A wide range of formal design languages exist, but most of them, referred to as models of computation, enable tractable analysis in exchange of lack of expressiveness. The software-based embedded systems would ideally be designed similarly as hardware, i.e., using a language such as Verilog/VHDL, for which all important physical properties like timing, consumed energy, occupied space can be formally imposed and/or derived in a fully automated design process. This is unfortunately very often not the case for the software and it requires a significant effort up to even a change in mentality of software developers to write programs that are 'aware' of non-functional constraints [9]. Synchronous languages, such as Lustre [10], are an important step in the direction of solving this problem, and they are actively developing in the direction of multi-core mapping [11] and are becoming an important subject of research for mixed critical systems [12]. Also application written in widely used data-flow languages as Simulink can be translated into synchronous languages [13]. However, unlike their hardware-language 'brothers', Verilog/VHDL, for software the synchronous languages by far do not present a 'one-size-fits-all' solution, because they assume very specific properties of the system behavior, and it can be very difficult and costly to tailor a given software project to fit these properties [14]. Therefore, rigorous embedded system design frameworks, such as BIP, do not restrict themselves to synchronous languages and offer themselves more openly and in more general way to the functionality to be implemented in various safety-critical systems. At the same time they share with synchronous languages the ability to reason on the behavior formally and the potential to achieve full automation for the given physical constraints in terms of timing, energy and space/weight.

The BIP framework is expressive enough to model various models of computations. Due to its unique expressiveness, it takes a very special role in our design methodology. The same language is used to express both the application and the hardware, timing and functionality, scheduling and mapping. The paradigm of updating and analyzing a homogeneous intermediate formal model of a real design object to support the design decisions is a well-recognized paradigm in the field of electronic design automation in hardware design. The tools for logic synthesis and physical synthesis exploit so-called timing graphs, which provide an intermediate timing model of the digital logic design, being updated in conjunction to the modifications made in the design by the design flow and being used to guide the decisions made in the flow. The idea to use

The research leading to these results has received funding from **CERTAINTY** – European Community's Seventh Framework Programme [FP7/2007-2013], grant agreement no. 288175.

some sort of timing graph to express the application, mapping and scheduling for the multicore applications is less widely known, but there are such example, *e.g.,* [15].

In this paper we first present the BIP framework in general, and then present our current work on modeling the mixed-criticality systems in BIP.

II. BIP Component framework

A. General BIP

The BIP (Behavior Interactions Priorities) framework [16] builds around a component-based language. This language enjoys a simple syntax with clear and expressive semantics. At the heart of BIP lies an idea to use as few as possible different kinds of building blocks. It is well-known that so-called finite-state machines are 'bricks' used to construct the operational semantics of many other more complex programming models and are widely used for formal validation of software and hardware. Therefore, BIP directly uses this basic concept in its language.

BIP supports a component-based modeling methodology based on the assumption that components are obtained as the superposition of three independent layers, that is:
1. Behavior, specified as a set of finite-state machines (basic components)
2. Interactions, used to coordinate the actions of behavior
3. Priorities, used to schedule among multiple enabled interactions

The states inside the components denote control locations where the components wait for interactions. A transition is an execution step from one control location to another. Each transition has an associated condition that enables this transition and an action that is executed at this transition. In BIP, all actions executed by transitions are written in C/C++, a popular and efficient programming language supported by most of the mature professional embedded systems.

Multiple components run concurrently and execute interactions with each other. A transition in every component is only executed when some interaction for this transition is enabled. An interaction can occur in two situations: when all involved components are ready to participate (strong synchronization) or when a component triggers the interaction without waiting for other components (broadcast). Every interaction can result in data transfer between the components. The valid interactions are formally defined by algebraic expressions, enabling a small but provably complete and powerful mechanism to define various synchronous and asynchronous communication protocols [17].

To filter amongst possible interactions, the designer can specify priorities between simultaneously enabled interactions. Interactions and priorities define a clean and abstract concept of composition glue. The glue in BIP is thus a first class concept with well-defined semantics that can be analyzed and transformed. Moreover, it enables expressiveness unmatched by any other existing programming model for concurrent systems [18].

