Resource Modeling for Embedded Systems Design*

Oleg Sokolsky
University of Pennsylvania

Abstract

The paper describes a formal framework for designing and reasoning about resource-constrained embedded systems. The framework is based on a series of process algebraic formalisms which have been previously developed to describe and analyze various aspects of real-time concurrent systems. We present a uniform framework for formal treatment of resources and illustrate modeling of common resource classes.

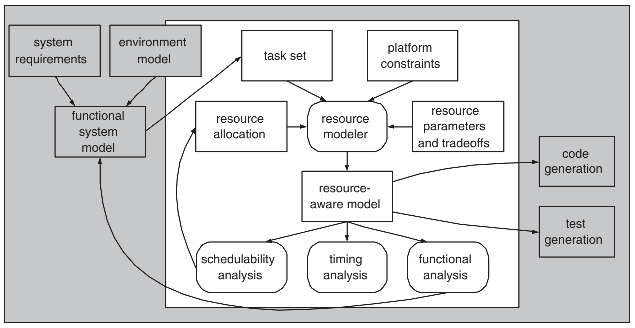

Figure 1. Model-based development framework

1. Introduction

An embedded system consists of a collection of components that interact with each other and with their environment through sensors and actuators. Embedded software is used to control these sensors and actuators and to provide application-dependent functionality. An important distinguishing characteristic of embedded applications is limited resources (processing power, memory, network bandwidth, power consumption, *etc.*). Many embedded systems are part of safety-critical applications, *e.g.*, avionic systems, manufacturing, automotive controllers, and medical devices. Their reliability has to be properly assured.

At the same time, complexity of embedded systems has been progressively increasing due to more powerful microprocessors and wider use of networking. A natural response to the increasing complexity is an increased emphasis on model-based development of embedded software. Models can be constructed for embedded systems and their properties can be analyzed through simulation and model checking. Models can be used to generate code skeleton and task structures and then platform dependent code can be added to work on specific environment. Since it may not be possible to completely automate code generation, models can later be used to validate the implementation.

Figure 1 provides an overview of our model-based development framework. The development process begins with system requirements that specify high-level behavior of the system, and a model of the system environment. They are combined into a single model that captures the system functionality with respect to a given environment. The system contains a collection of concurrent components. Components can communicate with each other during execution, representing a wide variety of data dependencies and precedence requirements. Such a model, however, does not take resource constraints into consideration. Every concurrent process that is ready to execute, can proceed forward. From this behavioral model we extract a set of tasks that capture the structure of components in the model, abstracting away computation details. The task set is then enriched with information about resource types and constraints. Each computational step of a task is annotated with required resources. If a resource that is needed by a step is shared with another task, the two tasks cannot proceed concurrently. On this model, we perform schedulability and timing analysis of the system to ensure that, for a given allocation of resources to components, timing requirements of the system are satisfied and no resource conflicts arise. If a resource conflict is found, a new resource allocation has to be used, or more resources need to be added to the system. In addition, we can perform analysis of behavioral properties of the model to ensure that functional requirements are satisfied. Once model analysis is performed to a satisfactory degree, we can proceed with the implementation of the system, driven by the resource-aware model. Automatic code

* This research was supported in part by NSF CCR-0086147, NSF CCR-0209024, ARO DAAD19-01-1-0473

generation can create a skeleton of the system implementation, while test generation provides a test suite that demonstrates the compliance of the implementation to the model.

In this paper, we limit our discussion to the resource-handling part of the framework, highlighted in Figure 1. We use a formalism that provides a formal semantical foundation for understanding resource-constrained behaviors subject to real-time constraints, memory limitations, power consumption, etc. The notion of a resource plays a central role in the specification and design of embedded systems, where execution is subject to various resource constraints, such as timing, power consumption, size and weight, etc. We feel that, in order to properly specify and analyze such systems, a modeling formalism should incorporate the notion of a resource as a first-class entity. Other aspects of the development framework are described in [2, 3, 10].

