2014 IEEE 17th International Symposium on Object/Component-Oriented Real-Time Distributed Computing

Programming Paradigms for Real-Time Systems

Dr. Christopher Landauer, Dr. Kirstie L. Bellman
Topcy House Consulting
Thousand Oaks, California, USA
topcycal@gmail.com, bellmanhome@yahoo.com

***Abstract*—This paper is about designing and constructing real-time systems, using mechanisms that support self-organization.**

Objects make a popular and convenient paradigm for keeping persistent data encapsulated with the operations that act on it, but they tend to omit what is arguably the most important aspect of real-time systems, which is time. There are other mechanisms that do model time, but few of them make time the central part of the programming model.

In this short position paper, we offer two alternative (but complementary) approaches to modeling time (and other resource dependencies) in real-time systems, which should lead to more effective designs, since the timing and other interactions can be modeled sooner, and in any case, provide models with a different emphasis.

We also make suggestions about the expressive language that can be used for the suggested kinds of modeling, and the infrastructure that underlies the process, which provides the computational reflection that enables self-organization.

***Index Terms*—Real-Time Systems; Model-Based Development; Scenario-Based Engineering Process; Computational Reflection; Wrapping Infrastructure; Problem Posing Programming Paradigm**

I. Introduction

One of the most difficult things about building real-time systems is the interaction between functionality, timing, and resource dependencies. In part, the problem is difficult because the system under consideration must be examined from different viewpoints, that is, not only in terms of its functional characteristics, but also as embedded in its execution context, including the platform, operational environment (where the system is expected to perform), and concept of operations (how the system is expected to be used).

There have been a number of model-based approaches to this problem, many of which are useful in certain application contexts, and some more general ones that can be specialized to various application contexts [36] [35] [34] [19].

There are also some more theoretical approaches, of which [46] is a particularly good example, because it explains what is hard about the various theoretical approaches, especially in regards to compositionality, one of the main goals of many theoretical approaches.

Another related approach is *Communicating Shared Resources* (CSR) [12] [6], which is based on the *Communicating Sequential Processes* (CSP) of [16] [17], and is related to, though clearly developed independently of, our own early work on the communication protocol specification notation "com" [22] [23] [24], which changed the definition and interpretation of CSP slightly (a change that Hoare also made in between his two cited publications), and added a timing construct to CSP to support simulation. The result was a specification notation ideally suited to defining complex concurrent systems that allows both simulations and verifications of exactly the same model. It also inherently supports distributed models with distributed time [21] [12], allowing developers to model the vagaries of data transport and processing, in addition to the difficult issue of synchronizing clocks.

Among the other interesting mathematical methods for modeling real-time systems are timed petri nets [5] and the related stochastic activity nets [44], process algebras [6] [10], and timed automata [1] [5]. All of them have advantages and disadvantages. They are all explicitly about events and times, and have varying analytic and simulation capabilities, but none of them has a useful level of task content, that is, what is a task supposed to do, and how is its effect to be incorporated into the running model.

Object-oriented methods have been used for implementing real-time systems [34], but we think they are insufficient as a design and implementation paradigm. Objects by themselves make a popular and convenient paradigm for keeping persistent data encapsulated with the operations that act on it, and decompose the system functionality into convenient components. However, they omit what are arguably the other most important aspects of real-time systems, which are the flow of time, limitations on resources, and task and resource dependencies.

We do understand that most design approaches either expect serious modeling of the task dependency and deadline lists, or *a priori* computations based on common simplifications of the problem, such as Rate Monotonic Analysis (RMA) or Earliest Deadline First (EDF). What we are suggesting here is that that modeling become the centerpiece of the design process and that time behavior become the centerpiece of the modeling. Moreover, we are also recommending [26] a large increase in the amount of self-monitoring available over traditional self-adaptive software [13].

The rest of this paper is organized as follows. In Section II, we describe the relationship between objects and real-time models, and provide some background on the common services made available in Real-Time Operating Systems (RTOS), and a popular reference model for real-time systems that explains many of the important considerations for design or modeling. (for [46] and for us, they are essentially two different times in the same process).

In Section III, we describe two kinds of replacement model

1555-0885/14 $31.00 © 2014 IEEE
DOI 10.1109/ISORC.2014.42

approaches, event mesh models and resource access models. We show how to use them separately, and what kind of analyses they support, and then we show how to combine them.

We claim that these models can be used to simulate the prospective behavior of a system, at gradually increasing detail, so that they become good predictors of system performance. Once the models are good, there still remains one problem: converting good models of a real-time system into a good real-time system program. We believe that our approaches ease this transition somewhat, but we need to prove that through trying them with different difficult systems.

In Section IV, we briefly describe the infrastructure we use to make all of this modeling work, and to support the transition from model to program. We show how to implement such systems, explain why we think they are good candidates for real-time operational environments [33], but we cannot illustrate the method here with an example implementation; instead we refer to other more detailed descriptions [2] [3] [26] [29] [33]. This infrastructure also allows us to design a large number of instrumentation processes, to monitor all aspects of system behavior during development, and then to compile away the ones we do not use at run time, so the design effort does not cost more than is required to make sure that the necessary instrumentation is available for the self-monitoring part of self-adaptation. The flexibility that underlies self-adaptation reduces efficiency; we choose to make efficiency choices at the very last design steps, so we know which ones we need.

