On the Trade-offs between Generalization and Specialization in Real-Time Systems
(Invited Paper)

Georg von der Brüggen
Department of Computer Science
TU Dortmund University, Germany

Alan Burns
Department of Computer Science
University of York, UK

Jian-Jia Chen
Department of Computer Science
TU Dortmund University, Germany

Robert I. Davis
Department of Computer Science
University of York, UK

Jan Reineke
Saarland Informatics Campus
Saarland University, Germany

***Abstract*—While academia favours general research that is applicable to a large class of systems, this paper highlights the necessity of research into specific scenarios and aims to increase its acceptance in the real-time systems community. We argue that such research is not only motivated by greater applicability to industry, but that specialization can also provide valuable information from a purely academic perspective. In addition, the trade-offs between generalization and specialization are examined, considering not only theoretical performance, but also the impact on essential non-functional properties that are important for industry, namely composability, robustness, extensibility, and parametric simplicity.**

Preamble

The intended audience for this paper is researchers (students to seniors and practitioners to theoreticians) who are involved in the derivation and use of analysis techniques used in the off-line verification of timing constraints and hence timing guarantees for embedded real-time systems. In particular those techniques relating to scheduling and schedulability analysis. We critique the way in which research in this area often emphasizes a theory-centric approach that seeks to solve general problems in ever increasing detail, rather than considering specific problems and practical implications. As there is no coherent view of generalization in the literature, we first investigate different perspectives on generalization followed by a series of highlighted observations. These observations come from our experience as researchers, and are supported by examples. They are not laws that are universally true and hence can be proven correct, rather they identify trade-offs that can and do happen, and hence warrant careful consideration. These observations support a shift in perspective, summarized in the conclusions, that we hope readers will consider in their current and future research.

I. Introduction

When analysing a specific problem, two approaches are possible. A *practice-oriented* approach focuses on the specific problem at hand and provides a tailor-made solution. It solves the specific problem, perhaps only to the degree that is necessary in the given situation. This approach is common in industry, but is often seen as less appealing by academia. The reason being that while such solutions may be easier to develop and work well for the specific problem considered, they may be difficult or impossible to generalize. On the other hand, a *theory-oriented* approach tries to find solutions for a class of problems with similar characteristics. Such general solutions can then be applied to a class of specific problems that have something in common with the original one, i.e., they have some similar characteristics. The underlying assumption here is that if an excellent solution can be developed for the general problem, then it can be applied to all of the specific problems covered, and hence is much more valuable in the long run. Thus, real-time systems research, and for that matter any research, has a natural bias towards general models, general system implementations, and general analyses, and it is often much harder to establish results that focus on special cases. This concept effectively focuses on maximizing the size of the problem domain covered by a solution. Figure 1 depicts the problem domain and solution domain, showing how a specific solution covers a smaller part of the problem domain, but may include many practical problem instances, whereas a more general solution can cover a larger part of the problem domain, but may still not cover all problem instances of practical relevance.

Our aim in writing this perspective paper is to increase the acceptance of research into specific problems, special cases, and restricted scenarios in the real-time systems research community. Our goal is not to discourage or question the need for research into general problems and solutions, but to provide arguments for, and to increase the acceptance of, research into specialized problems and their specific solutions. The main reason for this position is that such results are more likely to be useful to industry and therefore applied in practice, since industry has a strong preference for solutions that are simple to implement and easy to understand. Further, we argue that examining special cases can also provide valuable insights from a purely theoretical perspective.

The paper provide a systematic view of the value of special cases and the possible drawbacks of placing too much emphasis on generalization (i.e. general cases) in real-time systems research[1]. The key areas we examine in this paper are:

- **Special Cases**: When are special cases useful? Why is the exploration of special cases important and valuable?
- **General Cases**: When are general cases useful? What should be considered when evaluating the quality of solutions and analyses for general cases?

We motivate, and provide evidence for, a series of **observations** that distil the key differences between a focus on special rather than

[1]This work covers a broad spectrum of topics related to real-time systems. We assume that the reader is familiar with basic concepts from the real-time systems literature, and therefore, to increase the reading flow, we do not introduce them in detail. Less common concepts and terminology related to a specific sub-problem are introduced in the appropriate section of the paper. A glossary of common real-time systems terminology can be found at https://site.ieee.org/tcrts/education/terminology-and-notation/.

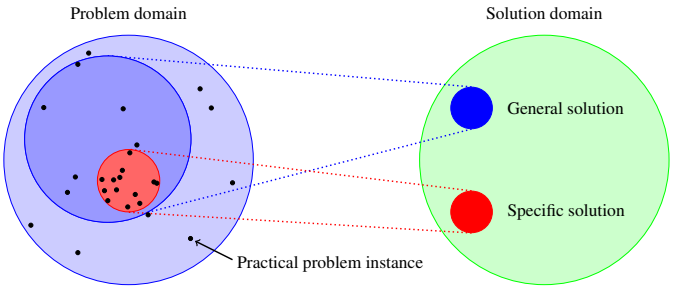

Fig. 1. Depicts the problem domain and solution domain, showing how a specific solution covers a smaller part of the problem domain, but may include many practical problem instances, whereas a more general solution can cover a larger part of the problem domain, but may still not cover all problem instances of practical relevance.

general cases.

There are different notions of generalization that are quite different in nature. Therefore, in Section II, we distinguish four classes of generalization that are discussed individually in subsequent sections. The first three of these generalizations are related to the system model, while the fourth relates to the system implementation:

1) **Abstraction refinement (Section III):** Involves describing a problem in more detail, in a way that is still able to fully describe simpler instances of the problem. For example, multiframe task systems[2] are more general than periodic task systems.
2) **Parameter relaxation (Section IV):** Involves supporting a wider range of problem instances with the same behaviour. For example, constrained-deadline task sets are more general than implicit-deadline task sets.
3) **Behaviour relaxation (Section V):** Involves supporting problem instances that allow additional behaviour. For example, sporadic tasks are more general than periodic tasks.
4) **Implementation relaxation (Section VI):** Involves the inclusion of additional solutions. For example global multiprocessor scheduling can be viewed as an implementation relaxation of partitioned scheduling that allows tasks to migrate between processors.

We examine these four classes of generalization individually, and also the topic of abstraction levels, observing advantages and disadvantages of generalization based on examples from the real-time systems literature. Their interplay and possible system-level implications are discussed afterwards in Section VII. Specifically, a further important topic in the context of generality is that, in addition to simplicity, industry also calls for solutions that fulfil other important *non-functional properties*, for example: composability, robustness, and extensibility (definitions are given in Section VII). The reason is that in industry real-time systems are developed by multiple teams and go through a number of updates both before and after deployment. Hence systems that are not easily composed, are fragile or brittle to change, or hard to extend, are seen as problematic by industry, since they incur substantially higher on-going development costs. We discuss the interplay of abstraction levels, industry's need for simplicity, and the importance of these non-functional properties.

[2]With a multiframe task, an array of values is used to represent the largest execution times of a repeating sequence of jobs of the task, that is, the execution time of the jobs repeat in a cyclic pattern given by the array. Each of these values is bounded by the worst-case execution time of the task, as used in the periodic task model.

II. Classes of Generalization

The main focus of real-time systems research is to provide timing guarantees for the functionality of a system. We examine the concept of generality in different areas of real-time systems research, considering modelling, system implementation, and analysis. This section classifies the different concepts of generality and establishes our terminology. As the perspectives of the system specification are different from those of the implementation, generalization in these two directions are different and are discussed separately. We then discuss how generalization impacts the analysis of algorithmic behaviour and the resulting solutions. There is a strong interplay between the model, the system implementation, and the analysis. In particular, the applicability of a system implementation relies on certain characteristics of the model. The analysis relies on the model as well as the system implementation. Further, often the only way to observe the impact of a modelling change or an algorithmic change is to look at how an analytical result changes. Hence, when we discuss changes that affect the model or the algorithm, we often discuss the effect as shown by the analysis.

A. Generalization in the input space

A more general model can imply greater expressiveness (abstraction refinement), less constrained parameters (parameter relaxation), or inclusion of additional behaviours (behaviour relaxation) compared to a less general model.

