Resource Aware Modeling & Simulation (RAMS)

Paul Stoaks, CTO, Foresight Systems M&S

Preface

Resource Aware Modeling and Simulation creates a virtual prototype that allows you to visualize, and interact with, your system in action. This powerful methodology enables you to analyze performance, verify behavior and perform trade studies on architecture very early in your design cycle. This paper explains the fundamental concepts of the technology.

Introduction

Every system has two components: What it does, and what it's built on. These are common to every kind of system, from naturally occurring biological systems to business processes to complex embedded systems. These two facets work together to complete the system's mission. While they're connected, and place requirements on one another, they are distinct and separable. The systems engineering discipline works to develop both facets of the system in order to fulfill system requirements. The challenge of optimizing a system is employing minimal resources to fully meet the system requirements.

We can define these two facets of a system as follows:

Behavior: The behavior of the system is what it does in order to accomplish its objectives and fulfill its functional requirements. "Functionality" is another word for behavior in this context. The behavior of the system often consists of a collection of interconnected functional components, each of which performs some aspect of the job.

Platform: The platform is the collection of resources on which the behavior is implemented. A resource is something that supplies a good, or service, required by the behavior. The platform consists of a variety of potentially interdependent resource components.

Both the behavior and the platform can be decomposed hierarchically to manage complexity.

The behavior and platform are truly separable. Different platforms can underlie a given behavior, while the same platform can serve multiple behaviors. For example, either a *police officer* resource, or a *traffic signal* resource can implement a *manage traffic flow* behavior. Likewise, a *police officer* resource can serve either the *manage traffic flow* function or the *enforce speed limit* function. In some circumstances, the *police officer* resource might make use of a *patrol car* resource in order to accomplish the *manage traffic flow* function.

The behavior clearly places requirements on the platform for operations that must be performed. In turn, the platform places resource constraints on the behavior that ultimately determine the performance of the system.

As an example, if the system under consideration is electronic document filing, the behavior might consist of discrete steps, including *receive document*, *scan document*, *enter document properties into software form*, and *file document in database management system*. This behavior is implemented on a platform that consists of the resources *filing clerk*, *scanner device*, *database management system* and a pool of *computer workstations*. The time it takes to complete this process will be a function of the speed of the *scanner device*,

the *filing clerk*, the *computer workstation* and the underlying *database management system*. If a second *filing clerk* is added without adding a second *scanner* or *workstation*, the time required to scan documents may not be reduced at all due to the contention for these resources. Adding a second *scanner* without adding a second *file clerk* will only increase performance if the *scanner* is the bottleneck. Furthermore, we may find that the performance of the system increases over time due to the *file clerk*'s increased proficiency with the equipment and tasks. We refer to this latter improvement as a Learning Curve effect.

Of course, costs are inevitably associated with resources. The specific nature of the cost depends on the resource. In our example above, there is the initial capital cost for the equipment plus ongoing maintenance costs. A *file clerk* has an hourly cost. Other costs might include time, volume, weight, power, etc. Understanding the performance drivers for the system, and how they interrelate, is absolutely necessary to meet our overriding objective of building an optimal system (minimizing cost while meeting all of the requirements).

RAMS is a powerful tool to help you design and build fully optimized systems. Users have employed the software for performance analysis and design to achieve this objective across a wide variety of systems engineering applications, including business processes, embedded systems, avionics, web services, and software development.

What is Resource Aware Modeling and Simulation?

RAMS is a methodology for performance analysis and system optimization that uses simulatable models to leverage the distinction between behavior and platform. The first step of a RAMS process is using a Modeling Language to create models of both the behavior and platform. Next, the behavioral model is mapped to the platform model. Finally, we simulate the resulting composite model to evaluate the performance and behavior of the system. We'll explore this process in more detail below.

Modeling the Behavior

We model the behavior of the system using traditional functional modeling techniques. This aspect of the system usually consists of an interconnected network of functional components that combine to provide the complete behavior of the system. This network can generally be decomposed hierarchically in order to manage complexity. The behavior of the system includes:

- Data flow
- Control flow
- Functional component behavior (i.e. state machine, data transformation, hand-shaking, etc.)