We describe a generic programming notation called *wrex* that we developed for use with Wrappings, which allows far fewer design decisions to be made before simulating a system. That allows design decisions to be based on observation and knowledge, instead of only expertise and engineering judgment. Using this notation is not necessary to use the rest of our approach, but it is convenient because it already supports the partial evaluation [9] we use to simplify the model code and produce the operational code.

We also describe the system development process that we use to build such systems, which differs from conventional methods by taking the basic artifacts of development, considered as partial models of aspects of the system in its environment, and retaining them all in a model hierarchy, so that different levels of detail can be related to each other. That way, the most detailed models can be validated against the original expectations, and it also eventually becomes the definition of the run time system. That is, the code in the models is compiled into the code in the system.

We have described this basic architectural approach before [26] [27] [29], so this section is necessarily brief, and, in fact, rather cryptic.

In Section V, we briefly describe an application of this approach to fault management of cyber-physical systems in complex environments. We show how to build explicit models of the likely or possible faults, and use those as contingency system behavior descriptions, which makes fault management a natural part of the architecture, instead of a complication added at the end, as is too often the case.

Finally, we describe our conclusions and present some prospects for future model analysis.

II. Objects in Real-Time

One of the complications of any kind of theoretical model of real-time systems is their lack of compositionality, that is, just because two parts of a program have property P, that does not mean that the program has property P (this is also a big problem in the information security world).

In particular, this partially debunks the utility of modularity and encapsulation, because timing is not generally a composable property, and some other properties are not composable at all (deadlock-free, minimal progress in every time interval has a permanent positive lower bound, etc.).

Instead, to reason about the properties of the entire system requires perspective, some way to examine local behavior in the context of all the other system behavior. The Assume-Guarantee or contracts-based approach [15] [37] does that with some success.

Objects are a good way to collect persistent data and maintain it with the operations that can change it, but that is not really the real-time problem. The real-time problem is about scheduling processes or process fragments, to make sure that enough of them get enough computational resources to satisfy their needs well enough and soon enough.

This sounds to us like the old time-share problem, at a much faster time scale, and yet we're trying to solve it without very much operating system support infrastructure, since, of course, that also competes for the same resources.

A. Real-Time Operating Systems

It has gradually become clear that having an underlying real-time operating system (RTOS), with minimal services, is extremely useful for reliable real-time applications [7] [42]. The most familiar RTOS is kernel-based, and has a System Call Interface to request services. The services provided usually include

- task management and scheduling,
- interrupt servicing.
- communication, and its subset synchronization,
- memory management vs. memory protection.

One of the references [7] compares RTOS functionality to Linux and various real-time extensions of commonly used operating systems. The other one [42], has what we believe to be the right definition of real-time "the system does not control its own time domain", and its corollary "it is a widespread myth that real-time systems have to be fast". Instead, a real-time system only needs to be "good enough" "soon enough". While we're here, we'd like to dispel another myth about real-time analysis. There is a widespread assumption that if a worst case execution time analysis succeeds, then the system is schedulable, but there are instances of schedules for which the overall time required by a set of tasks is longer

when some of the tasks take less time, or when an additional processor is made available [14].

There are long discussions of the different kinds of schedulers that may be used, from Rate Monotonic (RM), which is optimal among fixed priority schedulers, Deadline Monotonic (DM), which is convenient for some applications, and optimal for static priorities when deadlines are not more than the periods, and Earliest Deadline First (EDF), which is optimal among a certain class of varying priority schedulers. But all of them require knowing the task time distribution, or at least the minimum and maximum times.

The biggest complication to any prior analysis is preemption, the pausing and resuming of a task, which requires retaining enough data that the task can start exactly where it left off (or restart near enough to that point that no data and precious little time is lost).

If we could severely limit where a task can be preempted, then we might be able to reduce the difficulty. We can have the compilers identify break points in every source module, with provable maximum inter-point intervals. The first part is easy, since compilers already divide program source code into a graph of sequential segments. We would then need to have good estimates for the times of those segments, whence the compiler could provide that information and the scheduler could use it for recurrent predictive scheduling.

B. Perspective

However, we seem to run our programs for the very first time, every time we run them. We almost never keep the global or even historical perspective that would help us understand when a program is working properly (have you ever noted an unusual sound in your car, or an unusual smell in your house and thereby detected an early sign of a potentially dangerous problem, so that it could be avoided?), and even when we do, that information is not made available to the running program so that it can make such an assessment. We think that this is a serious failing of most real-time systems, especially now that modern processors are fast enough to do that. This is another example of the conflict between efficiency and robustness.