BIP supports the construction of composite, hierarchically structured (sub-)systems. It lets developers compose systems

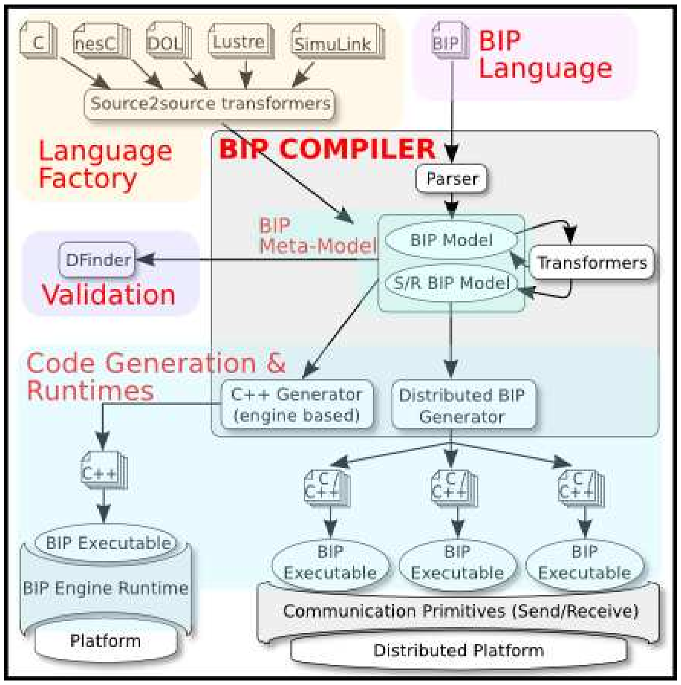

Fig. 1. BIP Toolset

by layered application of interactions and priorities. There is a clear separation between behavior (the finite-state machines) and composition glue (stateless interactions and priorities).

The BIP and its real-time extension RT-BIP are currently supported by an extensible toolset including a concrete modeling language together with associated analysis and implementation. The BIP language leverages on C++ style variables and data type declarations, expressions and statements, and provides additional structural syntactic constructs for defining component behavior, interactions and priorities.

The BIP framework provides constructs for dealing with parametric and hierarchical descriptions as well as for expressing timing constraints associated with behavior. The toolset allows functional validation, model transformation and code generation features. See the illustration of the toolset in Figure 1. In particular, code generation targets both simulation and execution (*e.g.,* distributed, multi-threaded, real-time, etc.). It is important to note that both of them are driven by specific middleware, the so-called engines, available for both BIP and RT-BIP, for a single platform and for distributed set of platforms. This allows to run, explore and inspect execution traces corresponding to systems. Also, the BIP toolset supports translations from various input languages.

B. Real-time BIP

Correct deployment of systems where multiple real-time applications run on a given hardware platform remains by far an open problem. A key challenge is meeting safety and timing constraints, whose satisfaction depends on the features of the execution platform, in particular its speed and the run-time variability of hardware characteristics such as probability of hardware faults. Existing rigorous deployment techniques are applicable to specific classes of systems *e.g.,* with periodic tasks (*e.g.,* rate-monotonic analysis) and deterministic systems (*e.g.,* synchronous dataflow). When mixing different criticality levels on a single platform, different deployment setups should be combined, and one cannot consider them in isolation or otherwise the isolation mechanisms themselves should be rigorously modeled and verified.

Real-time (RT) BIP [19] is an extension of the BIP component-based design language to continuous time model closely related to timed automata [6]. In addition to offering syntax and semantics for the timing-aware modeling of concurrent systems, the real-time BIP also envisions a general model-based implementation method for safety-critical multi-core systems. This method is based on the use of two models: (1) an abstract model representing the behavior of real-time software with user-defined timing constraints; (2) a physical model representing the behavior of the real-time software running on a given platform. The former is obtained directly from the specification provided by the user. The latter is derived from augmenting the software model with the detailed models of the processor and memory hardware blocks, services provided by on-chip communication networks, and the runtime software libraries/kernels/schedulers. A necessary condition for a correct deployment is time-safety, that is, any timed execution sequence of the physical model is also an execution sequence of the abstract model, thus meeting all the deadlines. The time safety means that the platform is fast enough to meet the timing requirements [19]. Also, if the time safety property is preserved while reducing the execution times, then the system is said to be *time robust*. It is the physical model that is used for the final validation of the given design for time safety. For a time robust system a simple simulation with worst-case execution delays of actions is enough to validate the time-safety, due to monotonic dependency of all system timing on the delays of system components [19]. Sufficient static analysis conditions are given in [19] to check whether the system is time robust.