Related work in the area of resource modeling in embedded real-time systems falls into two categories. On the one hand, the importance of the issue has been long realized by practitioners and a number of model-based, albeit informal, approaches have been published. We mention [15, 17, 11, 4] among many others. On the other hand, several formal approaches have emerged that aim at scheduling of sets of tasks under constraints. For the most part, these approaches consider only timing constraints and do not introduce the notion of resource, implicitly considering the processor as the only shared resource in the system [6, 7, 9]. A different approach is taken in [1], where the authors view the scheduling activity as control and use controller synthesis techniques to model scheduled real-time systems.

In our previous work, we have proposed a family of process-algebraic formalisms for modeling and reasoning about resource-constrained systems (see [13] for an overview). The family is built around ACSR, a discrete-time process algebra. Extensions and variations include ACSR-VP that includes a value-passing capability; PACSR, a probabilistic formalism for quantitative reasoning about fault-tolerance properties; P^{2}ACSR [20] specifying power-constrained systems. The PARAGON toolset [19] provides tool support for modeling and analysis using these formalisms. The formalisms share the modeling approach, where a system is modeled as a collection of communicating concurrent processes, and offer similar modeling constructs. The difference lies in the way resources are used and the attributes they carry. For example, in PACSR resources have an attribute that captures the probability of failure for the resources. In ACSR-VP we can have tuples of data values associated with a resource that are manipulated during execution, and P^{2}ACSR has power consumption attributes. In each formalism, the set of resource attributes and the way the attributes are manipulated are slightly different. We outline a general process algebraic framework to facilitate the construction of system models that allow us to capture faithfully all of their relevant functional and non-functional requirements. The framework captures all the formalisms in the ACSR family and allows us to uniformly incorporate new features both into the formalism and PARAGON in a uniform manner.

Acknowledgements. I would like to thank Insup Lee of the University of Pennsylvania and Anna Phillippou of the University of Cyprus for fruitful collaboration in the development of the framework.

2. The Framework

We define a process-algebraic formal framework for reasoning about real-time systems. The basic entity of the framework is that of a resource. We assume that a system contains a finite set of resources. A resource is characterized by a set of attributes, such as timing parameters, priority, power consumption; probabilistic or communication behavior. Resources are partitioned into classes. All resources in a class have the same attributes. An attribute a of a class is specified as $a : \langle T : \mathsf{kind}\rangle$, where T is a basic type such as integers, characters, tuples, etc, and $\mathsf{kind} \in \{\mathsf{stat}, \mathsf{dyn}\}$ denoting whether the value of the attribute's element remains constant throughout computation (stat) or is associated to every resource use (dyn). We write a_r for the attribute a of resource r.

Example. As an example, consider the class of resources $\mathcal{R}_f$ that may experience failures, consume power, and whose use is regulated by priorities. We may characterize this class by three attributes as follows:

$$\mathcal{R}_f : [\pi : \langle [0,1] : \mathsf{stat}\rangle, pc : \langle int : \mathsf{dyn}\rangle, pr : \langle int : \mathsf{dyn}\rangle].$$

Attribute π captures the possibility of a resource failure. It has one element, that of the probability of failure, which is assumed to be constant throughout all executions of the resource. Attribute pc is the power consumption, which may be different in each resource access depending on the requested operation, and pr is the priority of the access, which may also be different depending on which process uses r.

When we use a resource r in a model we specify once the values of all the static elements of its attributes, and then, whenever r is used in the model, it is accompanied by values for all of the dynamic elements of its attributes in a tuple called *resource access*. We will always assume positional correspondence between the dynamic attribute names in the attribute tuple of a resource class and the values of attributes in the resource use. If we have resources cpu and $chan$ in the resource class $\mathcal{R}_f$, we specify initially that, for example, $\pi_{cpu} = 0.01$ and $\pi_{chan} = 0.1$. An access of the resource cpu may be $(cpu, 2.5, 1)$. We will always assume positional correspondence between the dynamic attribute names in the attribute tuple of a resource class and the values of attributes in the resource use. Therefore, in $(cpu, 2.5, 1), pc = 2.5$ and $pr = 1$. Resource accesses are specified in *actions*. An action A is a collection of resource accesses. By $\rho(A)$ we denote the names of resources used in A. Actions are building blocks for processes.