C. Example Reference Model

There are many design methods for real-time systems, but we show just one here for comparison. This is a reference model for real-time systems [36], which is quite popular, because it addresses many of the implementation issues that concern most real-time systems. This description is derived from several sources [8] [18] [4].

A real-time system is defined by

- a workload model,
- a resource model, and
- scheduling and resource management algorithms.

The workload model is about what behaviors the system will be asked to perform, the resource model is about what tools and capabilities the system has to apply to them, and the rest is about how the system coordinates its activities.

The system resources are divided into two types: processors processors and resources. Processors are active resources, such as computers, data link controllers, disk controllers, database servers, etc.. Processors have an associated speed of operation. The model calls the passive resources just resources, including memory, locks, semaphores, other data. Resources do not have a speed of processing (we think that this is an oversimplification, since memory and disk controllers should be processors, unless they are known not to be a bottleneck). Some resources are reusable, some are consumed.

A *job* is a unit of work. A *task* is a set of jobs that together accomplish some desired activity. Each job runs on a processor; each job depends on some resources. The resource model here is particularly simple, since it does not seem to allow resources to have a capacity for concurrent use (which requires the jobs to have a desired capacity of use for each resource it uses).

Each job is characterized by:

- temporal parameters:
  - release time, deadlines (we think these are not defined within the job, since they depend on external activity and intentions);
  - release time jitter
- functional parameters
- resource parameters
- interconnection parameters

Of course, the most important information for the real-time analysis are the timing prediction parameters. The execution time of jobs has many uncertainties:

- execution variability - caches, pipelines, etc.;
- internal variability - branches, loops, etc.;
- external variability - preemption, interference.

All of these issues lead to difficulties in making *a priori* decisions about how much time to allow, and promote the use of simulation models to help.

The workload model, or task graph, is a partially ordered set of tasks, each with timing information. The ordering represents any dependency that requires one task to finish before another. This method has one level of task decomposition, into jobs, which we think is not nearly flexible enough. Our approach to this problem underlies our event mesh model in Section III-A. It includes the notion of interferences, which occur when more than one task tries to access a common resource.

The resource model, or capacity graph, contains all of the passive resources that the system must use, and their acceptable patterns of use, generally just in terms of exclusion or number of simultaneous accesses allowed. This is the weakest part of the model, since it cannot handle complex interaction dependencies. Our approach to this problem underlies our resource access model in Section III-B.

Finally, the available scheduling algorithms have to take into account the dependencies and interferences among the tasks., and to handle the sporadic tasks and unexpected conditions, such as interrupts. They also have to manage the bane of all RTS design: preemption. Apparently, preemption renders

many scheduling problems undecidable [20], at least without other simplifications. Our approach to this problem underlies our resource access model in Section III-A.

Even the decidable scheduling problems (i.e., the ones without preemption) are prohibitively complex.

It might be possible to introduce a notion of timed objects, in which every method has a timing and resource utilization profile, either as a specification / requirement or as an observation / behavior. Instead, we prefer to keep the object style strictly functional, and change the approach.

Two alternatives to other modeling methods separately emphasize two important aspects of real-time systems:

- events
- resources

The next Section defines them and explores them in some detail.

III. Replacement Models

We offer two alternative models, one based on managing the collection of events that are intended to occur in a system, and the other managing the requests, allocations, and interferences of resources. Strictly speaking, we recommend using both together, with some analysis hints for making sure that they are consistent.

A. Event Mesh Models

This viewpoint replaces the objects with event mesh models, which are partially ordered sets of interacting events (the similarity to Petri Nets should be noticed). In this approach, events are encounters among resources, including data or control transfers, as well as task starts and finishes. All encounters among tasks are due to communication events or other interaction mediated by a resource (for example, function calls are communication events that imply waiting for a response). Each event has some duration, though many may be instantaneous at the time scale of the model.

This approach also includes a task graph, which is a completely hierarchical graph of tasks, with jobs as the atomic tasks, and dependencies defined whenever a task must finish before another starts, or less strongly when some parts of a task must finish before some parts of another start. We would also add a completely different set of edges called interferences, which are about contending for the same resource (or asking too much of a limited resource). The interferences among tasks would be defined by the interferences among their sub-tasks.

This task graph is like a program, defining the set of subsequent tasks when each task finishes. and the sets of tasks that need to be finished before each task starts. Then the event trace, the partially ordered set of events that occur, is like the program execution trace.

We think that the problem of preemption largely derives from poor task partitioning, either incomplete or incorrect. We advocate starting with a Failure Modes and Effects Criticality Analysis (FMECA) [11] to help determine what can go wrong, what the effects of a failure might be, whether or not is it critical to address the failure, and how soon it needs to be addressed, This analysis provides the necessary response time. Then partitioning of tasks should go to the largest units whose uninterruptable sections are less than that time (or less than half, to reduce the window of potential interference).

Units that can safely be cancelled quickly need not partition further. We also want to build low priority jobs so that they can be cancelled immediately without errors. It does mean that they need to start over when the emergency finishes. It depends on how fast they can be cancelled and restarted. That time had better be less than the smallest necessary response time.

