Model Classification

Oded Maler

VERIMAG

May 20, 2006

AMETIST DELIVERABLE 1.1

Project acronym: AMETIST
Project full title: Advanced Methods for Timed Systems
Project no.: IST-2001-35304
Project Co-ordinator: Frits Vaandrager
Project Start Date: 1 April 02
Duration: 36 months
Project home page: `http://ametist.cs.utwente.nl/`

1 Introduction

This deliverable is intended to classify scheduling problems in a way that will facilitate the task of identifying the, hopefully non empty, niche for timed automata technology in the scheduling landscape. Such a classification is not an easy task, especially for *real* scheduling problems, because, to rephrase Tolstoy, each scheduling problem is unhappy in its own way, that is, problems do not classify simply according to a small number of orthogonal decision variables. Moreover, since scheduling problems appear almost everywhere, it is not realistic to expect an exhaustive classification encompassing the theory and practice of so many domains of human activity, given the meager allocation of person months to this task. Nevertheless, in what follows we propose some useful features that partition the space of scheduling problems. These properties are of different characters, some refer to the meta-level (academic versus real), some to the application domain (manufacturing versus computing) and some refer to specific technical features of mathematical models (certainty versus uncertainty). As mentioned above, these variables are not independent and, in fact, are often correlated. Naturally, the classification will be of higher resolution when it comes to areas covered by the AMETIST project, and which are more relevant to timed automata. The presentation will be organized in the following format: we will discuss each classification feature and describe what are the implications of of being in either side of the spectrum that it spans. In the case that the value of the variable in question has some strong implications with respect to TA technology, we will comment on that, as well as on the dependency between variables (for example: in application domain X, scheduling problems tend to have technical feature Y). At the final section we will try to summarize.

Before embarking on this journey, let us clarify what do we mean by a *model*. To start with, we have some "reality" which consists of physical entities such as production lines in a factory, orders, deadlines, workers and machines. A model of this part of reality is a set of formal objects (words, mathematical symbols, diagrams, computer programs) whose formal inter relationships somehow reflect the important aspects of the concrete situation. Durations, precedence and resource constraints are examples of important aspects, while the maiden name of the mother of the shift managers is not. A good model provides for some kind of "virtual" analysis, using pencil and paper or computers, whose formal outcome corresponds to the concrete situation. For instance, if the formal analysis results in a solution in the form of a Gantt chart satisfying certain constraints, we would like this chart to be realizable in the real world with a proper translation of the "events" in the chart to actual initiation and termination of production steps at specific points in (real) time.

In the above discussion we talked about *instances* of situations (say, planning of operations for factory X in week Y) and their corresponding instances of models. More generally we would like to speak of *classes* of models, or model *schemes* (in the data-base sense). In the concrete world these classes correspond to similar situations, for example planning for the same factory at different weeks. Each instance will be different but will typically employ the same type of machines, products and constraints. A more general class of models may speak of situations which are common to all factories in a given sector. In the abstract world such model classes correspond to types of problems in the mathematical sense, for example, systems of constraints restricted to some form (e.g. linear), or classes of timed automata having certain features. It is mostly at this level of model classes that this deliverable is situated.

2 Real vs. Academic Models

One major distinction is between models coming from the theoretical/academic world and models intended to represent actual problems coming from practice. The purpose and use of these two types of models do not always coincide and their common use of the word *model* is sometimes a source for misunderstanding. These purposes are summarized below:

- *Theoretical models*: the goal is to capture some of the essential and generic features of a class of concrete situations, in order to be able to say something rather general about such situations. The more mathematically inclined and established is the community involved, the more will the models tend to be stylized and ignore non generic details. The outcome of theoretical models can be theorems, for example on the properties of optimal schedules or uniqueness of solutions, classification in terms of computational complexity, or efficient solution algorithms. Such results, especially when they are analytic, are very hard to obtain in the presence of non generic details. While this approach is very productive for the advancement of science[1] there is a risk that a theoretician will focus on those details that allow him or her to prove theorems and will tend to neglect and classify as non generic those that are not of much use for the paper industry. A typical example is the concentration of the operation research (OR) community on deterministic offline problems, partly because these models are amenable to formulation as decision or optimization problems whose complexity can be established. A good example of a useful abstraction of this type can be found in program verification where abstract computational models of transition systems were used, rather than models based on specific programming languages.[2]
- *Practical models*: these models should capture all relevant details for the actual operation of a concrete plant, here and now. Facts like national holidays, trade unions, relative importance of customers and many other details that may be of no interest to the theoretician, can be critical for correct or optimal operations. Sometimes these details, like many others in daily life, go without saying and escape formalization. However, if we like to mechanize the scheduling process as much as possible, we need to formalize many of them.[3] The formalization of less and less structured information in a form amenable to automatic deduction and computation is a known problem in AI, knowledge engineering and natural-language processing.

Perhaps the differences between these types of models are best manifested in while comparing the classical *job-shop* problem [JM99] and the Lacquer Production case study provided by Axxom. The job-shop problem is a crystallized formulation of one of the fundamental features of scheduling, namely the interaction between *precedence* and *resource* constraints. The beauty of the problem lies in its simplicity, which is sufficient for understanding why it is harder than (convex) linear programming due to the disjunctive form of the resource constrains, to prove its NP-hardness, to illustrate simple heuristic solutions such as greedy execution, give bounds on the distance between optimal and heuristic solutions, etc. On the other hand, the problem is very simplified and does not cover many realistic features. For example it assumes that every step consumes exactly one resource/machine, a fact which already does not allow the treatment of containers. It assumes that each step can be performed only on one type of machine. It uses the "maximal tardiness" (makespan) as an optimization criterion

[1]Newton's toy problem of point masses involved a lot of simplifications compared to real planets.

[2]Of course, the connection with the concrete reality of software production should later be re-established in order to make tools based on such abstractions usable in practice. However the insights gained while studying the abstract models were less likely to be obtained if one had to deal with real programs.

[3]It would be hard to prove nice theorems about such models, though.

while it is evident that in real life the termination times of *all* jobs do matter, for better (delivery) or worse (inventory) and that some deadline constraints are harder than others. Each of these and other restrictions can be relaxed, rendering the mathematical model more baroque and ugly.

The Axxom case-study, represents the opposite side of the spectrum. It has containers allocated to each job during its lifetime, it has constraints on the waiting time between steps, it has job-dependent conditional activities such as rinsing, it has deadlines and a non-trivial cost function and has to take into account the non-regular structure of the Julian calendar and its holidays.[4] So a priori, such a practical problem seems to be much harder than the abstract job-shop problem.[5] On the other hand, this problem is “solved” every day (or week, or month) by humans and tools of various sorts, so where is the catch? This can be explained by the fact that if one does not insist on global optimality (which is never really the issue) and applies some common-sense ideas such as non-laziness[6] or non overtaking (which is very plausible hypothesis for pipeline-like jobs) to reduce the solution space, one can easily find reasonable solutions.

The moral of this section is that the practical solution of real-world problems does not necessarily involve the use of the theoretical and algorithmic state-of-the-art in the domain. The best available abstract scheduling algorithm is not necessary nor always sufficient to deliver real viable solutions. However, we strongly believe that if one wants to scale up and attack problems whose solution admits a serious computational bottleneck, focusing on the essential features that cause the combinatorial explosion is a better strategy in the long run than attacking immediately real detailed problems.