Syntax. Below, P, Q range over processes, A ranges over actions, and I ranges over sets of resources. The following grammar describes the syntax of processes and actions.

$$P \quad ::= \quad \text{NIL} \mid A : P \mid P + Q \mid P \| Q \mid [P]_I \mid P \backslash\backslash I$$

Process NIL represents the inactive process. Process $A : P$ executes action A and proceeds as P. Process $P + Q$ represents a nondeterministic choice between the two summands. Process $P \| Q$ describes the concurrent composition of P and Q: the component processes may proceed independently or interact with one another while executing actions. The construct $[P]_I$, $I \subseteq \mathcal{R}$, referred to as *resource closure*, produces a process that reserves the use of resources in I for itself, extending every action A in P with resources in $I - \rho(A)$. Finally, $P \backslash\backslash I$, referred to as *resource hiding*, allows the process to *hide* or *restrict* the identity of resources in I so that they are not visible on the interface with the environment of process P.

Example. As an example of a process, consider

$$P \overset{\text{def}}{=} \{(cpu, 3, 2), (chan, 1, 0)\} : P_1 + \{(cpu, 1, 1)\} : P_2 \ .$$

where resources cpu and $chan$ are drawn from the resource class $\mathcal{R}$ of Example 1. Process P represents a processor that can accept messages from a channel. We assume that reading the message from the channel requires additional power than remaining idle. Depending on whether the message arrives or not, P has two alternative behaviors. If the message arrives, as described in resource access $(chan, 1, 0)$, the processor may receive the message, consuming 3 units of power, and proceed to process it as P_1. Otherwise, the processor consumes 1 unit of power and continues as P_2.

Semantics. The semantics is given operationally by a transition system that captures the behavior of processes. It is based on the notion of a configuration which comprises of a process and a world/state that can be used to keep useful information about the resources of the process. We write $\mathcal{C}$ for the set of all configurations and we write $S(P) \in \mathcal{C}$, for a configuration containing process P in state S. Finally, Act is the set of all actions a configuration can engage in. The semantics is based on a function

$$F(\mathcal{C}, Act) \quad - \to \quad 2^{\mathcal{C}}$$

which, given a process configuration $S(P)$ and an action A, returns the set of configurations that can be reached from $S(P)$ by performing action A. Thus, the semantics is based on the following rule:

$$\frac{S'(P') \in F(S(P), A)}{S(P) \xrightarrow{A} S'(P')}$$

Domain specialization. In order to adjust the general framework for the needs of a specific application domain, we must give meaning for resources and their attributes and establish the semantics for the processes. The following steps perform the specialization for a particular domain: 1) Establish a finite set of resource classes for the domain, along with the attributes of the class. 2) Provide the syntactic consistency predicate $valid$. For a given action A, $valid(A)$ denotes that the action can be legitimately used in a model within this domain. Furthermore, we may restrict the domain for the set of resources I appearing in process constructs $[P]_I$ and $P \backslash\backslash I$. 3) Define the set $\mathcal{C}$ of configurations and the function F.

3. Resource classes

We consider a number of resource class definitions that are useful for modeling embedded systems. Class definitions include serially reusable shared resources, used to model processor units, communication resources, used to model synchronous and asynchronous communication channels, and multi-capacity resources that naturally correspond to memory modules. For each class of resources, we specify the set of attributes and define the semantic function $F(\mathcal{C}, Act)$ for a formalism that uses this resource class. Note that if a formalism uses multiple resource classes, the semantic function should also deal with actions that contain resources of several kinds. Functions for several such formalisms are given in [14].