That means that all computation is short sequential computing segments with predictable duration distributions.

The modeling language to be used to write these models can essentially be any programming language, augmented with segment timing (and other resource usage) notes. Compilers can separate these requirement notes out into a source code meta-data repository, which is initially empty, but is augmented each time the program is run (there will need to be some decay rules for update, since the program segment may change). That way, there is always some notion of time requirements for each job (sequential statement sequence). We can then insist that these segments are not preemptable. In cases where the FMECA has discovered a very short term response need, we can even have the compiler break up the sequences into shorter ones to match the latency need. This code and the corresponding resource load behavior can be used for simulations of the system behavior and for monitoring of the execution behavior.

The task graph is relatively easily produced by a compiler if it has access to all of the source code (it has to consider out of scope references), so, for example, library functionality should be as localized as possible (since we often do not get the library source code)

There is an extension to the task graph called the precedence constraint graphs, which allows a task to depend on a boolean expression of the incoming task completions. The easiest to model are the and and or connections, but the constraints can also be made arbitrarily complex. In most applications, however, the condition will be relatively simple. There would need to be some new model notations to indicate these complex dependencies.

For example, in our CARS testbed [2] [3] [26] [29] [33], each car has a short sensor integration loop, from which the car develops notions of its position and velocity in the world. It computes a fault assessment via consistency analyses of the sensor input, and ascribes a credibility score to9 each sensor which is used in the computation of the dynamics,

Each sensor has a task graph derived from the system goals (one of a set of games), and can decide at any time whether it is capable of continuing or not.

B. Resource Access Models

This viewpoint replaces the objects with resource access models, which are of two flavors: local protocols that define

how much of a resource can be accessed, under what circumstances and for how much time, and global access patterns, which are partially ordered sets of accesses by tasks (yes, this approach has a task graph, just as the event approach has the local resource protocols). In this approach, processors are also resources that pick and choose which tasks to run at any given time.

Each resource should be defined with capacities for simultaneous use, as well as interaction adjustments (for example, two costs the sum and some interference time or capacity). That is, each resource has a definition of local constraints for each use, in time and capacity, and a definition of global constraints across all simultaneous uses. These constraints perform arbitration among multiple requests, and may include explicit delays and even refusals.

Because the resources decide which tasks access them when, at what level, and for how much time, we can address all resource contention issues directly. The global access graph for a processor contains the task dependency information that is not dependent on resource contention. This model doesn't have resource to resource interactions modeled explicitly, so that if there are suggested alternatives, that only appears implicitly.

This approach allows the processors to drive the system, making them analogous to interpreters of their task and access graphs.

For example, in our CARS testbed [2] [3] [26] [29] [33], the pace of events is such that the only resource that is shared enough for potential contention is the processor. There is a large set of tasks, from get current sensor values (at different rates for each sensor), to assimilating them, using them to update the dynamics estimates, integrating the dynamics, the obstacles, and the current goals into a course of action. There are clearly data dependencies among these tasks, and the resource access model will contain those explicitly.

This approach is not explained in much detail, since we are more interested in the combined approach that follows.

C. Combination Models

The event mesh model takes the viewpoint of the active entities attempting to use resources to do something, possibly encountering interferences. The resource access model takes the viewpoint of the resources attempting to satisfy all the requests they receive, even simultaneous ones (we can assume that the hardware serializes the requests, unless too many occur).

For this combined model, we want to use the advantages of each of the previously described approaches.

We take events to be task starts or stops, resource access requests and responses, allocations and releases, all instantaneous. Resource accesses that are faster than the model time scale can also be represented explicitly without the four parts.

Atomic tasks are short, representing uninterruptable sequences of statements without looping or branching.

Tasks are built recursively from other tasks, from contingencies, iterations, and resource accesses.

This construction defines a task graph that includes ordering dependencies. Under the assumption that the times required for each segment are fairly precise (they need not be exact), the main uncertainty lies in when tasks start, once they can start, and how long they wait for accesses.

The resource graph for each resource is a partially ordered set of all events that are accesses to that resource. It can be studied directly for potential conflicts.

The event graph defines all of the encounters, and can be compared to the task and resource graph for consistency.

This collection of information can be interpreted (simulated) quite easily, and various exploratory excursions can be studied (i.e., sensitivity to execution time intervals, sensitivity to different resource capacities, expanded scenario scope, more difficult environmental behavior, faults and failures in the hardware and software, etc.).

These experiments can be used to identify potential difficulties as early in the design process as possible, so that the scope of the redesign / rework is minimized, and the model hierarchy allow us to trace the difficulties directly up to the design stage at which the problem occurred (not always an error; sometimes the problem occurred because there was not enough information to make a good choice at that time). They can also be used to learn what anomalous behavior is associated with various kinds of failures, so that they can be more easily identified.