- **Abstraction refinement** is a generalization that enables a given problem instance to be described more accurately. This increased expressiveness provides the potential for a more accurate analysis and optimization. Model A is an **abstraction refinement** of another model B if model A is able to describe all instances described by model B, and to describe at least some of them more accurately. Conversely, model B is said to be an **abstraction approximation** of model A if it is able to describe all instances described by model A, but in some cases with less accuracy. For example, the multiframe task model is an abstraction refinement of the periodic task model, since it includes more accurate information when the runtime behaviour of a task has a cyclic pattern of execution times. Abstraction refinement and abstraction approximation are transitive (i.e., if C refines B and B refines A, then C also refines A).
- **Parameter relaxation** is a generalization that includes more instances, but with the same runtime behaviour. More specifically, model A is a **parameter relaxation** of model B if A and B have the same parameters and the parameter range of model B covers only a subset of the parameter range of model A, and models A and B describe the same runtime behaviour for the parameter range that they share. Conversely, model B is said to be a **parameter restriction** of model A. For example, the constrained-deadline task model is a parameter relaxation of the implicit-deadline task model. Parameter relaxation and parameter restriction are transitive.
- **Behaviour relaxation** is a generalization that enables additional behaviour to be modelled. More specifically, model A is a **behaviour relaxation** of model B if the runtime behaviours of the systems described by model A are strict supersets of the runtime behaviours of the corresponding systems described by model B. Note that in many cases this relaxation implies that some system behaviours described by model B are relaxed by potentially discarding structural information, which cannot be preserved in model A. Conversely, model B is said to be

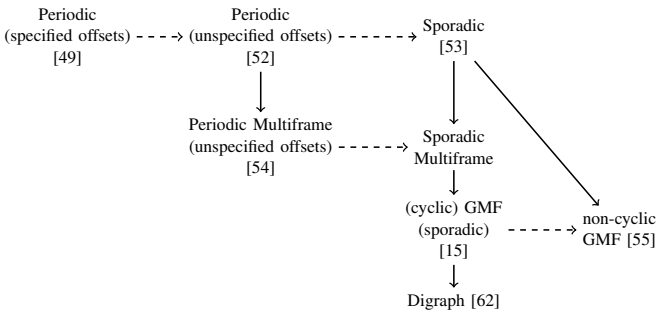

Fig. 2. Behaviour relaxations (dashed arrows) and abstraction refinements (solid arrows) between a selection of sporadic and periodic real-time task models. Non-cyclic GMF [55] is an abstraction refinement of the sporadic task model, which does not follow immediately from the other relations depicted.

a **behaviour restriction** of model A. For example, a sporadic task model can be used to model a periodic task, but it is more general, since it includes additional job release patterns that do not exist in the strictly periodic setting. Behaviour relaxation and behaviour restriction are transitive.

Figure 2 illustrates the relations between task models in terms of abstraction refinement (solid arrows) and behaviour relaxation (dashed arrows). Note, we have omitted relations that are implied by the transitivity of abstraction refinement and behaviour relaxation.

B. Generalization in the implementation space

An implementation A is an **implementation relaxation** of another implementation B if the solutions allowed by A are a strict superset of the solutions allowed by B. For example, dynamic-priority scheduling is more general than static-priority scheduling in the sense that each static-priority schedule is also a dynamic-priority schedule, but a dynamic-priority schedule is not always a valid static-priority schedule. Another example is the way in which caches can be used in a single core system, where a cache that is shared between tasks such that the sets of cache blocks that are used by the tasks may overlap is an implementation relaxation of a partitioned cache where no such overlap can happen. Note that some implementations may share similar concepts but they are unrelated to implementation relaxations. For example, global EDF and partitioned EDF both exploit EDF as the underlying implementation based on a global queue among all processors and an individual queue per processor, respectively. However, global EDF is not an implementation relaxation of partitioned EDF, since the the former always produces work-conserving schedules, whereas the latter may produce non-work-conserving schedules.

C. Historical perspectives of generalization

We note that the term *generalization* is used in the real-time systems literature in a number of different ways, some of which may seem counter to the common use of the term. In particular, consider abstraction refinement, and as an example the multiframe task model which is an abstraction refinement of the periodic task model. The multiframe task model can be used to describe, with *exactly the same degree of accuracy* and hence information content, any system described using the periodic task model. However, the multiframe task model can also be used to describe some systems (e.g. where the execution time of jobs follows a repeating cyclic pattern) with a *higher degree of accuracy* and hence more information content than the periodic task model. Thus the multiframe task model subsumes the periodic task model and is thus referred to in the literature, and in this paper, as a generalization of it. Conversely, the relationship between the periodic task model and the multiframe task model can also be viewed in terms of abstraction approximation. The precise behaviour of a multiframe task (where the execution time of jobs follows a repeating cyclic pattern) can be approximated by the periodic task model, with a lower degree of accuracy and less information, using only the worst-case execution time of the task.

One might naively consider that the periodic task model abstracts the multiframe task model and could therefore be considered a generalization of it. For example, the periodic task model can be used to describe, in an approximate way, systems where the jobs of a task exhibit execution times that are dependent in some way that is not expressible as a simple cycle, and so cannot be described via the multiframe task model with any higher degree of accuracy. However, the point here is that the multiframe task model can still be used to describe such systems with precisely the same level of accuracy and information as the periodic task model. In other words there is no increase in generality in moving to the periodic task model, rather there is just a reduction in the information provided by the multiframe task model to the trivial case of a cycle of length one and a single bound that equates to the worst-case execution time of the task.

These commonly adopted relations have been widely used in the literature of real-time systems research. Generalizations defined in this paper follow the descriptions in the literature. To avoid confusion, generalization should be precisely specified by stating its concrete perspective, i.e., with respect to expressiveness, parameters, behaviour, and implementation. In this paper, we therefore use the four classes discussed above to avoid confusion and ambiguity.

D. Analysis of models and implementation

The primary purpose of schedulability analysis is to determine whether a specific problem instance is schedulable or not. Such schedulability tests can be classified into (i) sufficient tests, which may give false negatives, but no false positives, (ii) necessary tests, which may give false positives, but no false negatives, and (iii) exact tests, which are both sufficient and necessary, and therefore give neither false positives nor false negatives.

The different types of model generalization and implementation relaxation (as well as their counterparts) result in different impacts on how sufficient, necessary, and exact tests retain or change these properties when examining task sets that were originally specified in another model. To apply a test for model A to another model B, the considered instance of B must be transferred to an instance of A before the related test is applied. For example, considering abstraction refinement/approximation, a test for the multiframe task model (i.e., the refined model in this case) can be applied to an instance of the periodic task model (i.e., the approximated model in this case) by transforming the periodic tasks into multiframe tasks with one frame each. On the other hand, if a test for the periodic task model is applied to the multiframe task model, then all the frames of a multiframe task have to be represented by a single periodic task, where the WCET of the periodic task equates to the maximum WCET of any frame of the multiframe task.

The impact of such transformations on the tests are depicted in Figure 3 with (S) denoting **sufficient** tests, (E) **exact** tests, and (N) **necessary** tests. A *solid black arrow* means that the test retains this property (e.g., a sufficient test for an abstraction refinement is also sufficient for the abstraction approximation and vice versa). A *red dashed arrow* means that the test does not retain all of its properties. Note that *red dashed arrows* only start from exact tests, meaning

that those tests lose one of their properties (i.e., they become either not necessary or not sufficient). Thus, an exact test for an abstraction approximation remains sufficient, but is not guaranteed to remain necessary. For example, if a multiframe task set is deemed schedulable by a test that is exact for the periodic task model, then it is schedulable, since the test remains sufficient. However, due to the precision lost in the transformation, an exact test for the periodic task model may deem another task set unschedulable (e.g., due to utilisation greater than 1) that is actually schedulable in the original multiframe description. A circle without any outgoing arrow implies that such a test does not always retain its properties in the other case (although there may be some exceptions where it does).