Where do timed automata stand here? As has been demonstrated in the project they can model in an elegant way clean theoretical problems such as the job-shop and task-graph problems. To go beyond that, they pay the performance price that any other methodology that allows richer constraint would pay. Adding relative deadlines and more complex temporal constraints one might lose the optimality results specific to the job-shop problem such as non-laziness. Adding more sophisticated cost functions, as has been demonstrated in the work on *linearly-priced timed automata* [BFK$^+$01], we are still in the domain where the strong theoretical results for timed automata hold, although the performance may deteriorate. Features needed to capture more complex situations can be incorporated into timed automata, losing perhaps the decidability feature which, anyway, does not seem to be pertinent for synthesis and scheduling problems. So, along the theory to practice axis, timed automata do not seem to be inferior nor superior to alternative formulations of the scheduling problem. As will be indicated in the sequel, a major disadvantage of timed automata is manifested in problems where non-temporal resource constraints are dominant, while their potential advantage lies in the effective modeling of uncertainty and more intuitive representation of the system dynamics.

3 Application Domains

Due to the universality of the scheduling and resource allocation problem, it appears everywhere and no attempt is made to be exhaustive. We will talk of three large classes of application domains which, needless to say, are far from being internally homogeneous.

[4]Even the trivial but tedious translation of time between decimal numbers to hours/minutes format is an issue for applicability and usability.

[5]Some explanation can be given in terms of the difference between the more syntactic notion of *descriptive complexity* and the inherent *internal complexity* of problems which need not coincide.

[6]For job-shop problem this is not a heuristic but an exact optimization.

3.1 Manufacturing

We use this generic term to denote “traditional” industries that transform physical entities (raw materials) into other entities (products) using well-known processes (recipes). The whole field known as *operations research* originated from the need to organize the production process in an economically competitive manner. Many of these decisions are design decisions for constructing the plant. Other decisions are related to machine renovation, marketing and inventory policies which are not directly related to scheduling. Scheduling problem manifest themselves typically in the following form: given the factory as is, with its production capabilities and constraints, with inputs of orders and material, find an optimal or reasonable way to orchestrate the production so as to produce the output in a timely and economic manner, while meeting the deadlines associated with the orders.

The scheduling problem is meaningful when there are *bounded resources* that can be used for the production of *several* products or product instances. Although these resources are re-usable, they can be used for one purpose at a time and conflicts occur when we have to decide at which order to allocate the resource to competing tasks. Different solutions correspond to different ways of resolving these conflicts, and the cases which are meaningful from an economic point of view occur when these solutions differ significantly among themselves in quality (cost, meeting deadlines). If this is not the case, choosing an arbitrary schedule would do. In a more modern formulation, the scheduling problem is referred to as the *value chain management problem*, where more holistic considerations including transportation between different production sites (or subcontractors), inventory costs, or financing are integrated into the decision process.

We mention briefly some features that seem to be common to problems coming from this domain and their corresponding models:

1. In many (but not all) of manufacturing domains, the time scale of the operations is much larger than that of computation. Typical production steps may take hours and days and the process of planning a weekly schedule can afford many hours of computation on powerful machines.[7]

2. The cost of individual decisions can be very significant, especially when one deals with large-scale operations such as refineries and other chemical plants.

3. Most such operations involve humans and physical processes whose exact performance cannot be predicted exactly. Moreover, the problem specification may change during execution due to machine breakdown, order cancellation and more. Hence the schedules chosen should be *robust* under a reasonable amount of disturbances, and cannot be based on some rare and complex opportunities that are possible only in a short temporal window. Moreover, it seems that the best approach to these problems is not fully-automatic but more in the spirit of an interactive decision-support system where the user may force some decisions, and let the computer do some exhaustive computations only for some parts of the problem.

3.2 Transportation

In transportation problems, the work to be done is given in terms of quantities of goods and persons to be transferred between geographic locations, using the transportation infrastructure which includes vehicles, routes and junctions. Train and airline schedules are fairly regular and periodic and are also

[7] Here too, one can find exceptions: when a machine in a production line breaks down, a quick solution should be found without rescheduling the whole production. However, in this case we are talking about different granularity.

subject to safety constraints. However, more often than not, there are unexpected delays that call for rescheduling, which is not always lead to good results, as many of us might have experienced.