The task models can gradually be replaced by task code, and the variables used to communicate among tasks then replace the resources (or the visible interfaces to them). The graphs are an abstraction of the *wrex* code derived from the design, and the task code can be any appropriate language. If we want the flexibility of Wrappings, then we can simply keep the *wrex* code, with the inevitable changes that occur as we learn more about the model. We should emphasize here that *wrex* is not a necessary part of this process; any programming language can be used. We use *wrex* because we already have the partial evaluator and the Problem Posing interpreter, two tools that use the Wrapping infrastructure as we have described earlier. The effort required to write a partial evaluator is roughly the same as that for the interpreter, which is about a summer master's project level, possibly easier if you have access to the compiler source code, or even a good parser for the language.

This approach has many aspects in common with the reference model we described earlier, but each component is more expressive (which reduces the scope of the formal mathematical results that can be derived). In addition, this model can be simulated directly, and also migrated into the operational code.

Another aspect of this approach deals with the difficulties of preemption. The slogan is "prevention, not preemption". We think that periodic tasks are really intended for continuous low density activities, and if a slow processor were available for them, they would not be considered further, but would simply be farmed out to the slow processor. The periodic scheduling style is just a way to get processes of different necessary speeds (and low duty cycle) to share a single processor. If

task context switching more expensive than interprocessor communication in a small local network, then we should not bother with interleaving concurrent tasks; we should just use lots of simple processors. Remember, the tasks come first, the hardware second, so we can decide exactly how many processors we need (taking into account some spoilage and eventual breakdown), in advance.

IV. Expressing Models and Writing Programs

In this Section, we describe the infrastructure we have developed for constructed complex systems, and show how it can be used for embedded real-time system design and implementation. We have shown elsewhere how it provides an enormous amount of self-adaptivity [31] [32].

We start with the Wrapping approach, which is an integration infrastructure specifically designed to provide the Computational Reflection necessary for self-organization, Then we describe a generic programming notation called *wrex*, which allows very late design decisions. and finally, we describe the Scenario-Based Engineering Process (SBEP), which is a design and development process well-suited to Wrappings and *wrex*, though it was defined independently.

A. Wrapping Infrastructure

The Wrapping infrastructure [30] [29], or the Wrapping Approach to integration infrastructure, is defined by the Problem Posing Programming Paradigm [27], a strict separation of information service requests in a system (called em problems) from information service providers (called *resources*), and the knowledge-bases and integration processes that map problems to resources in context.

The integration processes are also resources, so they are also Wrapped, giving this approach as much Computational Reflection as may be desired. The details can be found in the references.

The Wrapping approach is well-suited to the adaptivity requirements of self-modeling systems [31] [32], which are systems that contain models of their own behavior, which they interpret to produce that behavior. When they change the model, the behavior changes.

B. Expressing Models *in* wrex

The Wrapping Expression notation *wrex* [30] [29] is a generic programming notation that reduces the number of implementation decisions required by the programmer. It partly alleviates the need to define certain aspects of a system design too soon [45].

It relies on the Problem Posing Programming Paradigm to separate problems (analogous to specifications of services) from resources (analogous to implementations providing the service in a particular context), and to write each resource in *wrex* as an organized collection of interacting problems, then using Wrappings to associate the problems with resources in many different ways, according to context.

The use of *wrex* relies on the fact that we can use partial evaluation [9] to remove and reduce any decisions that will not change during run time. In this usage, partial evaluation is a kind of compilation in context that retains only that flexibility that is required for the application.

The syntax of *wrex* is not the point here; it is a rather conventional block- and statement-based notation. In fact, any programming or modeling language can be interpreted using Problem Posing, because compilers can almost always identify which instances of a term are definitions, and which instances are invocations.

The developers can then use program fragments and common conventions to build small resources, and combine them using familiar techniques into larger units. The run-time support is provided by the Wrapping infrastructure. In addition, the same problem structure can be combined with resources for simulation, and also resources for execution, so that the bulk of the code is identical (same problem, different context, means different resource). That greatly simplifies the transition from modeling a system to running it.

C. Model-Driven Development

Our model-driven development process begins with the Scenario-Based Engineering Process (SBEP) [38] [39], which we describe quickly, and continues with our Wrapping approach at the implementation end [27] [28] [33]. Our process is not altogether different from other [19], except that we explicitly defer the notion of requirement, first to expectation, and then to responsibility. In all systems we have examined for which we have seen the requirements, they have far too small a granularity: they are like the machine language of expectations and responsibilities, extremely difficult to map to the actual expected system behavior, replete with errors and inconsistencies, without adequate explicit context and always incomplete. This is why we like to have executable observable models from the beginning: we can check for completeness, validate expectations, examine alternatives, and generally reduce these typical requirements errors. We have developed other empirical methods for examining requirements [25], but explaining them is beyond the scope of this paper.

The designers begin with stakeholders, who are the organizations that will develop or deploy or depend on the system, expectations and scenarios, which are definitions of what the system is supposed to do in certain circumstances. Preferably, these scenarios are defined in terms of simple system simulation models with specified expectations (either computational or even formal). This part includes a domain analysis to determine what the relevant entities and processes are for the system under construction (this includes whatever aspects of the operational environment are relevant)..