In Section III by means of an example, we show how modeling and analyzing the mixed-critical systems in RT-BIP. This example is composed manually, but according to a particular architecture pattern for mixed-critical systems. We use this example to introduce the elements of the BIP language on the fly as we construct the example. Having constructed the BIP model, we use the available RT-BIP engine to simulate all possible simple scenarios after which the system comes back into the original state. Because each scenario in this example is time deterministic, this simulation is sufficient to prove the time safety, showing the usefulness of the RT-BIP engine for at least partial validation of mixed-critical systems. Moreover, for non-preemptive variant of the considered systems, the engine can be also used for their implementation.

III. Mixed-Critical Systems in BIP

In this section, we describe our current approach to model mixed criticality systems in BIP. This approach follows the architectural pattern shown in Figure 2. There are one or more applications, plugged to the environment via environment interfaces. To provide some fallback possibilities in the case of errors, the applications support different modes, corresponding to different levels of quality of service. The failure of an application means that it produces wrong results to its environment, so they should be all detectable at the environment interfaces, which therefore explicitly signal the failures. To avoid the failures, the system includes so-called observers, which monitor the state of the application at run time and report violations, i.e. situations that are leading to failure conditions, to run-time manager (RT Manager). The function of the latter is granting the hardware resources to different applications with proper scheduling policy and for updating the mode of the applications such that when violations occur the failures are prevented. This is done by acceptable degradation of service of the low-criticality applications which will prevent the failures and degradation of the high-criticality applications.

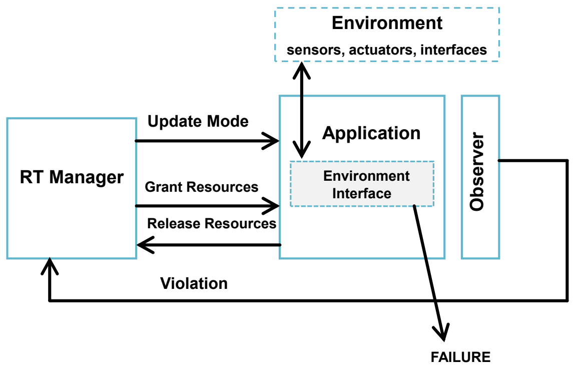

Fig. 2. System overview

A. Modelling Task Systems

Following the architecture pattern in Figure 2, we implemented in BIP the class of mixed-criticality systems originally proposed by [20]. This class is a generalization of the classical real-time scheduling of periodic tasks on a single-core machine. [20] proposes a mixed-critical scheduling approach that ensures certification by certification authorities according to currently existing procedures, hence this approach is called certification-cognizant. The main idea is that the schedule should be such that the high-criticality tasks conform to the requirements of the certification agency, while at the same time all tasks should also conform to (less demanding) conventional requirements posed by system engineers. Finding a feasible schedule in this setup becomes a hard problem to tackle from the point of view of computer science theory. In particular, it is an NP-complete problem when the number of criticality levels is a fixed constant [2], which is always the case in every practical application domain avionics, automotive, etc. (think of five SIL levels in the IEC 61508 standard for safety of industrial system). The bad news from this fact is that it is hard, in general, to find a feasible schedule. The good news is, nevertheless, that when a feasible schedule is found then it can be verified in polynomial time with respect to number of tasks and this can be done by simulating a (polynomial-size) set of alternative scenarios (ideally, with some backtracking to avoid re-exploration of the common parts of similar scenarios). This is exactly the strategy (although, without backtracking yet) we currently follow to verify the physical model which we build in RT BIP for this class of the systems. The notions of (basic) scenarios and the scenario-based verification of mixed-critical schedule are defined formally in [2].

The scheduling problem is defined as a set of periodic tasks, partitioned into two criticality levels: low (2) and high (1). Each task has two values of WCET: $\mathrm{WCET}_{\mathrm{LO}}$ for the lowest criticality level and $\mathrm{WCET}_{\mathrm{HI}}$ for the highest level (we assume only two criticality levels to simplify explanation). The former corresponds to the WCET level obtained with conventional WCET estimation tools, and the latter to the WCET obtained by (sometimes much more pessimistic) tools used by the certification authorities. In addition $\mathrm{WCET}_{\mathrm{HI}}$ can also model problems related to hardware faults. For instance a software module may mask such errors implementing error checking at the end of the execution. If an error is detected the task is executed again. This (exceptional) event will, of course, increase the execution time. Every task is also assigned a criticality level that is 'own' for this task: either level 1 (HI=high) or level 2 (LO=low). Next to this, as in the usual scheduling model, each task has a period and a relative deadline. A schedule is feasible if the following conditions are met:

Condition 1: (Normal mode) If all jobs run at most for their LO WCET, then both critical (HI) and non-critical (LO) jobs must complete before their deadline.