Parameter restriction is the only case where none of the tests retain their properties on relaxation. (We discuss why it is still meaningful to consider such restricted scenarios in Section IV). The reason is that, in this case, it is not clear how an instance under parameter relaxation can be transformed to the restricted case, since certain parameter combinations under parameter relaxation are simply not allowed under parameter restriction. How the transformation is achieved affects whether or not the test retains its properties. For example, consider a constrained-deadline sporadic task set where the deadlines are strictly smaller than the periods, and the task set needs to be modelled as having implicit deadlines, i.e., a parameter restriction. In this case, the transformation can be made in such a way that sufficient tests for static-priority scheduling retain their properties. This is achieved by setting the tasks' periods equal to their deadlines. The sufficiency of the schedulability tests holds in this case, since static-priority scheduling of sporadic tasks is sustainable [14] with respect to increases in task periods (i.e., the transformation back to the original constrained-deadline task set). We note that the same would not be true of periodic task sets with fixed offsets, since static-priority scheduling that utilises the values of the offsets is not sustainable with respect to increases in task periods.

III. Abstraction Refinement versus Abstraction Approximation

In this section, we discuss *abstraction refinement* (i.e., supporting more detailed information to describe behaviour) and its counterpart *abstraction approximation*. On the face of it, abstraction refinement appears preferable, as the additional information available in refined abstractions may be required to show schedulability. However, to exploit this advantage in the analysis and optimization, the information must be precise. For example, the (cyclic) generalized multiframe[3] (GMF) [15] model can only be applied once a precise cyclic pattern of job releases and execution times has been determined. Hence, additional effort is required to obtain the more detailed information and to process it.

Observation 1: *Detailed information used in abstraction refinements has a cost (in terms of time and effort) associated with obtaining it and in ensuring and maintaining its correctness (i.e. accuracy as a representation of the real system).*

A hidden cost of abstraction refinement is the sacrifice of robustness. Since the information is refined, stricter guarantees are needed. With the GMF task model, multiple minimum inter-arrival times between the different frames of each task must be ensured, along with adherence to specific execution time budgets for each frame. This may require further fine-grained inspection of the specification or system design. When we use more approximate information (e.g. the sporadic task model), compliance is simplified, since there is only one minimum inter-arrival time and one execution time budget for each task. In this respect, simpler models, with less information, are more *robust*. They can tolerate some changes in parameter values with no impact on computed results that would not be tolerated by a more refined abstraction and analysis that relies on precise values for those parameters rather than some upper bounds.

Observation 2: *Abstraction refinement may ultimately result in analysis results that are brittle rather than robust, and can be invalidated by any minor change to the system or its parameters.*

Abstraction refinement does not always yield tighter or better results. For example, if the only GMF task in a system has the lowest priority, then there will typically be no improvement in schedulability compared to using the simpler sporadic task model.

Observation 3: *Abstraction approximation may reduce analysis complexity without significantly impacting upon analysis accuracy.*

Summary

Considering abstraction approximations can be beneficial in many situations. Abstraction approximation reduces the effort and costs related to obtaining detailed information and ensuring its correctness. In some cases, detailed information may not improve the accuracy of analysis or optimization results. Abstraction approximation may also be unavoidable to reduce the complexity to an affordable level. Further, abstraction approximation can yield analysis results that are more robust to changes in information or underlying assumptions.

IV. Parameter Relaxation versus Parameter Restriction

In this section, we discuss *parameter relaxation* (i.e., supporting more problem instances, with the same runtime behaviour) and its counterpart *parameter restriction*. Here, a solution under parameter relaxation is also always a solution for restricted parameters. It is, however, still meaningful to examine solutions for the restricted setting.

Consider the case where under parameter relaxation approximated solutions are used to solve complex problems (e.g., NP-hard problems) efficiently. Such an approximation can happen within the algorithm itself (e.g., when a heuristic is used for priority assignment) or within the analysis (e.g., when a sufficient rather than exact schedulability test is used). One well-known class of sufficient schedulability tests are utilisation bounds, where the schedulability of a system is determined by evaluating whether the utilisation of the system is no greater than a specific bound. The seminal paper by Liu and Layland in 1973 [52] established that this bound is $ln(2) \approx 0.693$ for implicit-deadline task sets under preemptive rate-monotonic scheduling. If the periods of all tasks are restricted to be harmonic (i.e., for each pair of tasks the period of one is an integer multiple of the period of the other) this bound is increased to 1 [46]. In this case, the utilisation-based test becomes exact, and preemptive rate-monotonic scheduling becomes an optimal algorithm.

Observation 4: *The quality or accuracy of a solution or an analysis can become better, even optimal or exact, under parameter restriction.*

When an exact solution for a parameter relaxation is known, it may seem questionable to examine restricted cases, as the exact solution applies to them as well. However, even though there can

[3]Both the execution times and the inter-arrival times are specified by a cyclic pattern.

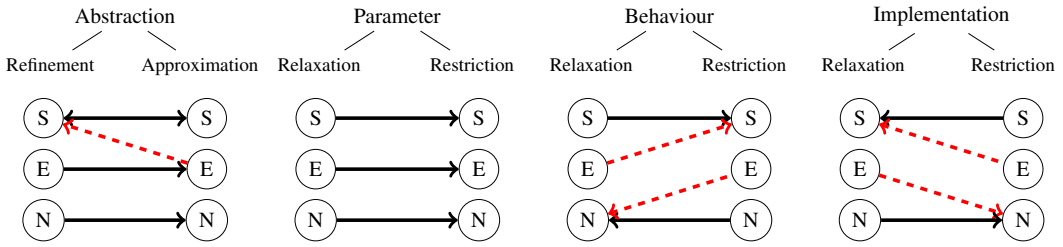

Fig. 3. Impact of the different classes of generalization on **sufficient** (S), **exact** (E), and **necessary** (N) tests, with black arrows showing where the properties of a test are retained in the other case, and red arrows showing where they are changed.

be no improvement in accuracy, investigating the scenario with parameter restriction can still be meaningful. For instance, EDF is an optimal scheduling algorithm for preemptive implicit-deadline task sets [52], but for harmonic periods rate-monotonic scheduling, which is easier to implement, is optimal as well. With respect to schedulability analysis, time demand analysis (TDA) [48] is an exact schedulability test with pseudo-polynomial time complexity for rate-monotonic scheduling of implicit-deadline tasks. However, for harmonic task sets, the utilisation-based test with linear time complexity becomes exact. We note that this behaviour is not limited to corner cases (e.g., with utilisation 1). For some semi-harmonic task sets, prevalent in automotive applications, where task periods are in the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ms [45], an exact test based on utilisation [72] is also known.

An optimal solution under parameter restriction can have properties that make it easier to apply. For example, when considering static-priority scheduling of sporadic arbitrary-deadline tasks, finding an optimal priority ordering requires the use of Audsley's algorithm [6] for each task set, considering the period, deadline, and execution time of each task. However, for constrained- and implicit-deadline task sets deadline-monotonic [49] and rate-monotonic [52] priority ordering, respectively, are optimal, regardless of the task execution times. Further, they are also always the most robust priority orderings in these cases, irrespective of the actual form that any additional interference (e.g. omitted overheads, under-estimated task execution times, etc.) may take [33].

Observation 5: *Optimal solutions under parameter restriction are often simpler and more efficient, or have preferred properties compared to optimal solutions under parameter relaxation.*

The examination of special cases is common practice when deriving theoretical results, since only one specific example is needed to establish lower bounds for utilisation bounds or speedup factors [44]. For example, considering implicit-, constrained-, and arbitrary-deadline task-sets under preemptive scheduling [52] the lower bound speedup factor[4] for static priority scheduling with optimal priority assignment, compared to an optimal dynamic scheduling algorithm is $1/ln(2)$ (approximately 1.44269), $\frac{1}{\Omega}$ (approx. 1.76322) [38] and 2 [35] respectively. Each of these results is derived from a different task set with very specific parameters. Interestingly, the upper bound on the speedup factor for the constrained-deadline case (also $\frac{1}{\Omega}$) holds under abstraction approximation, when a sufficient utilisation-based, hyperbolic test is used instead of an exact test. This results in a greatly simplified proof [28] for the upper bound with an exact test, since that can be no greater than the bound determined under abstraction approximation. Similarly, in the arbitrary-deadline case, the upper bound of 2, holds even when the priority assignment policy used is deadline monotonic, a parameter restriction that is not optimal in this case, and a simple linear time schedulability test is used, again leading to a greatly simplified proof [69] for the upper bound with an exact test and optimal priority ordering.