Then the different roles and responsibilities are defined, roles defining acting entities and responsibilities defining what is expected of them. Design is then determining a hierarchical structure for these entities, and refining the responsibilities into activities (there are clearly verification steps to show that the activities carry out the responsibilities). Then identify the activities as problems (service requests), define resources to address those problems, and the mappings from problem to

resource in context. The decomposition of activities can go into as much detail as is appropriate (the stopping point is always an engineering judgment call).

There is much more detail to be found in [27]. The point is that this process allows developers to go quickly from intentions to simulation models to system software, in such a way that there is an executable analyzable model almost from the beginning.

No matter what kind of schedulability analysis or other theoretical analysis is done, nobody will believe it applies to a real system until they see the model in execution (and many not even then), so for us it is much more important to be able to simulate a system early than to prove things early.

Of course, once we can run the model and observe its behavior, we can formulate hypotheses about that behavior, and then go back and try to prove them.

These hypotheses can come from commonalities in the behavior, as identified by persistent pattern inference, from notably avoided areas in the state space, or from much more sophisticated mathematical techniques such as manifold discovery (also called "dimensionality reduction" in a particular special case, it is a generalization to non-linear subspaces of the usual eigenvalue analysis done for Principal Component Analysis).

The point is that the hypothesis can come from anywhere, but its utility lies in how much of it can be proven. This is clearly an iterative process, since we can adjust our notations to reflect the persistent patterns, and continue in a "test a little, prove a little" cycle for as long as we want.

V. Fault Management Support

One of the main difficulties in operating cyber-physical systems is that they break, either because the real-world environment has become too difficult, there were undetected manufacturing or materials defects, or from simple wear and tear.

In this Section, we show how our design approach allows us to build fault detection and management processes into the core of the system, so that we can expect the system to deal with many anomalies.

A fault management system [11] [40] [41] is responsible for monitoring system behavior (at different event densities according to a perceived level of risk), detect anomalies (unusual behavior, as defined by the expectation models [28]), respond to explicit alarms from various system components, and decide what to do.

For example, if the system has all of the detailed models of the timing and task dependencies, then it knows how long any given task should take, so it can indicate a problem when that time is exceeded.

Similarly, the FMECA described above [11] can be used to decide which anomalies and faults should be addressed, and which should be mostly ignored (and placed into the context so that a different operating mode can be used). This is especially true for hardware failures, since they often force the system to change its mode to a smaller and less capable set of behaviors that do not use the broken parts.

At a higher level of perspective, the system knows something about what should be happening in the form of patterns of tasks, so it can compare the task execution observations with the model-based expectations to detect potential faults [28].

The message here is that the developers can know what sorts of anomalies and faults may occur, can decide which ones to address immediately, which ones require a change of operations, and which ones to announce and ignore, and can make all of those decision and response processes part of the system.

VI. Conclusions

There are many methods for modelling real-time systems, and some are quite popular in different application domains, but none of them are so widespread as to be universally accepted. We think there need to be many more, and many more experiments with them, to improve the state-of-the-art.

We have provided two more model mechanisms that warrant some study of their properties, and especially their drawbacks. We actually believe that a combination of these methods with others, all used for different aspects of the same system, will be the most effective, because they will force developers to confront different design questions.

References

[1] Rajeev Alur, David Dill, "Automata for Modeling Real-Time Systems", p.322-335 in Michael S. patterson (ed.), *17th Annual Colloquium on Automata, Languages, and Programming*, 16-20 July 1990, Warwick U., Coventry (1990)

[2] Dr. Kirstie L. Bellman, Dr. Christopher Landauer, Dr. Phyllis R. Nelson, "Managing Variable and Cooperative Time Behavior", *Proceedings SORT 2010: The First IEEE Workshop on Self-Organizing Real-Time Systems*, 05 May, part of *ISORC 2010: The 13th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 05-06 May 2010, Carmona, Spain (2010)

[3] Dr. Kirstie L. Bellman, Dr. Phyllis R. Nelson, "Developing Mechanisms for Determining G̈ood Enoughïn SORT Systems", *Proceedings SORT 2011: The Second IEEE Workshop on Self-Organizing Real-Time Systems*, 31 March 2011, part of *ISORC 2011: The 14th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 28-31 March 2011, Newport Beach, California (2011)

[4] Ricardo Bettati, "Lecture 2: reference model for real-time systems", CPSC 663, Real-Time Systems, TAMU (Fall 2013), http://faculty.cs.tamu.edu/bettati/Courses/663/2013C/Slides/slides_overview.html, last accessed 03 February 2014

[5] Patricia Bouyer, Serge Haddad, Pierre-Alan Reynier, "Timed Petri Nets and Timed Automata: On the Discriminating Power of Zeno Sequences", p.420-431 in Vol. II of Michele Bugliesi, Bart Preneel, Vladimiro Sassone, Ingo Wegener (eds.) *Proc. ICALP 2006: The 33rd International Conference on Automata, Languages, and Programming*, 09-16 July 206, Venice, Italy (2006)