Serially reusable resources. The basic kind of resources that is necessary to consider for schedulability analysis of real-time systems are serially reusable resources. While such a resource is used by a process, no other process can access it. As soon as the process releases the resource, it is immediately available for use by another process. It is a natural abstraction for a processor unit. One task is scheduled to execute on a processor, and any other task that is also ready to run is blocked until the processor becomes available. Tasks access processors according to their priorities. Different tasks can access processor resources with different priorities, therefore priority is a dynamic attribute of a serially reusable resource. We will assume here that all resources are used for one time unit. A more complex definition can be given, where a resource access has another attribute, access duration.

The resource class is defined as $\mathcal{R}_{sr} = [pr : \langle int : \mathsf{dyn} \rangle]$. The consistency predicate $valid(A)$ states than an action A is well-formed if the resources occurring in A, $\rho(A)$, are pairwise distinct. The semantic function uses the following auxiliary functions:

$$\begin{aligned} A \| B \quad &= \quad \begin{cases} A \cup B & \text{if } A, B \in \mathcal{D}_R, \ \rho(A) \cap \rho(B) = \emptyset \\ \bot & \text{otherwise} \end{cases} \\ [A]_I \quad &= \quad A \cup \{(r, 0) \mid r \in I - \rho(A)\} \end{aligned}$$

$$A\backslash\backslash J \quad = \quad \{(r,p) \in A \mid r \notin J\}$$

$A||B$ ensures that two timed actions may be composed together only if they do not compete for the use of any common resources. Otherwise, $A||B = \bot$, signifying that a deadlock arises. $[A]_I$ reserves access to resources I for the process performing A by employing all of the resources $r \in I - \rho(A)$ at priority level 0. As a result, resource in I cannot be shared with any other process (according to $A||B$). $A\backslash\backslash J$ hides the use of J-resources from timed actions. We also use the *preemption relation* on actions. Intuitively, action A_2 preempts action A_1, written $A_1 \prec A_2$, if A_2 has a resource r that has a higher priority in A_2 than in A_1, and A_1 does not have such resources. There is no need to store extra state information, so for each configuration S, $S(P) = P$.

$$\begin{array}{lll}
F(A{:}P, A) = \{P\} & & \\
R \in F(P_1 + P_2, A) & \textit{iff} & \exists i \in \{1,2\} \text{ such that } R \in F(P_i, A) \\
& & \text{and, if } F(P_{i+1}, B) \neq \emptyset \text{, then } A \not\prec B \\
R \in F(P_1||P_2, A) & \textit{iff} & [(P_1' \in F(P_1, A_1), P_2' \in F(P_2, A_2), \\
& & A = A_1||A_2, R = P_1'||P_2') \\
& & \text{and, if } F(P_1||P_2, B) \neq \emptyset \text{ then } A \not\prec B \\
R \in F([P]_I, A) & \textit{iff} & R' \in F(P, A'), R = [R']_I, A = [A']_I \\
& & \text{and, if } F([P]_I, B) \neq \emptyset \text{ then } A \not\prec [B]_I \\
R \in F(P\backslash\backslash I, A) & \textit{iff} & R' \in F(P, A'), R = R'\backslash\backslash I, A = A'\backslash\backslash I, \\
& & \text{and, if } F(P\backslash\backslash I, B) \neq \emptyset \text{ then } A \not\prec B\backslash\backslash I
\end{array}$$

Communication resources. To model synchronous communication resources, we introduce a resource class that allows us to specify input and output actions. We will use this distinction in the semantic function to define that an input and an output on the same channel can synchronize and produce an internal step. We introduce a resource class $\mathcal{R}_c = [p : \langle\{?, !, x\} : \mathsf{dyn}\rangle]$. The class has one dynamic attribute that indicates the *polarity* of resource access, distinguishing between an input action on channel a (denoted $a?$ instead of $\{(a,?)\}$), an output action on channel a, denoted $a!$, and an internal step that involves a busy channel a, denoted a_x. The consistency predicate stipulates that a communication action can contain only one resource of the class $\mathcal{R}_c$ and no other resources. The semantic definition uses the following function:

$$A||B \quad = \quad \begin{cases} a_x & \text{if } \{A, B\} = \{a?, a!\} \\ \bot, & \text{otherwise} \end{cases}$$

Thus two actions may be composed in parallel if they are send and receive actions on the same channel, and the composition results in a busy channel step. Composition of any other communication actions is undefined.