This relation extends to complexity results (i.e., showing that a restricted model is NP-hard directly translates to the relaxed model). Restricted scenarios can also help to pinpoint the actual cause of the complexity. For example, Ridouard et al. [60] showed in 2004 that the scheduler design problem for segmented self-suspending tasks[5] is NP-hard in the strong sense, even if each task only has two computation segments and one suspension interval. In 2019, Chen et al. [27] remarked that this already holds in the simplest setting where each task only releases one job and all computation segments have the same execution time [74], which means that the complexity results directly from the self-suspending behaviour.

Observation 6: *Negative theoretical results derived via parameter restriction directly translate to more complex scenarios via parameter relaxation, and can thus simplify proofs.*

Summary

Considering scenarios under parameter restriction can be beneficial in multiple situations. Good solutions under parameter restriction can be much simpler, more efficient, and have preferred properties. Such solutions can be more robust to small changes and more easily extended, both of which are qualities that are highly appreciated by industry. Further, examining parameter restrictions enables negative results, such as complexity results, utilisation bounds, and speedup factors that provide fundamental properties of the studied problem to be extended to more complex scenarios via parameter relaxation.

V. Behaviour Relaxation versus Behaviour Restriction

Considering a model that permits additional behaviour (i.e., a behaviour relaxation) sounds promising, as it directly enables additional systems to be covered. However, since a behaviour relaxation usually results in losing some information (compared to behaviour restriction) this generality often leads to an imprecise analysis for the restricted scenario.

We first examine the downside that, when including additional behaviour, the scenario with restricted behaviour may be modelled less accurately (i.e., information is discarded), which can result in

[4]The maximum speedup factor $\rho^{A \to B}$ between two scheduling algorithms A and B is the minimum increase in speed that is necessary to guarantee that every task set that is schedulable with algorithm B can also be scheduled with algorithm A.

[5]In the segmented self-suspension model, the execution behaviour of a task is specified by an interleaving array of m execution intervals with related WCET and $m-1$ suspension intervals with related maximum suspension time.

a less accurate analysis. For example, consider three tasks with parameters (WCET, deadline, period, offset): periodic task τ_1 (0.5, 0.5, 2, 0), periodic task τ_3 (0.5, 1.5, 2, 0.5), and sporadic task τ_2 (1, 2, 2, -). If they are scheduled with static-priority scheduling according to the task indexes (i.e., τ_1 has the highest priority and τ_3 the lowest), then an exact analysis for sporadic systems deems the task set unschedulable as it assumes all tasks could be released at time 0 and in this scenario task τ_3 would miss its deadline. However, when considering the offset of the periodic tasks, the task set is in fact correctly determined schedulable under the given priority order.

In some situations, only highly relaxed or highly restricted models are available and the trade-offs between these models have to be examined. Consider tasks with self-suspension behaviour, where the majority of research uses one of two models [29], [30]. The segmented self-suspension model describes tasks via one specific interleaving execution-suspension pattern, which all jobs of the task must comply with. This enables an accurate analysis, but only covers a very specific behaviour (i.e., it is often too restricted). By contrast, the dynamic self-suspension model is very flexible but imprecise. The dynamic self-suspension model, compared to the sporadic task model, only assumes an upper bound on the total suspension time as additional information, and jobs are permitted to alternate between execution and suspension as many times as they like provided that the given bounds on total WCET and suspension time are not violated. This model can therefore cater for any task with suspension behaviour, but its flexibility results in a very pessimistic analysis if the tasks behaviour can be described more precisely [71] (i.e., it is arguably too relaxed). Models with different trade-offs provide a potential solution to the "Goldilocks" [32] issue of too much relaxation or too much restriction. For example, hybrid self-suspension models [71] provide improvements in schedulability compared to the dynamic self-suspension model if additional information about the task behaviour can be obtained. Knowing the number of suspension intervals and upper bounds on the execution times of the individual segments can increase the analysis accuracy considerably. Accuracy can be further improved if all possible execution-suspension patterns are known offline or even online [71]. Again there is the trade-off between analysis accuracy and the amount of information that needs to be provided.

Observation 7: *Behaviour relaxation may ultimately result in the description of the behaviour becoming too imprecise, leading to pessimism or inaccuracy in the analysis. By contrast, the additional information considered through behaviour restriction can make the analysis more complex, but also has the potential to improve its accuracy. It may however also result in not all of the required problems being covered.*

This example shows that separating different kinds of generalization is not always easy and that there is some overlap. On the one hand, hybrid self-suspension is a behaviour restriction of the dynamic self-suspension model and a behaviour relaxation of the segmented self-suspension model. On the other hand, starting from the segmented self-suspension model, the hybrid self-suspension model can also be seen as an abstraction refinement: the hybrid model describes the system with a set of execution/suspension patterns, while the segmented model is an abstraction approximation that (assuming the number of suspension intervals is constant) upper-bounds all execution segments and suspension intervals with the maximum execution time/suspension time value over all these patterns for the related segment/interval.

Summary

Behaviour relaxations cover systems with additional, often more flexible behaviour. However, this may come at a price of reduced analysis accuracy, since behaviour relaxation discards information that was available in the restricted case. It can thus be beneficial to choose the right level of behaviour restriction to obtain the desired trade-off between generality of the model and analysis accuracy.

VI. Generalization in Implementation Space

This section discusses *generalization of system* ***implementation*** (i.e., the inclusion of additional possible solutions for a given problem). In this sense, global scheduling is more general than partitioned scheduling on a multiprocessor, and dynamic-priority scheduling is more general than static-priority scheduling. By this we mean that global scheduling, which is able to support task migration, can be tailored to replicate any form of partitioned scheduling, but the converse is not true. Similarly, dynamic-priority scheduling can be tailored to replicate any form of static-priority scheduling, but the converse is not true.[6]

One might assume that the inexorable rise in demand for ever more complex functionality implemented in software and the associated increases in processor performance required would result in a high demand for solutions that enable the capacity of processors, networks and other components of real-time systems to be fully utilised. However, a recent survey [1], [2] examining industrial practice in the field of real-time systems, found that this is often not the case. Instead, simple and well understood solutions and analyses are typically preferred.

Further, applying a more general system implementation may be prevented by constraints and limitations. These can be the result of external requirements (e.g., that a certain COTS hardware platform must be used), compliance to certain standards (e.g., AUTOSAR [8]) is required, or restrictions of the middleware used (e.g., many commercial RTOS support static-priority scheduling, but support for EDF is less common).

Observation 8: *In real-world situations applying a more general solution can be prevented by constraints and limitations on the system.*

Even if there are no such restrictions and concerns, providing a more general system implementation will usually also have some downside that needs to be considered. A more general system implementation is often more complex to implement and to analyse. Therefore, investing this effort is only sensible when the expected gains are high enough, which is not always the case. For example, global scheduling is theoretically able to achieve the highest processor utilisation. However, a study by Brandenburg and Gul [24] showed that for implicit-deadline task sets, semi-partitioned approaches are able to schedule task sets with very nearly 100% processor utilisation, demonstrating that the potential for improvement under global scheduling is very limited in practice.

A choice of solution that appears to be beneficial in theory can also be sub-optimal in practice due to increased system overheads. One well-known example is the comparison between preemptive and non-preemptive scheduling. While preemptive scheduling in theory

[6] Note, this does not mean that any global scheduling algorithm is superior to any partitioned scheduling algorithm, for example global EDF does not dominate partitioned fixed priority scheduling. Similarly, it does not mean that any dynamic-priority scheduling algorithm is superior to any static-priority scheduling algorithm, for example, non-preemptive EDF does not dominate non-preemptive fixed priority scheduling.

increases system schedulablity, due to the fact that it assigns the processor to an arriving high-priority job immediately, the cache-related preemption delays resulting from context switches can negate this effect, since they may increase the WCET of tasks by up to 40% [56].