Condition 2: (Degraded mode) If at least one job runs for more then its LO WCET, than all critical (HI) jobs must complete before their deadline, whereas non-critical (LO) jobs may be even dropped.

Figure 3 shows the structure of a BIP model for a simple instance of this model that consists of two tasks. In BIP component communicate via 'ports' (circles and triangles in the figure), that are the transaction of the automata that participate in interactions. To illustrate ports that take part into the same interaction, we connected them with lines.

Every periodic task is placed in the context of an 'application' consisting of three components: the Source, the (relative-deadline) Sink and the task itself. The Source has a clock variable that increments automatically with the passage of time (like any clock in timed automata). We do not show the internal details of the Source, but the behavior is simple. When the value of his clock reaches the period of the corresponding task, the Source executes an interaction at port 'Job_in' and resets the clock. As a result, it executes 'Job_in' periodically. The meaning of 'Job_in' is that a new job starts and it has to be executed by the corresponding task and finished by the task's relative deadline with the respect to the time of arrival of 'Job_in', and the deadline is checked at the Sink. Thus, both the task and the Sink have to be notified on 'Job_in' and we see that they both have a port connected to 'Job_in'.

Figure 4 zooms into the BIP model of a task which we use in this example. The task has three groups of ports (see them on the left). The first group is for starting and finishing a job. It also includes one port that controls the mode in which the current job is running. The second group consists of two ports: Read and Write to read the input data in the beginning of the job and write the output data in the end. The third group is for obtaining and releasing the CPU resource from the scheduler of the core where the task is running. A triangle next to the port corresponds to the fact that the given component (in this case task) takes initiative to do an interaction and the other components connected to it are supposed to be ready

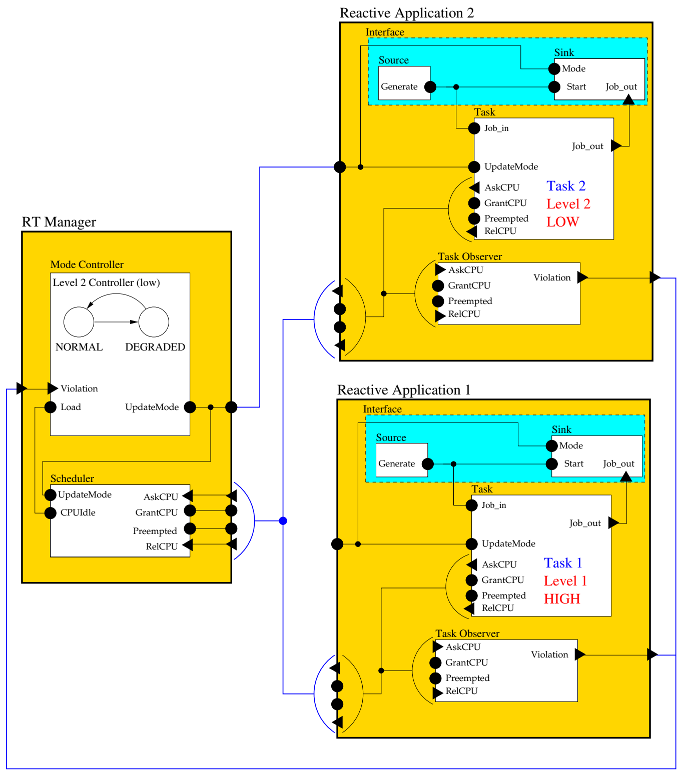

Fig. 3. BIP Model for a Mixed-Critical System

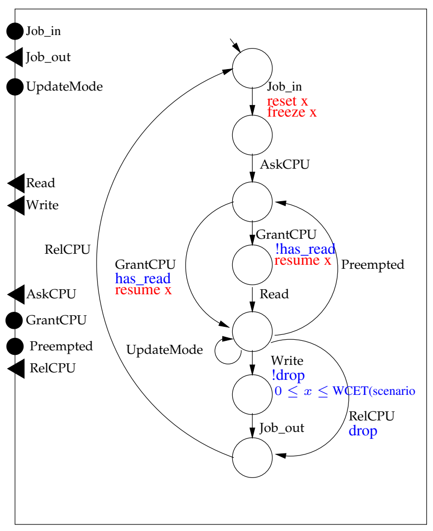

Fig. 4. BIP Task Model

to participate in this interaction at any moment when it can arrive. For example, it is the task that takes initiative to ask for a CPU. Otherwise the port is marked by a thick dot.