A complete behavioral model can be simulated without a platform model in order to functionally verify the model and system. This confirms that the behavior will produce the correct output, given the necessary inputs, and has been the primary value of traditional functional modeling and simulation.

Modeling the Platform

The platform is modeled by assembling representations for the resources required to implement the behavior. Resources can include people, electronic components, networks, software services, power, space, etc. Two fundamental types of resources are considered:

Process Resources perform work of some nature. They are characterized by the rate at which they accomplish work. When a request is made of a Process Resource, that request will take time to fulfill. The rate may be variable and a Process Resource may have interdependent

relationships with other resources. Resources may also have strategies for dealing with contention. For instance, when you order a sandwich at the deli counter, some waiting is often involved. The person who makes the sandwiches must become available and then construct the sandwich. The sandwich maker may have strategies for handling a long line, such as priority service for important customers. The sandwich maker is an example of a Process Resource.

Quantity Resources represent a quantity of something. They are characterized by a value, which indicates their capacity. When a request is made of a Quantity Resource, that request will either succeed, thereby reducing the current quantity of the resource by the requested amount, or fail, leaving the quantity unchanged. If the request succeeds, the requestor can continue and, if appropriate, return the requested amount when finished. If the request fails, the requestor can choose to wait until the necessary resource becomes available, request a lesser amount, or continue without. Computer memory is a common example of a Quantity Resource. Programs make requests for memory, which either succeed or fail, and return memory to the pool when it is no longer needed.

Platform models are made up of a combination of Process and Quantity resources. In our document processing example, the *file clerk*, *scanner* and *DBMS* might be modeled as process resources while a pool of *workstations* might be modeled as a quantity resource. (The modeling of a pool of *workstations* as a quantity resource may not be immediately obvious. The rationale is that there is a fixed number of workstations and there may be more than one potential user. A given workstation may be used by only one user at a time.)

Platform models are often layered, as are the real world architectures that they represent. For instance, in an embedded electronic system, the behavior may rely on an Operating System scheduler resource, which in turn relies on a microprocessor resource, which relies on a bus resource. Any request to the OS resource will result in a cascade of requests down to the bus resource in order to accomplish the work. In our document processing example above, the *scan the document* step makes a request of the *filing clerk* resource which in turn utilizes the *scanner device* resource to complete the task.

The embedded system example demonstrates another characteristic of resources. A resource may be merely an arbiter of requests to a another, shared, resource and do no work on its own (like the OS scheduler). The arbiter resource will behave like the resource that it is controlling access to (either a Process Resource or a Quantity Resource).

The resources in platform models can communicate with each other to accomplish the work required. This usually occurs if the behavior of one resource depends on the state of another resource. However, care should be taken not to model data or control flow in the platform model. This information belongs in the behavioral model. There may be resources in the platform model that enable the flow (such as a bus or network), but the flow itself should occur in the behavioral model.

Both Process Resources and Quantity Resources can manifest complex behaviors. Thus, the creation of custom resources that accurately represent their real-world counterparts is required. Some examples of sophisticated resource functions include task scheduling, learning curves, arbitration, reliance on other resources in complex ways or providing additional computational functionality (such as cost accrual).

The Composite Model

Mapping the Behavioral Model to the Platform Model creates the composite model. Resource requests from the functions in the behavioral model to named resources in the platform model establish this correspondence.

The resource requests include the name of the resource, the amount of work (for Process Resources) or requested amount (for Quantity Resources) as well as any additional information required to satisfy the request. The ease of making changes to the mapping within the Foresight environment enables trade studies and portability of the behavioral model across platform models.

Simulation of the composite model exposes a number of characteristics of the system for analysis, including:

- **Temporal behavior of the system** — The resource requests implicitly insert delays into the behavioral model, thereby revealing the time-based performance of the system. This allows the examination of latency, jitter, end-to-end processing time, etc. In addition, missed deadlines and other scheduling issues can be immediately identified.