[6] Patrice Brémond-Grégoire, Insup Lee, "A process algebra of communicating shared resources with dense time and priorities", *Theoretical Computer Science*, v.189, iss.1-2, p.179-219 (15 December 1997)

[7] Herman Bruyninckx, *Real-Time and Embedded Guide*, http://people.mech.kuleuven.be/~bruyninc/rhowto.pdf, last accessed 02 February 2014

[8] S. Congiu, "Lecture 5: A reference model for RT systems", Course , U.Padua (Fall 2008), http://www.dei.unipd.it/corsi/so2/05-Reference-Model.pdf, last accessed 03 February 2014

[9] C. Consel, O. Danvy, "Tutorial Notes on Partial Evaluation", *Proceedings 20th PoPL: The 1993 ACM Symposium on Principles of Programming Languages*, Charleston, SC (January 1993)

[10] Rocco De Nicola, "A gentle introduction to Process Algebras", Notes, U. http://www.pst.ifi.lmu.de/Lehre/sose-2013/formale-spezifikation-und-verifikation/intro-to-pa.pdf, last accessed 07 February 2014

[11] Roland Duphily, "Space Vehicle Failure Modes, Effects, and Criticality Analysis (FMECA) Guide", TOR-2009(8591)-13, 29 June 2009, The Aerospace Corporation (2009)

[12] Richard Gerber, In Sup Lee, "Communicating Shared Resources: A Model for Distributed Real-Time Systems", Technical Report MS-CIS-89-26, Computer and Information Science Dept., U.Penn. (May 1989)

[13] Robert P. Goldman, David J. Musliner, Kurt D. Krebsbach, "Managing Online Self-Adaptation in Real-Time Environments", in [43]

[14] Ronald L. Graham, "Bounds on multiprocessing timing anomalies", *SIAM J. Applied Mathematics*, v.17, no.2, p.416-429 (March 1969)

[15] Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, "You assume, We guarantee: Methodology and Case Studies", p.440-451 in Alan J. Hu, Moshe Y. Vardi (eds.) *Proc. CAV'98: The 10th International Conference on Computer-Aided Verification*, 28 June - 02 July, 1998, Vancouver, Canada, LNCS 1427, Springer (19989)

[16] C. A. R. Hoare, "Communicating Sequential Processes", *Comm. ACM*, v.21, No.8, p.666-677 (August 1978)

[17] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall (1985)

[18] Dongsoo Stephen Kim, "Lecture Notes on Embedded Real-Time Software", ESW 5004, IUPUI (Summer 2011), http://www.ece.iupui.edu/~dskim/Classes/ESW5004/RTSysLectureNote-ch02AReferenceModelforReal-TimeSystems.pdf, last accessed 02 February 2014

[19] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Springer (2nd ed., 2011)

[20] Pavel Krčál, Wang Yi, "Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata", *Proc. TACAS 2004: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 29 March - 02 April 2004, Barcelona (2004)

[21] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM* vol.21, no.7, pp.558-565 (July, 1978)

[22] Christopher Landauer, "Performance Modeling of Protocols", Paper 16.2, vol. 2, pp. 219-221 in *Proceedings of MILCOM'84: The 1984 IEEE Military Communications Conference*, October 1984, Los Angeles (October 1984)

[23] Christopher Landauer, "Network and Protocol Modeling Tools", pp. 87-93 in *Proceedings of the 1984 IEEE / NBS Computer Networking Symposium*, December 1984, NIST, Gaithersburg, Maryland (December 1984)

[24] Christopher Landauer, "Communication Network Simulation Tools", Part 3, pp. 995-1001 in *Proceedings of the 16th Annual Pittsburgh Conference on Modeling and Simulation*, April 1985, Pittsburgh, Instrument Society of America (Fall 1985)

[25] Christopher Landauer, "Correctness Principles for Rule-Based Expert Systems", pp. 291-316 in Chris Culbert (ed.), *Special Issue: Verification and Validation of Knowledge Based Systems*, *Expert Systems With Applications Journal*, Volume 1, Number 3 (1990)

[26] Christopher Landauer, "Abstract Infrastructure for Real Systems: Reflection and Autonomy in Real Time", *Proceedings SORT 2011: The Second IEEE Workshop on Self-Organizing Real-Time Systems*, 31 March 2011, part of *ISORC 2011: The 14th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 28-31 March 2011, Newport Beach, California (2011)

[27] Dr. Christopher Landauer, "Problem Posing as a System Engineering Paradigm", *Proc. ICSEng 2011: The 21st International Conference on Systems Engineering*, 16-18 August 2011, Las Vegas, Nevada (2011)

[28] Christopher Landauer, "Model Comparison for Fault and Attack Detection", (presented to) *FSW12: 2012 Spacecraft Flight Software Workshop*, 07-09 November 2012, San Antonio, Texas (2012)

[29] Christopher Landauer, "Infrastructure for Studying Infrastructure", *Proceedings of ESOS 2013: Workshop on Embedded Self-Organizing Systems*, 25 June 2013, San Jose, CA; part of *2013 USENIX Federated Conference Week*, 24-28 June 2013, San Jose, CA (2013)