A task initially starts in a state where it is ready to receive a 'Job_in' immediately. The task asks the scheduler for a CPU. Having obtained the CPU, the task goes into state 'begin' and clock 'x' is reset to 0. This clock is used to model the execution cycles consumed by this task on the CPU. At any time when the given task is preempted by another task, the clock is frozen. The task asks for CPU and waits again, and when it gets the CPU back then clock 'x' is resumed. To facilitate modeling the shared resource conflicts in future work, the job execution between the 'begin' and the 'end' states follows the so-called 'superblock' model, as described in [21]. In line with this work, we split the process execution into subsequent phases that may have different access patterns to the shared resources. In the typical case described here we assume three phases. First the task reads all input data from shared to local memory (the 'Read' transition), then it executes ('Execute') and then it writes the output data from local to shared memory ('Write').

Let us now examine the 'Execute' phase. Note that only this phase is assumed in this example to consume non-negligible amount of execution cycles, to support mode changes and preemption. The number of execution cycles is bounded by function $WCET(scenario)$, which can take one of two possible values, $\mathrm{WCET}_{\mathrm{LO}}$ or $\mathrm{WCET}_{\mathrm{HI}}$, depending on the execution scenario selected during the schedulability analysis (see the next sub-section). When clock 'x' reaches the WCET value of the current scenario the task releases the CPU. It can also be forced to release CPU if the mode, set through interaction UpdateMode, implies that the task should be dropped (i.e. enforced to complete). As we see in the figure, UpdateMode changes the variable called 'drop' depending on the variable 'mode' communicated from outside this task at this interaction. This is in fact a Boolean data variable. If its value is set to true, the task is forced to finish the current job urgently in the next interaction. This is required to allow that the LO tasks can be dropped to free the CPU for the HI tasks so that the latter do not miss their deadlines. When a task is dropped, the mode of its Sink component is updated so that it does not expect a timely Job_out from the task.

The task component illustrates the basic elements of real-time BIP used to construct the BIP components. Those are the clocks (e.g. 'x'), the states (e.g. 'ini'), the interactions enabled depending on conditions (e.g. 'Release_CPU' depends on 'drop'), and the data variables (e.g. 'drop'). The other components, such as Source, Sink, Scheduler, etc., are also composed of these elements, we skip their details for space reasons.

Let us come back to the BIP implementation of our two-task scheduling example, given Figure 5. Let us explain how the mixed-critical scheduling is reflected in this BIP model. Every task is equipped with a component, called Observer, which has an own analogue of task's clock 'x' and verifies whether the clock goes beyond $\mathrm{WCET}_{\mathrm{LO}}$. According to Condition 2, in this case Task 2, which is a LO task, is not obliged to meet its deadline and in fact can be dropped. When one of the two Observers in the figure report to the Resource Manager component that $\mathrm{WCET}_{\mathrm{LO}}$ budget is violated, the Resource Manager goes from state NORMAL to DEGRADED. In this state, Resource Manager updates the mode of Task 2 such that it immediately drops.

The scheduler in this example is a fixed priority scheduler, which uses the fixed priority per tasks computed by applying so-called Audsley's approach to the mixed-critical set of tasks [20]. According to this approach, one task has a higher priority than the other and the priority assignment is done taking into account the periods, deadlines, WCETs and criticality levels of the tasks. The scheduler also monitors the current (work-)load of the processor. When the processor is idle (i.e. waiting for jobs to arrive without any task requesting for the CPU), this is reported to the Resource Manager so that it can go back to the NORMAL mode if the mode is DEGRADED. Thus, we can stop dropping the low-criticality Task 2 when the conditions permit this.

B. Schedulability Analysis

In Figure 5 we show the WCETs for one of our experiments. Here T1 is a HI task, T2 is a LO task and D denote the relative deadline.