Tracks, junctions, air-corridors and landings are the shared resources for which decision should be made. We do not elaborate further on this application domain because it was not studied extensively within AMETIST .

3.3 Computing and Communication

In computing, the goods to be transformed and transported are pieces of information realized by low-energy electronics. Shared resources are computation devices such as processors or lower-level functional units which transform data from one form to another, communication channels that transport these data between different production and consumption sites and shared peripherals such as printers and other I/O devices. We make a quick review, roughly in chronological/historical order, of some of the most common manifestations of scheduling problems in this application domain.

Time-Sharing Operating Systems In time sharing, a CPU serves a variety of users, some interactive, working in front of a terminal, and some others issuing jobs that have to run in the background. The scheduler is part of the operating system and its role is to give the interactive users the illusion that the CPU works for them (they are sufficiently slow not to notice the difference), and execute the other programs as well. In addition, slow resources such as printers should also be allocated as well.[8] Since the pattern of demand of these computational resources is not known in advance, there is no use, to plan for a specific optimal schedule. The problems was resolved by assigning *priorities* to tasks according to their importance, with obvious preference to system tasks such as interrupt handlers.

Real-Time Systems The work in this domain is motivated by the realization of (parallel) control loops by sequential computers. These are typically *periodic* tasks, each of which has to be performed in a given frequency, and each has a known bound on its duration on the given execution platform. From the scheduler's point of view, the frequency requirements can be expressed in terms of *release times* and *deadlines*. Liu and Layland in their seminal paper [LL73] studied this problem and showed some analytic bounds on schedulability of such a set of computations on a given processor. Two basic scheduling policies are commonly used. The first is *earliest-deadline first* (EDF), which means always to execute the task whose deadline is the closest (recall that a single machine is used here). The EDF strategy is both common sense and optimal. The other strategy known as *rate monotonic* (RM), always gives priority to enabled tasks with the highest frequency of execution. While this fixed priority policy is not optimal, it is easy to implement by methods borrowed from time sharing. It should be noted that both policies use preemption. The weakness of the model lies in the assumption that tasks are considered as independent, that is, there are no precedence constraints.[9] Other simplifying assumption is that tasks do not occupy resources other than the CPU and that the overhead of context switching due to preemption is negligible. Various extensions to these policies were suggested in order to treat more realistic situations. One of those has even led to a famous and costly bug in the Mars Rover.

[8]The allocation of more than one resource to a job gave rise to the first deadlocks. Proving the absence of deadlocks in resource-allocation protocols became subsequently one of the first cases studied in program verification.

[9]Except, of course, precedence between subsequent instances of the same task.

Scheduling Parallel Programs Parallelism in computing is a fashion that periodically becomes popular in certain quarters. The idea is to execute programs, even if written using a sequential programming language, on a network of (typically identical) processors. Such programs can be decomposed into blocks that constitute the basic tasks. Precedence constraints among those tasks are deduced from *data-flow* analysis of the program, where the execution of each block should be preceded by the execution of those pieces of code that produce the data it uses as input. This precedence graph can be seen as both a generalization of the job shop problem (a richer form of precedence) and a restriction of it (machines are identical, and the problem is symmetric unless communication cost is taken into account). Specific classes of programs, especially those used in scientific computing, such as matrix operations, admit particular parallelization schemes. The whole topic is currently undergoing a renaissance due to the decision of semi-conductor industry to move to the so-called *multi-core* architecture with several processors on a chip.