Another example is multiprocessor scheduling, where global scheduling in theory enables 100% utilisation of the processors, while partitioned scheduling leaves empty capacity due to the underlying bin-packing problem. However, since partitioned scheduling allows the effective use of caches, avoids migration overheads, and reduces RTOS overheads as well as access contention compared to global scheduling, it has been shown that in practice global scheduling is not preferable to partitioned scheduling as the number of cores increases [16].

Observation 9: *The gain of implementation relaxation (i.e., a more general system implementation) can be minimal or non-existent when a restricted implementation is already nearly optimal. Further, the resulting overheads of the relaxed implementation may completely negate any theoretical gain.*

When the additional possibilities resulting from an implementation relaxation can be exploited without any significant drawback, these possibilities can often result in higher performance. For example, consider dynamic-priority and static-priority preemptive scheduling of sporadic tasks in a single processor system. Here, an optimal dynamic-priority scheduling algorithm such as EDF strictly dominates static-priority scheduling assuming optimal priority assignment: Every task set that is schedulable using a static-priority scheduling algorithm is also schedulable using EDF; however, the converse is not true. This dominance is reflected in the empirical performance of the corresponding exact schedulability tests.

When the additional analytical complexity (e.g., due to additional interference between system components that now has to be considered) results in a less precise analysis, then this effect may counter the potential gain of an implementation relaxation. Consider, for example, global and partitioned multiprocessor scheduling. Some exact schedulability tests for global static-priority [63], [65] and global EDF [12], [19], [39] scheduling are known, but their scalability is such that they cannot be applied to anything more than toy examples (from an industrial perspective). When sufficient schedulability tests are used, the current state-of-the-art analyses for global scheduling, both for static-priority [31], [40], [41], [63] and EDF [11], [13], [64], stem from the seminal work by Baker [10], which leads to an interference upper bound with a multiplicative factor of $1/m$ in the resulting tests, where m is the number of processors. These analyses are, however, not able to show any gain compared to partitioned scheduling, even if task migration costs are ignored. On the contrary, it has been shown for global static-priority scheduling [66] as well as for global EDF and global FIFO scheduling [17] that the sufficient analysis for global scheduling is dominated by partitioned scheduling (i.e., all task sets that are deemed schedulable under global scheduling according to these tests are also schedulable under partitioned scheduling when a certain partitioning algorithm is used). Hence, a fundamentally different analysis technique is needed to exploit the potentially higher utilisation of global scheduling compared to partitioned scheduling.

Observation 10: *The analysis for the general case of an implementation relaxation can be too imprecise to exploit the theoretical gain, due to high problem complexity and/or the complexity of the analysis itself.*

While special cases often have simpler structures that can significantly simplify the studied problem, these simplifications may sacrifice optimality or analysis accuracy. We use two concrete examples to discuss these potential pitfalls.

Federated scheduling was proposed by Li et al. [51] to handle *implicit-deadline* sporadic real-time directed acyclic graph (DAG) tasks on multiprocessor platforms. However, "*in terms of the speedup metric with respect to any optimal scheduling algorithm, federated scheduling strategies do not yield any constant speedup factors for constrained-deadline task systems with DAG structures*" [26]. Although federated scheduling has been adopted as a simplified scheduling paradigm, an optimal federated schedule may have unbounded performance loss in comparison to the actual optimal solution.

Considering resource sharing on multiprocessors, Brandenburg and Anderson [22] showed that $\Omega(m)$ *priority-inversion blocking* (pi-blocking) is unavoidable under suspension-oblivious schedulability analysis for multiprocessor locking protocols [22], where m is the number of processors. The underlying intuition is that a task may in the worst case be blocked by one task on each processor. Several protocols are known that are asymptotically optimal for minimizing the pi-blocking when using FIFO-waiting queues (e.g., FMLP [18], FMLP$^+$ [21], OMLP [23], and DFLP [20]). However, the actual performance bottleneck is the way in which the tasks are partitioned (respectively, the way task instances are assigned to the processors under global scheduling), as this can potentially avoid the worst case of $\Omega(m)$ pi-blocking. Therefore, algorithms that consider partitioning strategies and resource sharing protocols simultaneously have the potential to reduce the blocking time and as a result improve schedulability. *Resource-oriented partitioned scheduling* [42], [70] focuses on the shared resources instead of the processor workload (i.e., it partitions the shared resources first) and can result in a huge gain in schedulability [42], [70].

Observation 11: *The gain of an optimal algorithm under an implementation restriction can be negated by the fact that the restricted implementation is far from optimal, or because the performance bottleneck results from a different aspect of the problem.*

Summary

The gain of implementation relaxation (i.e., a more general system implementation) can be minimal or non-existent when a restricted implementation is already (nearly) optimal. Further, the resulting overheads of the relaxed implementation may completely negate any theoretical gain. The analysis for the general case of an implementation relaxation can be too imprecise to exploit the theoretical gain, due to high problem complexity and/or the complexity of the analysis itself.

VII. Abstraction Levels and Industry's Need for Simplicity

Industry has a strong preference for solutions that are both simple to implement and easy to understand. Industry practitioners, responsible for designing, developing, and verifying real-time systems, often follow the KISS ("Keep It Simple Stupid") principle, originally coined by Kelly Johnson [59], lead engineer at Lockheed Skunk Works, responsible for the U-2 and SR-71 aircraft. This maxim follows from a quote attributed to Albert Einstein[7] "make everything as simple as possible, but not simpler". In practice, real-time systems are developed by multiple teams of people, and typically go through

[7] https://quoteinvestigator.com/2011/05/13/einstein-simple/#more-2363

a number of revisions and updates before and after deployment. This life cycle of continuous development lends increased importance to a set of non-functional properties (criteria) that apply both to the system itself and to its analysis and verification:

- **Composability:** A system and its analysis are said to be *composable*, if changes to a component can be made, within some predefined limits (e.g., an execution time budget), without requiring changes to other components or impacting upon their analysis results.
- **Robustness:** The concept of *robustness* implies that small perturbations, either changes or errors in the values of parameters, or online behaviour, or extra interference or overheads, should not change the timing correctness of the system or the validity of the analysis results. Hence, a robust system and analysis will ensure that, as far as possible, a system remains schedulable when subject to small perturbations, while a fragile system and analysis will not.
- **Extensibility:** A system is extensible if new components can be added, without the need to make modifications to the existing system configuration and components. Appropriate analysis methods can be useful as part of a design approach that optimizes extensibility.
- **Parametric Simplicity:** There is a cost to collecting the information required by an analysis method, and to ensuring that it is correct (i.e., precise values or upper / lower bounds as appropriate). This cost may need to be paid each time a system is modified.

There is typically a trade-off between the abstraction level used in analysis, the accuracy of the analysis results, and the four criteria stated above. A high level of abstraction that has good parametric simplicity, may be composable, extensible, and robust, but the analysis of it may result in classifying some systems as unschedulable when in fact they are schedulable. This could lead to changes having to be made to the system in order to render it schedulable according to the simple analysis. On the other hand, a lower (i.e., more detailed) level of abstraction and a more complex analysis may sacrifice parametric simplicity, robustness, extensibility, and composability, in an effort to avoid revisions to a system that is in fact already schedulable. It is our contention that industry has a strong preference for keeping things simple, and that this principle extends to the choice of analysis methods, with simple analyses preferred, even if the downside of that trade-off is some wasted system capacity. Simple analyses are easier to understand and to support, and can enhance composability, extensibility, and robustness. In the following three subsections we provide examples and observations that support this view.

We note that different industry sectors, companies, and product developments may value these four non-functional properties differently, and thus choose to make different trade-offs between those properties and the accuracy of the analysis techniques employed.