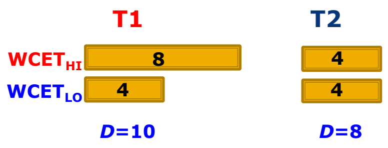

Fig. 5. WCET of our example

The tasks in this example have different deadlines[1] but, for simplicity of presentation, they have the same period and are perfectly synchronized with each other. By the beginning of each period, the CPU is idle, so the system comes back at the exactly same state as at the start. Thus, to exhaustively study all possible scenarios of this example it is enough to only consider what happens in one period. We use this fact and the verification of schedulability proposed in [2] to demonstrate verification by simulation, to show that the simulation tools (already available in the BIP framework) form an important first step towards verification with more general and realistic assumptions.

Consider two scenarios where the high-criticality task chooses to use execution time in interval $[0, \mathrm{WCET}_{\mathrm{LO}}]$ or $(\mathrm{WCET}_{\mathrm{LO}}, \mathrm{WCET}_{\mathrm{HI}}]$. The low criticality task may only choose from interval $[0, \mathrm{WCET}_{\mathrm{LO}})$ because when it exceeds this interval violation is reported and it is dropped. In every scenario one can show that the fixed-priority schedule is time robust (in the classical schedule, it is known to be such, but the mixed-critical systems deviate from classical schedules by the fact that the LO jobs may be dynamically dropped). From this it follows that to verify the time safety (i.e. the schedulability) of this example it is enough to use upper bounds of each scenario ($\mathrm{WCET}_{\mathrm{LO}}$ and $\mathrm{WCET}_{\mathrm{HI}}$) to simulate the schedule (see also Lemma 1 from [2]).

Thus, to verify the example of Figure 5 it is enough to run the BIP simulation for the duration of two periods, trying a different scenario in each period. With our BIP model, we did experiments with different parameter settings of tasks, including the cases where there is no failure or, due to an error in deployment (wrong priority assigned to tasks), a failure occurs (meaning a deadline miss of a HI task or unexpected miss of a LO task). If T2 is assigned a higher priority, it will execute first and meet its deadline 8, because it takes at most 4 units to execute. However, in the scenario where T1 runs at its $\mathrm{WCET}_{\mathrm{HI}}$ it will miss its deadline (completing at time 4+8=12), which is a failure (wrong priority assignment done by the deployment algorithms). Nevertheless, if T1 is assigned a higher priority, the worst thing that can happen is that in the scenario where T1 runs at its $\mathrm{WCET}_{\mathrm{HI}}$ it will thus violate

[1]different deadlines make this problem NP-hard

its WCET_{LO}, task T2 will be dropped (by RT Manager, as explained in the following session) and hence will not complete by the deadline. But this is acceptable degradation of a LO task that is allowed in this scheduling problem.

The scalability of our methodology is guaranteed by the theoretical results of [2], where is shown that the scheduling problem is time robust and only a polynomial number of experiments is necessary to ensure the schedulability of all the possible scenarios.

IV. Discussion and Future Work

A possible direction of future work is to generalize the 'time-robust per scenario' schedulabilty analysis from this example to more general cases. We have explained this reasoning for two tasks of equal periods and two levels of criticality, but in fact it can be generalized to more tasks with different periods and more criticality levels. In this case, one can still formulate verification by (possibly very lengthy) simulation using the current scheduling theory. However if we upgrade the tasks to have unknown relative arrival time inside each period or to be sporadic then it is not trivial. Example of mixed-critical scheduling policies for this case are EDF-VD [22] and demand-based [23]. However they provide schedulability conditions for their specific scheduling algorithm, and we are not aware of any generic schedulability verification procedure. Modeling in BIP potentially opens possibilities for using formal verification methods of time automata for such cases. Figuring out how this verification can be done in BIP is an interesting direction for future work.

Also, because the goal of our project is to study multicore system and resource conflicts, we will not always deal with time-robust systems, so more advanced methodology than simulations are required, such as compositional verification and (for lower criticality levels) statistical model checking. Extending the corresponding BIP verification methodologies to the real-time BIP is an important direction of work. Also we are interested in extending our scheduling algorithm [24] in the same way as the schedulability analysis.

In order to prove the effectiveness of our approach, we are also planning to apply the proposed methodology to model a real-life avionic application as a case study.