- **Resource utilization** — The utilization of each of the resources in the platform is available for inspection.

- **Cost** — The resources can directly accumulate cost for straightforward cost analysis. (Again, cost can be $$, power, time, etc.)

This ability to fully understand each of these dimensions of the system makes it possible to optimize against the original requirements. Bottlenecks, heavy users, and other problem areas are easily identified, allowing the design team to precisely target the most important areas for improvement. Trade studies can be used to evaluate alternative architectures (in the behavioral design, the platform design, or both) in order to identify the best solution to any deviations from the requirements.

Foresight's Approach

The Foresight systems engineering environment has featured RAMS capability for more than fifteen years. This powerful tool represents a key advantage for the Foresight solution over competitive systems modeling environments and has been employed by our customers for a wide variety of systems designs.

Both the behavioral and platform models are built with the tool's powerful graphical system modeling language. The modeling language is natural and easy to learn. This eminently approachable environment enables fast creation of models that can be simulated at any arbitrary level of abstraction. The combination of easily constructed models and powerful analysis capabilities provides a rapid prototyping approach to modeling and allows designers to receive useful feedback with minimal initial investment. The models can then be further developed through progressive refinement.

The environment includes a general-purpose parameterization mechanism that enables the creation of highly configurable models. This mechanism is used to specify the resource mapping (resource name, request amount, and other characteristics) making trade studies very easy to set up and execute.

Foresight's simulation, graphical visualization, and highly customizable data logging capabilities make analysis of the behavioral and composite models straightforward and readily accessible. Resource utilization can be viewed directly as the model executes. Customizable data logging allows for detailed information mining and performance analysis. The Foresight Visualizer tool makes examining resource behavior and contention nearly effortless by graphically displaying all clients of a resource and their state (whether they are waiting or executing) at any point in time.

Foresight's library includes everything necessary for RAMS design and analysis. An easy to use, general-purpose Process Resource is included. This Resource flexibly models many different kinds of behaviors, including pools of resources, resources with round-robin, priority-preemptive, and priority-nonpreemptive scheduling. A

general-purpose Quantity Resource, called the Basic Resource, is also included. The Basic Resource models many kinds of Quantity Resource behavior and provides the caller with both blocking and non-blocking capabilities. For situations where custom resource behavior is required, the Foresight modeling language supports the necessary building blocks to create arbitrarily complex User Defined Resources.

Foresight's User Defined Resource capability has been used to construct resources that model a broad range of real systems, including:

- OS scheduling behavior, including the Integrity RTOS and Linux

- CORBA ORB behavior

- Various types of software services

- Multiple on-chip and processor data busses

- Network interconnect and protocols including Ethernet, RapidIO, SONET, FibreChannel, etc.

- Memory and file systems

- Algorithmic blocks, including CODECs

- Human resources (including learning curve behavior)

- And many, many more…

Summary

Foresight's RAMS provides unprecedented insight and flexibility by giving you a working virtual prototype that allows you to visualize your system in action. Explicitly separating your system's behavior from its platform enables more effective trade studies and investigation. These powerful analysis tools allow you to look ahead and optimize implementation decisions during system design, with full awareness of your platform resources. Foresight's "real world" modeling gives you the ability to fully verify both the functionality and the actual performance of your system before you commit to a hard implementation. Anticipating system tradeoffs early allows you to manage complexity, provides additional flexibility, compresses product cycles and minimizes execution risk.

About the Author:

Paul Stoaks is a founder of Foresight Systems M&S and currently serves as the company's Chief Technology Officer. He has spent the last twenty years developing software tools that enable engineers to reach the next level in design productivity and system capability. His extensive engineering background spans from developing of IC layout tools at Mentor Graphics to solving real system design challenges for the Joint Tactical Radio System (JTRS) program. Paul has gained a keen appreciation for the problems facing systems designers building complex high performance systems. Along the way, he has developed proven strategies for reducing risk and ensuring project success. Paul is an active member of INCOSE and a frequent contributor to systems engineering discussions across a number of forums. Email: paul@foresight-mands.com

An Example

The Foresight whitepaper *Digital Imaging with ARM*[1] discusses a system design problem that illustrates the RAMS approach. In this case study, the objective is to develop a digital camera System on Chip (SoC) that meets certain performance requirements while minimizing cost.