Asynchronous communication resources. To model asynchronous buffered communication between processes, we can use the following resource class. Contents of messages are not modeled, however processes can store or retrieve several messages in one step. The resource class is described as $\mathcal{R}_b = [p : \langle\{?, !\} : \mathsf{dyn}\rangle, size : \langle int : \mathsf{dyn}\rangle, c : \langle int : \mathsf{stat}\rangle]$. The static attribute c_r represents the capacity of resource r, that is, the maximum number of messages that can be stored in the buffer. In order to define the semantic function for such resources, we need to maintain state information that keeps, for each channel, the number of messages stored in the buffer of the channel. Using $S(r)$ for the number of messages in the resource r in state S, the following clause of $F(S(P), A)$ deals with communication actions: $F(S(A : P), A)) = \{S'(P)\}$ *iff* either $A = \{(r?, n)\}, S(r) = m), m - n \geq 0, S'(r) = m - n$, and $S(r') = S'(r'), r \neq r'$, or $A = \{(r!, n)\}$,$S(r) = m), m + n \leq c_r, S'(r) = m + n$, and $S(r') = S'(r'), r \neq r'$. This definition represents the natural intuition that a process can read from a channel if the channel has enough messages and can write to a channel if there are enough empty slots to write all the messages. With these definitions we can obtain a formalism similar to communicating real-time state machines of [18].

Resources with failures. For probabilistic reasoning about fault tolerance of resource-constrained systems, we have defined a class of resources $\mathcal{R}_\pi$ that can experience transient failures with a fixed probability [16, 14]. It extends $\mathcal{R}_{sr}$ with an additional attribute π of type $\langle[0,1] : \mathsf{stat}\rangle$, representing the probability of the resource failing in the next step. To be able to reason about failed as well as non-failed resources, we also have the attribute $status$ of type $\langle\{up, down\} : \mathsf{dyn}\rangle$. The process $\{(r, 1, up)\} : P$ will succeed in using r with probability π_r, or fail, becoming NIL, with probability $1 - \pi_r$. Replacing up with $down$, we get the process that fails exactly when the other one succeeds. Failed resources are useful to specify failure recovery.

Combining resource classes. When multiple resources are used in the same formalism, the definitions need to be extended to handle the case when an action contains resources of multiple classes. For example, the process algebra ACSR [12] employs serially reusable resources and a variant of synchronous communication resources. However, from the point of view of the formalism, synchronization on communication channels happens instantaneously, while access of other resources takes time. Therefore, the syntactic consistency predicate includes a provision that resources of the two classes cannot appear in the same action.

4. Analysis

We can employ a number of different techniques to analyze models in the resulting formalisms. Analysis of timing behavior for systems using serially reusable resources has been performed in [5]. More generally, similar techniques can be used to analyze functional correctness with respect to arbitrary safety properties. Schedulability analysis for a variety of scheduling algorithms has been described in [8]. Probabilistic analysis of timing and fault tolerance properties has been considered in [16]. An extension of that approach to power-aware system has been considered in [20]. Analysis techniques include model checking with respect to a temporal logic that can express power consumption constraints, as well as computation of probabilistic bounds on power consumption. We can also compute minimum and maximum power consumption over a set of executions. Power-aware scheduling has been modeled and analyzed in [20]. The resource-based approach also allows us to explore a number of design trade-offs. For example, we have considered trade-offs between performance and power consumption [20] and between performance and memory consumption [14].