References

[1] L. A. Johnson, "DO-178B: Software considerations in airborne systems and equipment certification.," in *Radio Technical Commission for Aeronautics.*, RTCA, 1992.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Trans. Comput.*, vol. 61, pp. 1140 –1152, aug. 2012.

[3] S. Baruah and A. Burns, "Implementing mixed criticality systems in Ada," in *Reliable Software Technologies - Ada-Europe 2011* (A. Romanovsky and T. Vardanega, eds.), vol. 6652 of *Lecture Notes in Computer Science*, pp. 174–188, Springer Berlin Heidelberg, 2011.

[4] P. Amey, R. Chapman, and N. White, "Smart certification of mixed criticality systems," in *Proceedings of the 10th Ada-Europe international conference on Reliable Software Technologies*, Ada-Europe'05, pp. 144–155, Springer-Verlag, 2005.

[5] J. M. Faria, J. a. Martins, and J. S. Pinto, "An approach to model checking Ada programs," in *Proceedings of the 17th Ada-Europe international conference on Reliable Software Technologies*, Ada-Europe'12, pp. 105–118, Springer-Verlag, 2012.

[6] R. Alur and D. Dill, "Automata for modeling real-time systems," in *Automata, Languages and Programming* (M. Paterson, ed.), vol. 443 of *Lecture Notes in Computer Science*, pp. 322–335, Springer Berlin Heidelberg, 1990.

[7] P. Axer, M. Sebastian, and R. Ernst, "Reliability analysis for mpsocs with mixed-critical, hard real-time constraints," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '11, pp. 149–158, ACM, 2011.

[8] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 741–746, 2010.

[9] E. Lee, "Absolutely positively on time: what would it take? [embedded computing systems]," *Computer*, vol. 38, no. 7, pp. 85–87, 2005.

[10] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous dataflow programming language lustre," in *Proceedings of the IEEE*, pp. 1305–1320, 1991.

[11] A. Cohen, L. Gérard, and M. Pouzet, "Programming parallelism with futures in lustre," in *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '12, pp. 197–206, ACM, 2012.

[12] S. Baruah, "Semantics-preserving implementation of multirate mixed-criticality synchronous programs," in *Proceedings of the 20th International Conference on Real-Time and Network Systems*, RTNS '12, pp. 11–19, ACM, 2012.

[13] N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi, "Defining and translating a "safe" subset of simulink/stateflow into lustre," in *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*, pp. 259–268, ACM, 2004.

[14] H. Fuhrmann, R. von Hanxleden, J. Rennhack, and J. Koch, "Model-based system design of time-triggered architectures - avionics case study," in *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, pp. 1–12, 2006.

[15] N. Bambha, "Intermediate representations for design automation of multiprocessor DSP systems," in *In Design Automation for Embedded Systems*, pp. 307–323, Kluwer Academic Publishers, 2002.

[16] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the BIP framework," *Software, IEEE*, vol. 28, no. 3, pp. 41–48, 2011.

[17] S. Bliudze and J. Sifakis, "Causal semantics for the algebra of connectors," *Form. Methods Syst. Des.*, vol. 36, pp. 167–194, June 2010.

[18] S. Bliudze and J. Sifakis, "A notion of glue expressiveness for component-based systems," in *Proceedings of the 19th international conference on Concurrency Theory*, CONCUR '08, pp. 508–522, Springer-Verlag, 2008.

[19] T. Abdellatif, J. Combaz, and J. Sifakis, "Model-based implementation of real-time applications," in *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '10, pp. 229–238, ACM, 2010.

[20] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pp. 239–243, 2007.

[21] G. Giannopoulou, K. Lampka, N. Stoimenov, and L. Thiele, "Timed model checking with abstractions: Towards worst-case response time analysis in resource-sharing manycore systems," in *Proc. International Conference on Embedded Software (EMSOFT)*, pp. 63–72, ACM, Oct 2012.

[22] S. Baruah, V. Bonifaci, G. DAngelo, A. Marchetti-Spaccamela, S. Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms ESA 2011* (C. Demetrescu and M. Halldrsson, eds.), vol. 6942 of *Lecture Notes in Computer Science*, pp. 555–566, Springer Berlin Heidelberg, 2011.

[23] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pp. 135–144, 2012.

[24] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga, "Mixed critical earliest deadline first," in *Proc. ECRTS'13*, 2013.