A. Controller Area Network (CAN)

Controller Area Network (CAN) is widely used in automotive applications. The CAN protocol is an excellent example of design, based on fixed priority non-preemptive scheduling, that has the *potential* to support composability, robustness, extensibility, and parametric simplicity. However, whether or not these properties hold for a particular schedulability analysis for CAN depends on the level of abstraction used, as the following discussion illustrates.

The first correct schedulability analysis of CAN [36] provided exact and sufficient schedulability tests, assuming a sporadic model of message releases. Later work extended the analysis to periodic transactions (groups of messages with offsets) again providing both exact and sufficient schedulability tests [73]. More recent works have investigated robust [34], [37] and extensible [57] priority and message ID assignments for CAN.

It is interesting to consider the simplest form of sufficient analysis for CAN, discussed in [37]. This analysis effectively makes three simplifying assumptions: (**A1**) message releases are sporadic, (**A2**) every message is considered as having the maximum permitted length, (**A3**) there are soft real-time messages of the maximum length at low priorities, hence the blocking factor always takes a fixed maximum value. These assumptions can impact the accuracy of the analysis; however, for many automotive systems, the pessimism caused by (**A2**) and (**A3**) is minimal or non-existent. Typically, the majority of messages are of the maximum length, since CAN has a large message overhead compared to payload, and thus it is efficient to pack data into maximum length (8 data byte) messages. Further, there are mandatory diagnostic messages that are of the maximum length and are assigned background (low) priorities. These simplifying assumptions are specific to the analysis of CAN, and would not apply to fixed priority non-preemptive systems in general. There are many positive aspects to using such a specific and simplified analysis:

- Composability: Changes can be made to the content of the messages increasing the number of data bytes up to the maximum without any changes in the analysed worst-case response times or network schedulability. Further, the sender of a message can be changed[8], provided that the same basic timing parameters are respected (i.e., message periods and release jitter), without any change to schedulability. Note, this is not the case when the more sophisticated periodic transaction model and its analysis are used [73], since the offset between messages sent by different nodes is subject to clock drift.
- Robustness: Deadline-minus-Jitter monotonic priority ordering has been proven optimal [75], [37] with respect to the simple analysis. In this case, it is also optimally robust [37] with respect to message delays due to re-transmissions caused by errors on the CAN bus.
- Extensibility: As shown in [57], the simple analysis facilitates the use of a priority ordering and an initial message ID assignment that maximizes extensibility in terms of future upgrades, i.e., the addition of further messages without having to change the IDs of the existing messages or the software on the other nodes.
- Parametric Simplicity: The simple analysis does not need to know the length of any of the CAN messages, or their sending nodes, only their basic timing parameters. Subsystem developers are therefore free to change these aspects without compromising network schedulability.

There is of course a trade-off here, with many recent CAN systems developed using more detailed analysis, for example taking into account different message lengths; however, this unavoidably sacrifices flexibility (the above properties) in exchange for more precise analysis of the available bandwidth (schedulability). In our view, practitioners may well benefit from being mindful of the properties that are being sacrificed in the quest for improved schedulability, and thus only make choices that compromise those properties when it is necessary to do so, rather than by default.

It is interesting to note that early commercial use of flawed analysis for CAN (see [36] for a discussion) fortuitously avoided issues

[8] As would happen if a task processing data obtained via the CAN bus and outputting its results back on to the bus were moved from one node to another.

in deployed systems because simplifying assumptions were used, namely (**A3**). This meant that the flawed analysis did not manifest in incorrect schedulability results. Here, the KISS principle delivered a welcome additional benefit.

Observation 12: *Taking account of the specific details of an application domain can lead to a simplified analysis that makes a trade-off in analysis accuracy (pessimism) that is acceptable to industry, while meeting other important criteria, such as composability, robustness, extensibility, and parametric simplicity.*

B. Static-priority co-operative scheduling

Arguably two of the greatest advances in real-time systems research during the 1990s were the development of response time analysis as a practical engineering approach [43], [7] to analysing single core systems that use static-priority scheduling, and the coincidental development of effective resource locking protocols, e.g., the Stack Resource Protocol [9]. The standard response time analysis formulation for static-priority preemptive scheduling is reproduced below. C_i and T_i are the execution time budget and minimum inter-arrival time or period of task τ_i. R_i is its computed response time. If $R_i \leq D_i$ then the task is schedulable, where D_i is the task's constrained deadline ($D_i \leq T_i$).

$$R_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{1}$$

Assuming that the Stack Resource Protocol [9] is used, then the blocking factor B_i is given by the longest time for which any task of lower priority than task τ_i holds a resource that is also accessed by a task of the same as or higher priority than task τ_i. To compute the blocking factor B_i, it is therefore necessary to know: (i) which tasks access each resource, and (ii) the maximum length of time that each task holds each of the resources that it accesses. The need for this information places a burden on system developers to provide it and to ensure that it is correct. It also damages composability and extensibility, since adding a new resource access to a task can impact the schedulability of other tasks, even if the task still complies with its execution time budget. Similarly, adding a task at the lowest priority that locks a resource that is shared with the highest priority task potentially impacts the schedulability of every task in the system.

An alternative approach is to set $B_i = C^s$, where C^s is a budget on the maximum length of any critical section in any task, where either preemption is disabled or a resource is locked. The formulation then becomes a sufficient test for static-priority co-operative scheduling, where C^s is an upper bound on the time that any task may execute non-preemptively before it yields a scheduling point. The use of such a critical section budget decouples the analysis from a reliance on information about other tasks, save for their execution time budgets and periods, thus supporting composability and extensibility. Further, this formulation belongs to a category of analyses where Deadline Monotonic priority assignment has been proven to provide optimal robustness [33] with respect to any reasonable[9] additional interference function. Examples of such additional interference functions include scheduler, RTOS, and interrupt handler overheads, and underestimation of the execution times of other tasks.

This alternative formulation corresponds to what is now common practice in the automotive industry, where a small number of tasks with nearly harmonic periods are used as containers for hundreds of "runnables" [45]. The runnables execute sequentially within their task, and preemption is only permitted at explicit preemption points between them. (This structure was designed to make the development of large numbers of runnables manageable and to simplify the run-time environment / operating system.) Provided that the total execution time of the runnables in a task does not exceed the task's execution time budget, and no runnable executes for longer than the critical section budget, then the analysis holds. Thus the system and its analysis are both composable and extensible. Further, the co-operative scheduling behaviour enables all shared resource accesses, which can amount to thousands of accesses to global variables (referred to as "labels") [45], to take place without the need for, or overhead of, explicit resource locking.

Observation 13: *Co-design of analysis and runtime behaviour specific to an application domain can produce solutions that are more efficient, have simpler analysis, and support industry-relevant criteria, including composability, robustness, extensibility, and parametric simplicity.*

C. Cache-related preemption delays

As a final example, we consider a single core system with a shared cache. With static-priority preemptive scheduling, a preempting task can evict cache lines that a preempted task was using, and hence the analysis needs to account for Cache-Related Preemption Delays (CRPD). Schedulability analysis accounting for CRPD in static-priority preemptive scheduling [4] can be achieved at various levels of abstraction, using more or less information. In the following, ECB refers to the Evicting Cache Blocks of a preempting task and UCB to the Useful Cache Blocks of a preempted task.

- Coarse approximation: Assumes that on each preemption, the whole cache is evicted. In this case, only information about the hardware (i.e., the size of the cache, block reload time etc.) is required, no information is needed about the ECBs or UCBs of each task. A composable analysis is thus possible, since the software for the tasks may be changed, respecting their execution time budgets, without impacting analysed response times including the impact of CRPD on schedulability.
- ECB-Only approximation [25], [68]: Assumes that on each preemption, all the ECBs of the preempting task are evicting useful cache blocks of preempted tasks that will subsequently need to be reloaded. In this analysis, the additional cost can be added to the execution time of the preempting task, and amounts to the size of the set of ECBs that the task uses multiplied by the block reload time. This approximation is less coarse than the one above, but requires more information. In this respect, the size of the set of ECBs for a task is relatively easy to obtain, as compared to UCBs, at least considering instruction caches. Further, if each task is given an ECB set size budget, then the analysis can still be composable, since it supports changes to the software of other tasks that respects their execution time budgets and ECB set size budgets. Experimental evaluation of CRPD analysis techniques [61] shows that ECB-Only comes close in accuracy to significantly more complex and brittle approaches, such as Multi-set approaches discussed below.
- UCB-Only approximation [47]: Assumes that on each preemption by a task τ_j, the maximum size UCB of all the tasks that may be preempted by τ_j and have higher or equal priority to task τ_i (the task under analysis) are evicted. This analysis needs only the size of the UCBs of the tasks; however, this information is typically more difficult to obtain than that for ECBs. Again

[9]Reasonable meaning that the maximum additional interference is no smaller in a longer time interval than it is in a shorter one.

the analysis could potentially be composable if tasks were given UCB set size budgets; however, compliance with these budgets would be much more difficult to achieve in practice than with ECB set size budgets.