5. Conclusions and future work

We have presented a formal approach to the design of real-time embedded systems, which explicitly captures resource constraints that affect the system behavior. The approach includes an extension mechanism that allows us to easily incorporate new kinds of resources and resource constraints. We defined several resource classes for commonly encountered modeling scenarios. The resource-modeling formalism is incorporated into a larger model-based development framework for embedded systems.

We are working to identify additional classes of resources and develop means of incorporating them into the formalism, as well as providing flexible tool support for the model development in the formalism. An interesting direction is to consider *consumable* resources, which can be used only a fixed amount of times during a computation and possibly replenished after a certain amount of time passes. Such an extension will allow us to have a more natural treatment of battery-operated devices.

References

[1] K. Altisen, G. Goessler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Journal of Real-Time Systems*, 23:55–84, 2002.

[2] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement for hierarchical hybrid systems. In *Proceedings of Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 33–48. Springer-Verlag, Mar. 2001.

[3] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '03)*, June 2003.

[4] L. Baum and T. Kramp. Towards a uniform modeling technique for resource-usage scenarios. In *Proceedings of PDPTA '99*, June-July 1999.

[5] H. Ben-Abdallah, I. Lee, and Y. S. Kim. The Integrated Specification and Analysis of Functional, Temporal, and Resource Requirements. In *Proceedings of Conference on Requirement Engineering*, 1997.

[6] V. Braberman and M. Felder. Verification of real-time designs: combining scheduling theory with automatic formal verification. In *Proceedings of the 7th European Engineering Conference*, pages 494–510, 1999.

[7] M. Buchholtz, J. Andersen, and H. Loevengreen. Towards a process algebra for shared processors. In *Workshop on Models for Time-Critical Systems*, BRICS Notes Series NS-01-5, pages 87–99, Aug. 2001.

[8] J.-Y. Choi, I. Lee, and H.-L. Xie. The specification and schedulability analysis of real-time systems using ACSR. In *Real-Time Systems Symposium*. IEEE Computer Society Press, December 1995.

[9] J. Ermont and F. Boniol. TPAP: an algebra of preemptive processes for verifying real-time systems with shared resources. In *Workshop on Theory and Practice of Timed Systems*, Apr. 2002.

[10] H. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage and generation. In *Proceedings of TACAS '02*, Apr. 2002.

[11] E. Huh, L. Welch, B. Shirazi, and C. Cavanaugh. Heterogeneous resource management for dynamic real-time systems. In *Heterogeneous Computing Workshop*, pages 287–296, May 2000.

[12] I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[13] I. Lee, J.-Y. Choi, H.-H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *Formal Techniques for Networked and Distributed Systems (FORTE'01)*, Aug. 2001.

[14] I. Lee, A. Philippou, and O. Sokolsky. A general resource framework for real-time systems. In *Post-workshop proceedings of the Workshop on Radical Innovations of Software and Systems Engineering in the Future*, LNCS, 2003. to appear.

[15] A. Mehra, A. Indiresan, and K. Shin. Resource management for real-time communication: Making theory meet practice. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 130–138, June 1996.

[16] A. Philippou, O. Sokolsky, R. Cleaveland, I. Lee, and S. Smolka. Probabilistic resource failure in real-time process algebra. In *Proceedings of CONCUR '98*, pages 389–404. Springer-Verlag, 1998.

[17] S. Saewong and R. Rajkumar. Cooperative scheduling of multiple resources. In *IEEE Real-Time Systems Symposium*, pages 90–101, 1999.

[18] A. Shaw. Communicating Real-Time State Machines. *IEEE Trans. on Software Eng.*, 18(9):805–816, 1992.

[19] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with PARAGON. *Annals of Software Engineering*, 7:211–234, 1999.

[20] O. Sokolsky, A. Philippou, I. Lee, and K. Christou. Modeling and analysis of power-aware systems. In *Proceedings of TACAS '03*, volume 2619 of *LNCS*, pages 409–425, Apr. 2003.