Instruction Level Parallelism At a smaller scale, scheduling can be performed also at the level of the *micro architecture*, where a processor has a number of functional units for performing arithmetic and logical operations and it may try to parallelize some executions that are data independent. Moreover, in certain cases, the amount of parallelism in the hardware architecture is sufficient for engaging in *speculative* executions, where precedence constraints are not fully respected. That is, if a task has only few outcomes, it is possible to start executing in parallel several instances of a successor task with each of the possible outcomes. When the first task terminates and the outcome becomes known, those task instances that do not correspond to the outcome are aborted. While, in principle, this problem is very similar to scheduling on parallel machines, there is a quantitative difference in the time scale and in the computational resources available when decisions are to be taken, and these differences may influence to choice of solution techniques.

Network Scheduling Communication channels at various levels of granularity constitute a particular resource in the computational infrastructure with an ever increasing importance. Some examples of scheduling problems include the allocation of time slots in a field bus in distributed control systems, the allocation of switches in routing networks or the allocation of frequencies in wireless networks. Some of these problems have a “bulk” character, the number of individual “tasks” is very large and models where individual tasks are represented explicitly are of no practical use. Instead, as in the case of scheduling policies for time-sharing operating systems, statistical models based on queuing theory are more suitable.

Server Scheduling Another example of a class of massive resource allocation problems is encountered in the operation of large distributed data-base systems (via the web or another communication infrastructure) where a “farm” of servers processes different type of queries coming from all over the world. Here, as in the routing problem mentioned in the previous subsection, the modeling style is more close to queuing theory.

What are the major characteristics of scheduling problems in the computation domain, compared to problems in manufacturing?

1. Time scale is a major difference with respect to most physical processes. Production steps in computation will typically have a very short duration[10] and one can apply traditional optimal and nearly-optimal solution techniques only for very restricted decision horizons.

[10]Of course one may find exception, for example if we want to schedule on a cluster or over the Internet a heavy sci-

2. The economic significance of better schedules is of a somewhat different character. In many cases better resource allocation policies may improve the tradeoff between the cost of the hardware infrastructure and the quality of service. The problem is that, unlike physical production, improving the throughput of a computational system by buying few more processors is not such a big expense given the price decline in computer technology. Hence such a reduction might be significant only for companies whose servers answer queries on planetary scale such as banks and search engines.

   Another niche of the information market where optimal scheduling may be meaningful is *embedded computing*. If a more efficient scheduling policy can reduce, even by a fraction, the computational resources in a car, or in a cell phone, multiplying these savings by the number of products sold may lead to economic significance. And indeed, optimization the cost of embedded hardware, either by more efficient circuit synthesis or by more efficient scheduling is a very hot topic, sometimes referred to as *hardware-software Co-design*. However, as we discuss below, such embedded systems are also limited in the computational resources they can invest in the additional task of solving the scheduling problem itself, if the latter is to be solved online. Offline solutions may be more feasible but restricted to dedicated applications.

3. Finally, unlike real manufacturing, operations in the computation domain are often performed in a completely automated fashion. This makes them, in certain cases, much more time predictable than operations that involves matter and humans[11] and since the entity that executes the schedule is also a computer, sophisticated and non-intuitive schedules which are hard to implement in a human environment can be realized in this context.

Timed automata have been demonstrated to be capable of modeling phenomena in both manufacturing and computation. In general they seem to share with constraint optimization methods the infeasibility of solving large problem in real time, where one has to resort to simple solutions such as priority-based scheduling that do not even attempt to approximate the optimum.

4 Time Scales and Computational Resources

One important characteristic feature of classes of scheduling problems is the relationship between the following factors:

1. The length (in absolute time) of the decision/execution horizon[12] for which a solution is sought;

2. The time scale of the process to be scheduled, that is, the duration of a typical execution step. This factor determines the size of the scheduling problem: if the decision horizon and the activity density are kept fixed, the shorter is the average duration, the more tasks have to be scheduled in the same temporal window;

3. The computational complexity of finding optimal or satisfactory schedules for the class of problems in question. Note that problems that have the same number of tasks may differ radically in complexity;

entific computation which would take years on a single computer, we may have time-scales that approach those of typical manufacturing.

[11]Worst-case execution time (WCET) of modern processors with cache is a notable exception.

