Some observations on methodology and research objectives in timed systems

Following is an attempt to make sense out of some different directions I observe in research on real-time systems. It is subjective, and I may be mistaken. It is for discussion purposes.

Different communities

With respect to the topic of the role of time in systems there seem to be several communities. The Real-Time community (RTC) is essentially concerned with properties of task sets and resource allocation (reservation) policies. Typical questions circle around topics like maximizing the fraction of task sets that are feasible, utilization bounds for accepting new tasks, improving resource utilisation and responsiveness for different types of applications and different resource types, and studying virtualization models. The task sets are abstractions and are adopted as a starting point for the analysis. Additional concerns relate to having correct and effective implementations of such policies and virtualization, and about the interference of models and implementations.

The Formal Methods community (FMC) is concerned with verifying correctness of a system by checking a model of it. This can be done on a specification model (before the system has been created) or on a retrieved model, as verification after the fact. These two approaches are, in fact very different in terms of their role in the design cycle. The properties against which the model is checked can be stated as another model in the same language (which leads to a 'contains' relationship as requirement) or in a different language (which leads to some form of equivalence relation to be verified). Example questions are: is invariant *I* maintained? or: is a deadlocked state reachable?

One way of specifying a model using time is, for example, a timed automaton. This is a finite state automaton extended with the concept of time. Predicates referring to time can occur as events. A timed automaton can be mapped to timed traces or some other model that allows reasoning about the system. Questions that can be asked is whether a certain state will always be reached, whether some timed interaction leads to a response within a certain time, or not before a certain time. In the latter case the model checker can be used to construct a schedule: a counter example is a schedule about how to achieve response *before* a certain time.

Research agenda items of the FMC include modeling repetitive tasks and sensitivity to activation jitter (my interpretation). Regarding the latter, the model checking approach is to regard this as a game between an environment (that causes jitter) and scheduler (that counters it).

The Control Systems community (CSM) is concerned with stable control loops which are sense-compute-actuate cycles. Stability is achieved by choosing sampling frequency in relation to time dependent variables of the system (i.e., the assumed extreme values thereof). Integrated control takes all variables, including user interactions, into such a control loop, possibly with different sampling frequencies. Supervisory control is perpendicular to such control loops and decides about the overall behavior of the system, typically leading to controlling system *modes* (which can be regarded as the selection of local integrated control loops).

Recent work in CSM is about how to map this model to networked systems and, conversely, about how to take the execution model into account in the derivation of the control. In addition, rather that having a continuous time-based adaptation of the actuation, *event-based* control aims at corrections when required. Also for this the impact on the derivation of the controller is studied.

The studied problem

The underlying problem in all cases pertains to the design and realization of an application that satisfies user requirements, including timing requirements (that can be derived or implicit). The typical steps in the design cycle where timing properties are studied are the following.

1. *In the specification, viz., as requirement*. Then a model is built that represents the required timing behavior, e.g., a double press within a certain time, an elevator door close sequence with timing or the specification of an airbag behavior. Tools can then be used to verify if such specification satisfies certain properties. It is questionable to what extent such specification represents design. For example, an environment model may be required to obtain timing specfication. This may further be blurred by automated mapping to a code or a platform. When using such a mapping the question becomes where *composition* takes place: at the specification level (e.g. as the cartesian product of several timed automata), within the transtion or at the platform (sharing). However, there should be a clear separation between platform and application in order to leave architectural freedom. Otherwise, the solution is limited to architectural solutions that are taken into account into the translation.
2. *Within process view and/or deployment view, and subsequent detailed design.* The position assumed in this case is that timing properties that are specified or derived have to be translated into a format that is acceptable for execution on a platform (machine and interconnect). The platform API represents a separation of concerns. The typical model here consists of memory spaces and groups of tasks with deadlines. The design process (i.e., how to get to this point) is diverse and not so clear. Important is whether composition takes place through the real-time platform API or through composition of task sets.
3. *In the specification of and mapping to real-time platforms.* A real-time platform (machine + RTOS or executive) and its API assume a certain application model. Hence, the result of the design process in 2. need to be mapped unto such API. This includes concepts like temporal protection of applications and resource sharing, but also real-time communication and distribution. Rather important is that temporal isolation properties of such platforms suggest to do composition of applications at the platform level rather than in the design.
4. *In the implementation of real-time platforms.* Implementation aspects concern the scheduling and reservation steps required to give the guarantees in 3. In fact, the design of such platform is a similar problem by itself but at a meta level (i.e., instead of designing *specific* solutions, here a class of solutions must be realized).

5. *In the closed loop control of continuous systems.* Here, timing requirements are derived from control stability and perhaps latency requirements. From that point on it proceeds, in fact, as 2. and 3.

Notice that in the end the running system consists of hardware with time and event triggered activities on top of some background computation. The question is

- how such system results from the design process and
- how to show that the result satisfies the original specification.
    - this requires knowledge of the execution platform which cannot be fully abstracted.

Notive also that within 1. it is possible to further investigate properties of the models, classifying models and the like.

Observed differences

I compare the communities on the following aspects.

Place in the design cylcle. It is not obvious how e.g. timed automata are related to real-time task sets and vice versa. An integrated control loop can be regarded as one single time-triggered task but this is not very efficient (resource-wise). It seems more that they address different types of problems.

Mapping onto repetitive and timed tasks. Again, timed automata do not seem to have a natural concept of repetition. The timers involved look more like one-shot and the automaton is further event driven. Repetitive tasks are the natural result in CFM.

Composition at application level versus RTOS API as separation of concerns. Intuitively, the separation of concerns seems a better choice. However, in case of solid tooling support this is not clearly advantageous.

Do they study the same problems? What I find fascinating that only very little attention is being paid to the goal of designing and realizing solutions to problems. What I see is the following:

FMC: soundness, completeness and expressiveness are studied, different types of models, mappings between them, means to explore state spaces, based on a small and stable set of examples. While the question 'does this model have property X' is relevant for consistency and correctness, the question 'how to systematically design a system that has property X' is usually more important.

RTC: schedulability, resource allocation and sharing, application models, algorithms to maximizes utilization (online, offline) and implementations are studied assuming a given concept of a pameterized task set. Even mappings of tasks to OS concepts are not explicit while how to systematically obtain a taskset is out of scope. Important is that realistic assumpions are taken into the computations, in particular the effect of jitters. Rather than computing 'point' results, a range can be computed.

CSM: the design at hand is a stable control for a system consisting of differential equations. The resulting solution is of the form sense-compute-actuate with sampling frequency as a parameter for stability. More recently, properties of the platform are taken into account in the form of energy and disitrbution (networks). As such it is more or less a task within RTC, typically with unreasonable jitter requirements.

One way to glue them all together is to create a synthesis workflow where steps are taken by verified model transformations. Since this is often not possible to the very end, this would need to be supported by performance analysis: examining actual resource use of applications.