- ECB-Union [3], UCB-Union [67], and Multi-set approaches [4]: These provide the most precise analysis, but require detailed information about the UCB and ECB sets for all tasks. As these approaches use the intersection of the sets of UCBs and ECBs, rather than just their sizes, these analyses are not composable. Changing a task's code, even staying within budgets for the size of its UCB and ECB sets could impact the schedulability of other tasks, since the size of the UCB and ECB set intersections may change.
- Further improvements to the analysis of static-priority preemptive systems with CRPD can be achieved by considering cache lines that are able to persist in the cache between the execution of jobs of the same task, without being evicted by interleaving execution. This analysis focuses on Cache Persistence Reload Overheads (CPRO) [58], and dominates equivalent analyses that only consider CRPD.

The above example illustrates how different levels of abstraction and corresponding analyses are possible when considering the same problem.

Observation 14: *Deeper levels of abstraction support more complex analyses that can theoretically provide more accurate results; however, they also require more information to be collected, which is typically more costly to obtain, and the use of which is often detrimental to other desirable properties, such as composability, extensibility, and robustness.*

There is an age-old philosophical argument, attributed to Voltaire, that "the perfect should not be the enemy of the good". Stated otherwise, achieving a perfect solution may be impossible and since increasing effort results in diminishing returns, further activity beyond a certain point becomes increasingly inefficient. Thus solutions with a theoretical advantage when complex analyses are used (e.g., non-partitioned caches versus cache partitioning [5]) may see this advantage eroded or negated when simpler analyses are employed.

Observation 15: *The levels of abstraction and complexity of analyses that are viable for industry to use may be such that the "good enough" solutions that provide the best performance in practice take a quite different form from those that provide the best performance in theory.*

D. Summary

In this section, we have considered the different levels of abstraction that may be used in modelling a system and hence in its analysis. Here, we see that there is typically a trade-off, with deeper abstraction levels enabling a more accurate but also more complex analysis, at the expense of non-functional properties such as composability, extensibility, robustness, and parametric simplicity. While it is appropriate for researchers in the real-time systems community to work towards improved solutions and analyses that can provide the most precise results, it is also important to consider other significant factors that impact adoption by industry. In many cases, careful consideration of the details of a specific problem domain can enable a simple solution and corresponding analysis approach that in practice trades-off little in terms of accuracy, with substantial improvements to other important non-functional properties. In our opinion, such work is more likely to have an impact on industry practice.

VIII. Conclusions

In 1966, Levins [50] discussed the philosophy of modelling, arguing that there is an inevitable and unavoidable trade-off between three properties of a model: *precision*, *realism*, and *generality*. Levins argues that no useful model can exist that maximises precision (accuracy of results with respect to the model), realism (accuracy of replicating what is being modelled), and generality (applicability to multiple problems). For a model to be useful it must be *tractable*, thus introducing a forth dimension into this inevitable compromise.

In this paper, we have discussed different ways in which real-time systems research seeks to progress towards more general and realistic models (in Levins' terminology) and their accompanying analyses. This is achieved via *parameter relaxation*, *abstraction refinement*, *behaviour relaxation*, and *implementation relaxation*, discussed in Sections III, IV, V, and VI respectively, while specific problems with their constraints, restrictions, and limitations are given less attention. However, by focusing on the general rather than the specific, the system models and analyses developed inevitably become more complex and less tractable. Crucially, continual progress in the direction of high fidelity general models and analyses inevitably results in severe compromises to other non-functional properties, such as *composability*, *robustness*, *extensibility*, and *parametric simplicity*. These properties are arguably of greater value to industry than eking out the last percentage point of system capacity through ever more detailed modelling and analysis, as discussed in Section VII.

In conclusion, we argue that while real-time systems researchers should still investigate improvements and refinements to models and analyses in the direction(s) of generality, there are significant benefits to be had from allocating more effort to exploring specific problem instances that are relevant to industry, and also by looking at other important fundamental properties of analyses beyond their accuracy and complexity. Considering specific industry relevant problems, there are often constraints, restrictions, and limitations, that can be brought to bear to simplify the models and analyses used. The result being viable techniques that give little away in terms of accuracy and capability, yet provide strong support for key non-functional properties that are important to industry. Indeed, co-design of analysis and restrictions on the implementation focusing on a specific problem may combine to bring both a simpler analysis and efficiency gains.

We would like to encourage real-time systems researchers to think carefully about how their research can actually be applied. There may be some simplifications or constraints that emanate from specific practical examples of a problem that can make the difference between industry viewing the resulting analysis as an interesting intellectual curiosity that they will never use or as a viable method where they can see the potential for significant impact.

Finally, our advice to practitioners involved in developing real-time systems using tools that implement schedulability analysis or other timing verification techniques is as follows. To consider carefully the level and detail of information required by those tools and the implications that has on non-functional properties, such as *composability*, *robustness*, *extensibility*, and *parametric simplicity*. Choosing simpler analysis configurations and techniques may be preferable. Or where that is not possible, treating parameters as budgets (with headroom to accommodate change) rather than precise values, may bring substantial benefits.

Acknowledgement: The research in this paper is partially funded by the EPSRC grant STRATA (EP/N023641/1) and the Innovate UK HICLASS project (113213). EPSRC Research Data Management: No new primary data was created during this study. This result is part of three projects that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 803111, grant agreement No. 865170, and grant agreement No. 101020415).

REFERENCES

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. An empirical survey-based study into industry practice in real-time systems. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 3–11. IEEE, 2020.

[2] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Syst.*, page 41, 2021.

[3] S. Altmeyer, R. I. Davis, and C. Maiza. Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 261–271. IEEE Computer Society, 2011.

[4] S. Altmeyer, R. I. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real Time Syst.*, 48(5):499–526, 2012.

[5] S. Altmeyer, R. Douma, W. Lunniss, and R. I. Davis. On the effectiveness of cache partitioning in hard real-time systems. *Real Time Syst.*, 52(5):598–643, 2016.

[6] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, 1991.

[7] N. C. Audsley, A. Burns, M. M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Softw. Eng. J.*, 8(5):284–292, 1993.

[8] AUTOSAR. AUTOSAR standards. https://www.autosar.org/standards/, 2021.

[9] T. P. Baker. A stack-based resource allocation policy for realtime processes. In *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*, pages 191–200. IEEE Computer Society, 1990.

[10] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003), 3-5 December 2003, Cancun, Mexico*, pages 120–129. IEEE Computer Society, 2003.

[11] T. P. Baker and S. K. Baruah. An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. *Real Time Syst.*, 43(1):3–24, 2009.

[12] T. P. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*, pages 62–75, 2007.

[13] S. K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *21st Euromicro Conference on Real-Time Systems, ECRTS 2009, Dublin, Ireland, July 1-3, 2009*, pages 259–268. IEEE Computer Society, 2009.

[14] S. K. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS 2006), 5-8 December 2006, Rio de Janeiro, Brazil*, pages 159–168. IEEE Computer Society, 2006.

[15] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok. Generalized multiframe tasks. *Real Time Syst.*, 17(1):5–22, 1999.

[16] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*, pages 14–24. IEEE Computer Society, 2010.

[17] A. Biondi and Y. Sun. On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling. *Real Time Syst.*, 54(3):515–536, 2018.