[12]This issue, in fact, is not really separable from the question of uncertainty to be discusses next.

4. The available computing power, measured by the time it takes to perform a basic step in the scheduling algorithm (exploring a path in the case of timed automaton representation, finding a satisfying assignment in a constraint-based approach).

To illustrate the significance of these factors consider the following two concrete situations taken from the above mentioned application domains.

Example 1: Suppose we have to plan the *weekly* schedule of a factory. The decision horizon is in the order of a week, say 100 hours. Assuming that the average duration of execution steps is 5 hours, and the factory has m machines, the scheduling problem involves about $20m$ tasks.[13] Assuming that planning takes place in the weekend and that all orders for the next week are known at that time, the computing effort that can be dedicated to the problem is around 50 hours multiplied by the power of the available computers. This may be sufficient for finding reasonable (if not optimal) results for such a problem. Typically, computers are cheap compared to other machines and resources, so if optimal decisions have important economic impact, additional resources can be allocated to the computation to improve the quality of the solution (although NP remains NP).

Example 2: Suppose we want to schedule a set of instructions on the parallel micro architecture of a processor. If we deal with straight line programs without branching, we know in advance the set of instructions to be executed. Hence we can perform the optimization at *compile time*, which gives us sufficient computation time.[14] The situation is radically different when we deal with branching programs where the stream of instructions to execute is not known in advance (like orders to a factory for long time horizons). In this case the scheduler has to work *online*, taking as input the list of instructions accumulated in its execution stack at a given moment. If we deal with n instructions, we have to solve a scheduling problem for n tasks within a time bound which is of the same order of magnitude as the execution of n instructions, something which is totally infeasible unless we use an auxiliary computer which is orders of magnitude faster than the processor in question. But in this context, the optimization should be performed using the resources of the *very same processor* whose operation we want to optimize!

One can see that in the scheduling of fast processes, the importance of the *timeliness* aspect of the solution to the scheduling problem dominates the *quality* of the solution.[15] In such situations, simple and fast solutions seem to be the only viable alternative. As stated earlier, an interesting niche would be embedded systems (rather than general-purpose processors) who run a fixed (branching) program, which might change infrequently, only when the system is upgraded. For such programs, investment in doing the optimization in compile time may pay off.

Timed automata are rather neutral with respect to this issue. Computing shortest paths in timed automata within a very short time seems to be as infeasible as doing it using other techniques based on constraints. Quick heuristic solutions can be implemented using the timed automaton representation, although the advantages of using this representation still needs to be demonstrated.

5 Determinism versus Uncertainty

An issue that kept on surfacing in the previous discussion, namely *determinism* versus *uncertainty*, is very important and is not so much well-studied in the traditional scheduling literature, mostly due

[13]These are gross estimations, of course, the task durations nay vary, machines may be different, etc.

[14]Provided, of course, that the object code is refined to contain low-level directives to the micro architecture.

[15]Similar issues have been studied under the title of *anytime computation*.

to the lack of conceptual tools for expressing such phenomena.[16] In traditional optimization one has to choose between alternatives, and each choice leads to a *unique outcome* which is obtained by substituting the choices in the constraints and cost function. This is exactly the situation in the classical job-shop problem, where the jobs to be executed are fixed as well as their duration on the given machines. In this setting, a sequence of scheduling decisions *completely determines* the actual execution and its cost. In reality, scheduling problems are corrupted with a large amount of uncertainty, a fact that led some authors to comment: *"The static problem definition is so far removed from job-shop reality that perhaps a different name for the research should be considered"* [MSB98]. As we feel that the treatment of uncertainty could be a major advantage of timed automata, we will elaborate on this point.

One can classify the uncertainty concerning the operations of a production plant into two major types:

1. *Internal uncertainty*: All unexpected events that may change the production capacity and performance of the plant, including machine breakdown or slowdown, strikes, wrong estimations of the time to perform a step, or critical workers that need to go to the dentist;

2. *External uncertainty*: Changes in the problem specification, such as arrival of new urgent orders, cancellation of existing ones, changes in the cost function due to market fluctuations.