[30] Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Chapter 8, pp.108-154 in Gene McGuire (ed.), *Software Process Improvement*, Idea Group Publishing (1999)

[31] Christopher Landauer, Kirstie L. Bellman, "Self-Modeling Systems", pp.238-256 in R. Laddaga, H. Shrobe (eds.), "Self-Adaptive Software", Springer Lecture Notes in Computer Science, Volume 2614 (2002)

[32] Christopher Landauer, Kirstie L. Bellman, "Managing Self-Modeling Systems", in R. Laddaga, H. Shrobe (eds.), *Proceedings of the Third International Workshop on Self-Adaptive Software*, 09-11 June 2003, Arlington, Virginia (2003)

[33] Dr. Christopher Landauer, Dr. Kirstie L. Bellman, Dr. Phyllis R. Nelson, "Modeling Spaces for Real-Time Embedded Systems", *Proceedings SORT 2013: The Fourth IEEE Workshop on Self-Organizing Real-Time Systems*, 20 June 2013, part of *ISORC 2013: The 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 19-21 June 2013, Paderborn, Germany (2013)

[34] Philip Laplante, *Real-Time Systems Design and Analysis*, Wiley IEEE (4th ed., 2011)

[35] Ákos Lèdeczi, Árpád Bakay, Miklós Maróti, "Model-Integrated Embedded Systems", pp. 99-115 in [43]

[36] Jane W.S.Liu, *Real-Time Systems*, Prentice Hall (2000)

[37] Alessio Lomuscio, Ben Strulo, Nigel Walker, and Peng Wu, "Assume-Guarantee Reasoning with Local Specifications", p.204-219 in *Proc. ICFEM2010: the 12th International Conference on Formal Methods in Software Engineering*, 17-19 November 2010, Shanghai, LNCS 6447, Springer (2010)

[38] Karen McGraw and Karan Harbison, *Knowledge Acquisition using the Scenario-Based Engineering Process*, Lawrence Erlbaum (1995)

[39] Karen McGraw and Karan Harbison, *User-centered Requirements: The Scenario-Based Engineering Process*, Lawrence Erlbaum (1997)

[40] B. M. O'Halloran, R. B. Stone, I. Y. Tumer, "A Failure Modes and Mechanisms Naming Taxonomy", pp.1-6 in *Proceedings of RAMS2012: Reliability and Maintainability Symposium*, 23-26 January 2012, Reno, Nevada (2012)

[41] Claudia Priesterjahn, Christian Heinzemann and Wilhelm Schäfer, "From Timed Automata to Timed Failure Propagation Graphs", *Proceedings SORT 2013: The Fourth IEEE Workshop on Self-Organizing Real-Time Systems*, 20 June 2013, part of *ISORC 2013: The 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing*, 19-21 June 2013, Paderborn, Germany (2013)

[42] Franz Rammig, Michael Ditze, Peter Janacik, Tales Heimfarth, Timo Keerstan, Simon Oberthuer, and Katharina Stahl, "Basic Concepts of Real Time Operating Systems", Chapter 2, p.15-46 in Wolfgang Ecker, Wolfgang Müller, Rainer Domer (eds.), *Hardware-dependent Software: Principles and Practice*, Springer (2009)

[43] Paul Robertson, Howie Shrobe, Robert Laddaga (eds.), *Self-Adaptive Software*, *Proceedings of First International Workshop on Self-Adaptive Software*, 17-19 April 2000, Oxford U., England, Springer Lecture Notes in Computer Science, vol.1936, Springer (2001)

[44] William H. Sanders, John F. Meyer. "Stochastic Activity Networks: Formal Definitions and Concepts", p.315-343 in Ed Brinksma, Holger Hermanns, Joost-Pieter Katoen, *Lectures on Formal Methods and Performance Analysis*, *Proc. First EEF/Euro Summer School on Trends in Computer Science*, 03-07 July 2000, Bergen Dal, The Netherlands, LNCS 2090, Springer (2001)

[45] Mary Shaw, William A. Wulf, "Tyrrannical Languages *still* Preempt System Design", pp. 200-211 in *Proceedings 1992 International Conference on Computer Languages*, 20-23 April 1992, Oakland, California (1992); includes and comments on Mary Shaw, William A. Wulf, "Toward Relaxing Assumptions in Languages and their Implementations", *ACM SIGPLAN Notices*, Volume 15, Number 3, pp. 45-51 (March 1980)

[46] Joseph Sifakis, "Modeling Real-Time Systems: Challenges and Work Directions", p.373-389 in Thomas A. Henzinger, Christoph M. Kirsch (eds.) *Proc. EmSoft 2001, the First International Workshop on Embedded Software*, 08-10 October 2001, Tahoe City, California (2001), LNCS 2211, Springer (2001) http://www-verimag.imag.fr/~sifakis/RECH/emsoft01.pdf, last accessed 02 February 2014