[18] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson. A flexible real-time locking protocol for multiprocessors. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), 21-24 August 2007, Daegu, Korea*, pages 47–56. IEEE Computer Society, 2007.

[19] V. Bonifaci and A. Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63(4):763–780, 2012.

[20] B. B. Brandenburg. Blocking optimality in distributed real-time locking protocols. *Leibniz Trans. Embed. Syst.*, 1(2):01:1–01:22, 2014.

[21] B. B. Brandenburg. The FMLP+: an asymptotically optimal real-time locking protocol for suspension-aware analysis. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 61–71. IEEE Computer Society, 2014.

[22] B. B. Brandenburg and J. H. Anderson. Optimality results for multiprocessor real-time locking. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*, pages 49–60. IEEE Computer Society, 2010.

[23] B. B. Brandenburg and J. H. Anderson. The OMLP family of optimal multiprocessor real-time locking protocols. *Des. Autom. Embed. Syst.*, 17(2):277–342, 2013.

[24] B. B. Brandenburg and M. Gul. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*, pages 99–110. IEEE Computer Society, 2016.

[25] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. J. Gil, and A. J. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *2nd IEEE Real-Time Technology and Applications Symposium, RTAS '96, Boston, MA, USA, June 10-12, 1996*, pages 204–212. IEEE Computer Society, 1996.

[26] J. Chen. Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems. *Real Time Syst.*, 52(6):833–838, 2016.

[27] J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen. Scheduling self-suspending tasks: New and old results. In *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany*, volume 133 of *LIPIcs*, pages 16:1–16:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[28] J. Chen, W. Huang, and C. Liu. k2u: A general framework from k-point effective schedulability analysis to utilization-based tests. In *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*, pages 107–118. IEEE Computer Society, 2015.

[29] J. Chen, G. Nelissen, W. Huang, M. Yang, B. B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. C. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019.

[30] J. Chen, G. von der Brüggen, W. Huang, and C. Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*, pages 1–10. IEEE Computer Society, 2017.

[31] J. Chen, G. von der Brüggen, and N. Ueter. Push forward: Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. In *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*, volume 106 of *LIPIcs*, pages 8:1–8:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[32] J. Cundall. *Treasury of Pleasure Books for Young Children*. Grant and Griffith successors to Newbery and Harris, 1850.

[33] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 3–14. IEEE Computer Society, 2007.

[34] R. I. Davis and A. Burns. Robust priority assignment for messages on controller area network (CAN). *Real Time Syst.*, 41(2):152–180, 2009.

[35] R. I. Davis, A. Burns, S. Baruah, T. Rothvoß, L. George, and O. Gettings. Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms. *Real Time Syst.*, 51(5):566–601, 2015.

[36] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real Time Syst.*, 35(3):239–272, 2007.

[37] R. I. Davis, A. Burns, V. Pollex, and F. Slomka. On priority assignment for controller area network when some message identifiers are fixed. In *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, pages 279–288. ACM, 2015.

[38] R. I. Davis, T. Rothvoß, S. K. Baruah, and A. Burns. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real Time Syst.*, 43(3):211–258, 2009.

[39] G. Geeraerts, J. Goossens, and M. Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 49(2):171–218, 2013.

[40] N. Guan, M. Han, C. Gu, Q. Deng, and W. Yi. Bounding carry-in interference to improve fixed-priority global multiprocessor scheduling analysis. In *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015, Hong Kong, China, August 19-21, 2015*, pages 11–20. IEEE Computer Society, 2015.

[41] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 387–397. IEEE Computer Society, 2009.

[42] W. Huang, M. Yang, and J. Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*, pages 111–122, 2016.

[43] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[44] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[45] S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmarks for free. In *6th Internat. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[46] T. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the Real-Time Systems Symposium - 1991, San Antonio, Texas, USA, December 1991*, pages 160–170. IEEE Computer Society, 1991.

[47] C. Lee, J. Hahn, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), December 4-6, 1996, Washington, DC, USA*, pages 264–274. IEEE Computer Society, 1996.

[48] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989, Santa Monica, California, USA, December 1989*, pages 166–171. IEEE Computer Society, 1989.

[49] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[50] R. Levins. The strategy of model building in population biology. *American Scientist*, 54(4):421–431, 1966.

[51] J. Li, J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 85–96. IEEE Computer Society, 2014.

[52] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[53] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[54] A. K. Mok and D. Chen. A multiframe model for real-time tasks. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), December 4-6, 1996, Washington, DC, USA*, pages 22–29. IEEE Computer Society, 1996.

[55] N. T. Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *22nd Euromicro Conference on Real-Time Systems, ECRTS 2010, Brussels, Belgium, July 6-9, 2010*, pages 271–278. IEEE Computer Society, 2010.

[56] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of CPU and I/O transactions in cots-based embedded systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS 2008, Barcelona, Spain, 30 November - 3 December 2008*, pages 221–231. IEEE Computer Society, 2008.

[57] F. Pölzlbauer, R. I. Davis, and I. Bate. A practical message ID assignment policy for controller area network that maximizes extensibility. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, 2016.

[58] S. A. Rashid, G. Nelissen, S. Altmeyer, R. I. Davis, and E. Tovar. Integrated analysis of cache related preemption delays and cache persistence reload overheads. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 188–198. IEEE Computer Society, 2017.

[59] B. R. Rich. Clarence leonard (kelly) johnson 19101990: A biographical memoir. National Academies Press, 1995.

[60] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004), 5-8 December 2004, Lisbon, Portugal*, pages 47–56. IEEE Computer Society, 2004.

[61] D. Shah, S. Hahn, and J. Reineke. Experimental evaluation of cache-related preemption delay aware timing analysis. In *18th International Workshop on Worst-Case Execution Time Analysis, WCET 2018, July 3, 2018, Barcelona, Spain*, volume 63 of *OASICS*, pages 7:1–7:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[62] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 71–80. IEEE Computer Society, 2011.

[63] Y. Sun and G. Lipari. A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, 2014.

[64] Y. Sun and G. Lipari. Response time analysis with limited carry-in for global earliest deadline first scheduling. In *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*, pages 130–140. IEEE Computer Society, 2015.

[65] Y. Sun and G. Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*, 52(3):323–355, 2016.

[66] Y. Sun and M. D. Natale. Assessing the pessimism of current multicore global fixed-priority schedulability analysis. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 575–583. ACM, 2018.

[67] Y. Tan and V. J. M. III. Timing analysis for preemptive multitasking real-time systems with caches. *ACM Trans. Embed. Comput. Syst.*, 6(1):7, 2007.

[68] H. Tomiyama and N. D. Dutt. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *Proceedings of the Eighth International Workshop on Hardware/Software Codesign, CODES 2000, San Diego, California, USA, 2000*, pages 67–71. ACM, 2000.

[69] G. von der Brüggen, J. Chen, R. I. Davis, and W. Huang. Exact speedup factors for linear-time schedulability tests for fixed-priority preemptive and non-preemptive scheduling. *Inf. Process. Lett.*, 117:1–5, 2017.

[70] G. von der Brüggen, J. Chen, W. Huang, and M. Yang. Release enforcement in resource-oriented partitioned scheduling for multiprocessor systems. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, pages 287–296. ACM, 2017.

[71] G. von der Brüggen, W. Huang, and J. Chen. Hybrid self-suspension models in real-time embedded systems. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*, pages 1–9. IEEE Computer Society, 2017.

[72] G. von der Brüggen, N. Ueter, J. Chen, and M. Freier. Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, pages 108–117. ACM, 2017.

[73] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis. Controller area network (CAN): response time analysis with offsets. In *9th IEEE International Workshop on Factory Communication Systems, WFCS 2012, Lemgo, Germany, May 21-24, 2012*, pages 43–52. IEEE, 2012.

[74] W. Yu, H. Hoogeveen, and J. K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. *J. Sched.*, 7(5):333–348, 2004.

[75] A. Zuhily and A. Burns. Optimal (d-j)-monotonic priority assignment. *Inf. Process. Lett.*, 103(6):247–250, 2007.