Like any classification scheme, this one is not complete. In particular, the fact that a task took more or less time than expected can be seen as something internal (the plant executed differently from the expectations) or external (the task that arrived was different from the task envisioned) or as a combination of both.

One distinctive feature of timed automata is the *set-theoretic*, rather than *stochastic*, manner in which they define uncertainty. Instead of defining the duration of a task as a probability distribution, it will be modeled as an *interval* of possible durations. Although in many cases computations over sets tend to be more tractable than computations over probabilities[17], one has to ask what modeling style is more adequate for capturing the reality of scheduling problems. To examine this point we will make the following additional distinction between uncertainty types:

1. *Closed Uncertainty*: The task specification for the decision horizon is more or less stable in terms of the identity of tasks and their arrival times. What is uncertain is the exact duration or arrival time of the tasks, and perhaps some finitely many conditional variations on the task specification which are determined and revealed during execution.

2. *Open Uncertainty*: Task arrival is dynamic during execution and is underspecified. There might be constraints on the inter-arrival time between tasks of various types but not more than that.

There have been some attempts to apply timed automata technology to certain variants of open uncertainty, for example [FY04] can prove some properties of real-time scheduling problems in the presence of both periodic and *sporadic* tasks, with bounds on the inter-arrival time of the latter. However it is our feeling that for more open-ended problem specifications, the whole scheduling methodology, timed automata or not, is not the most adequate one, and models in the spirit of queueing theory are more suitable. This is not to say that queueing theory, which was initially conceived to treat simple

[16]The be fair, problems where the uncertainty is modeled probabilistically can sometimes be formulated as simple optimization problems.

[17]Unless one uses exponential distributions which are not always faithful to the modeled phenomena.

"atoms" without interactions such as precedence, is fully-equipped to treat the whole range of production systems with open uncertainty, but it looks as if the theory of timed automata in its current form is not.

So let us restrict our attention to closed uncertainty and see what a scheduling algorithm can do in its presence. We can designate one of all possible "realizations" of the problem (choices of the uncertain parameters) as a "nominal" one, the one which is likely to occur under normal conditions. With respect to this realization the problem is deterministic. We can then treat the uncertain problem in either one of these ways:

- Ignore the uncertainty altogether and provide a solution for the nominal deterministic model. Then implement the schedule in a *robust* manner with additional slack time between tasks, backup for machines in case of breakdown, etc. When such robustness measures have been taken, the original schedule can be gracefully deformed when a deviation from the nominal model occurs.
- Solve the nominal deterministic problem, and then re-schedule the residual problem (the remaining tasks) each time a deviation is observed. This solution is applicable, of course, only to slow systems.
- Compute a scheduling strategy *offline* using control synthesis algorithms for timed automata. Such a strategy covers all the contingencies that may occur in the actual execution, but the reaction need not be computed by rescheduling the whole problem online, but rather by a lookup table. This was the approach taken in [AAM06]. In the case of temporal uncertainty in task durations, the performance of the strategy thus obtained was very close to that of a clairvoyant scheduler.

Although one may argue that the latter approach will not scale up, and that the representation of the scheduling strategy can become too space consuming for certain applications, we believe that this approach, can eventually lead to interesting solutions for problem admitting closed uncertainty. The advantage of timed automata for posing such problems is in the natural way alternating fixpoint computations can be formulated and solved, while a constraint-based approach will need to formulate the problem using a long alternation of quantifiers (or $\min$ and $\max$ operations). Even if the ultimate working solution to the problem of dynamic scheduling will eventually use simpler techniques, the semantic insights gained by looking at the scheduling problem as a dynamic system will remain valuable.

6 Temporal versus Non-Temporal Constraints

Large parts of the optimization that takes place in operations research is not at all of a temporal nature. For example problems like bin packing, or optimal placement of production sites to reduce transportation costs are simply combinatorial optimization problems that do not benefit at all from their encoding using automata. Some problems in scheduling mix both temporal and non-temporal reasoning: for example, if we have to produce large volumes of different materials, each quantity being much beyond the capacities of a single machine or container, then, in addition to resolving conflicts between the jobs, one has to choose among the infinitely-many ways to partition the bulk quantities into smaller ones.

Although one can think of modeling such situations with timed automata augmented with auxiliary real-valued variables which are assigned non-deterministically, it seems that such problems are not the best advertisement for the use of automaton-based models in scheduling.

7 Constraints versus Optimization

Scheduling is a constrained optimization problem having two ingredients, the *constraints* and the *cost function*. In the job-shop problem, the cost function is trivial and most of the burden is to find the best out of the (exponentially many) ways to satisfy the constraints.[18] Part of the difficulty of the problem comes from the fact that apart from the precedence and resource constraints, the problem is not sufficiently constrained, and tasks can, in principle, be executed any time. On the other hand, when *release times* and *deadlines* are associated with jobs and tasks, the decision horizon can be partitioned into segments, in each of which only a small subset of the tasks can be executed.[19] These additional constraints can make the problem easier to solve satisfactorily because among the fewer solutions that remain, the variability in the cost would be typically smaller.

How are methods based on timed automata influenced by the level of constrainedness? A priori, adding more constraints, reduces the number of runs in the automaton that correspond to feasible schedules, which is a positive feature for a search-based algorithm. On the other hand it is a well-known fact about constraints satisfaction that the hardest problems are those that are close to the feasibility/infeasibility boundary. Current methods for finding schedules using timed automata work in a chronological manner (forward or backward in time) and for such hard problems, the infeasibility is likely to be discovered very deep inside the search tree. Perhaps some ideas such as “learning” in satisfiability problems can be helpful in this context.

8 Conclusions

This report presented insights and improved understanding of scheduling problems gained through the AMETIST project. It can be viewed as another step in the long journey from theory to practice and back. The AMETIST case studies provided the academic partners with very valuable feedback about the applicability and limitations of their techniques and tools, and a better understanding of the real-world niches where timed automata technology, or a future technology partly inspired by it, can find use. Perhaps a new class of models that somehow combines the advantage of time automata in expressing complex interacting objects and of more statistical “macro” methods will emerge to treat the problem of dynamic scheduling in its full generality.

References

[AAM06] Y. Abdeddaim, E. Asarin and O. Maler Scheduling with Timed Automata, *Theoretical Computer Science* 354, 272-300, 2006.

[18] For more complicated cost functions, which are often used to express “soft” constraints, the situation may be different.

[19] Such information can be extracted by static analysis also for problems that do not have explicit constraints of this type, but it seems that explicit absolute-time constraints restrict the solution space more effectively.

[BFK$^+$01] G. Behrmann, A. Fehnker, T. Hune, K.G. Larsen, P. Pettersson, J.M.T. Romijn and F.W. Vaandrager, Minimum-cost Reachability for Priced Timed Automata, *HSCC'01*, LNCS 2034, 147-161, 2001.

[FY04] E. Fersman and W. Yi, A Generic Approach to Schedulability Analysis of Real Time Tasks, *Nordic Journal of Computing* 11, 2004.

[JM99] A.S. Jain and S. Meeran, Deterministic Job-Shop Scheduling: Past, Present and Future, *European Journal of Operational Research* 113, 390-434, 1999.

[LL73] C.L. Liu and J.W Layland, Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment, *Journal of the ACM* 20, 46-61, 1973.

[MSB98] K.N. McKay, F.R Safayeni, and J.A. Buzacott, Job-shop Scheduling Theory: What is Relevant?, *Interfaces* 18, 84-90, 1998.