Behavioral Model

The example model focuses on the image processing functionality of the system. The Foresight behavioral model for the image processing subsystem is shown in Figure 1, below.
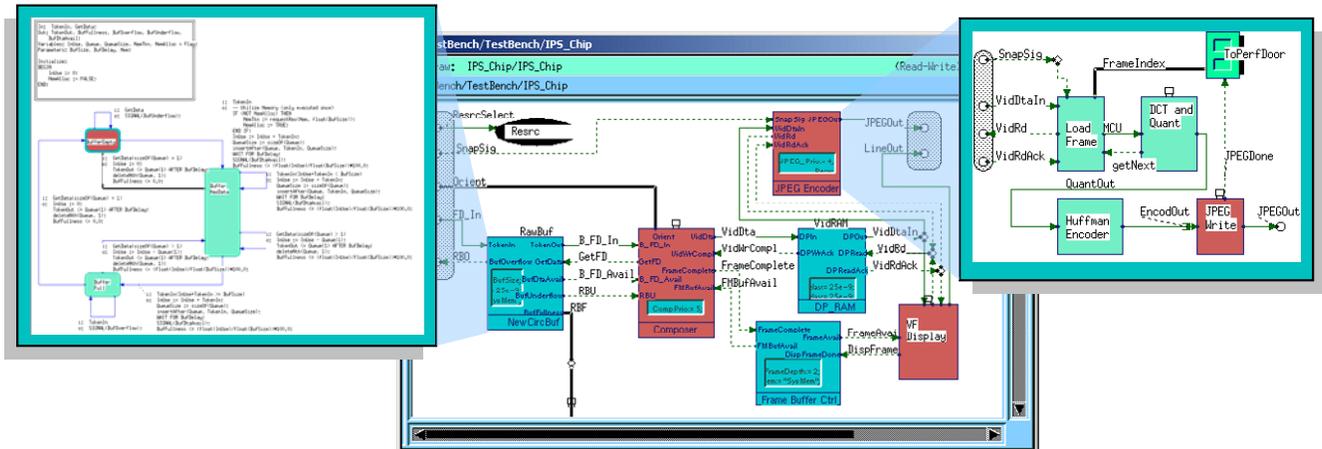

Figure 1: Foresight Behavior Model for a Digital Camera SOC

The center Data Flow Diagram (DFD) is the top-level diagram for the subsystem. It consists of an input buffer, a "Composer" component that performs various image preprocessing, a Frame Buffer, a viewfinder display, and a JPEG encoder. The state machine on the left is the definition of the input buffer functional component. The DFD on the right is the definition of the JPEG Encoder functional component.

This is the Behavioral Model of the system and it is simulatable by itself. In this particular example, the implementation of the JPEG encoder is actually connected into the model so that the behavior of the algorithm can be verified in the context of the system model.

Platform Model

The platform for the system is shown in Figure 2, below.

---

[1] The URL for the Foresight white paper, "Digital Imaging with ARM" is:

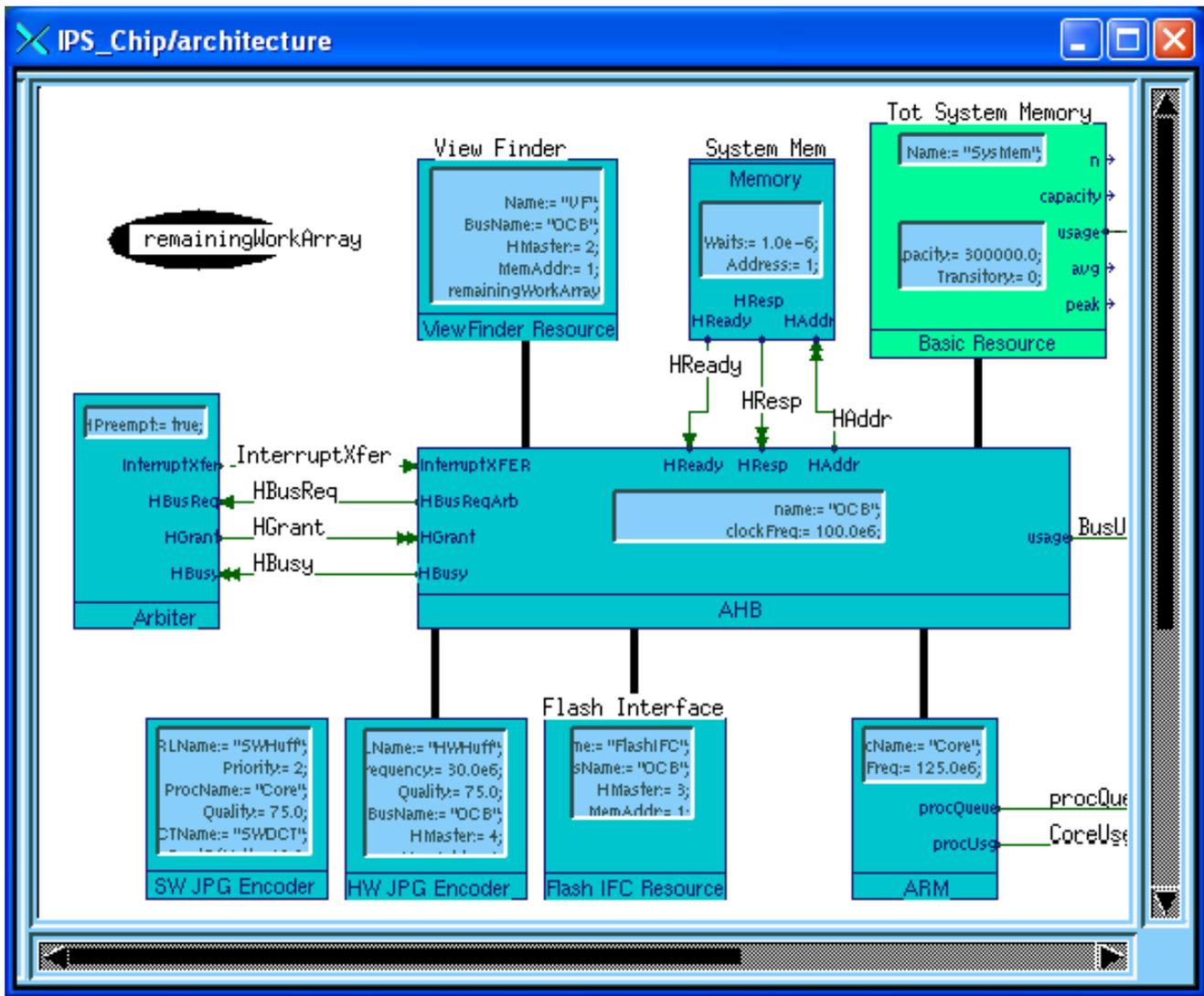http://www.foresightsystems-mands.com/php/docAccess.php?reqDoc=Digital_Imaging_with_ARM.pdf

Figure 2: Foresight Platform Model for a Digital Camera SOC

This is one of many possible implementation platforms for the camera. This platform model is designed in such a way that it can actually represent several different variations of the platform under parametric control. It consists of a general purpose processor (an ARM9 microcontroller), on-chip processor bus (the ARM AHB bus and associated arbiter), RAM, viewfinder, flash memory interface, and both hardware and software implementations of the JPEG CODEC. These resources are parameterized for clock speed and the names of other resources that they depend on as well as other configuration information. (Note that the bus-connected resources all rely on the bus, and the software JPEG CODEC relies on the ARM processor.)

Composite Model

The composite model consists of the mapping of the Behavioral Model to the Platform Model. In order to support trade studies, the model is parameterized to support a variety of mappings. One mapping of the JPEG encoder is shown notionally in Figure 3, below.
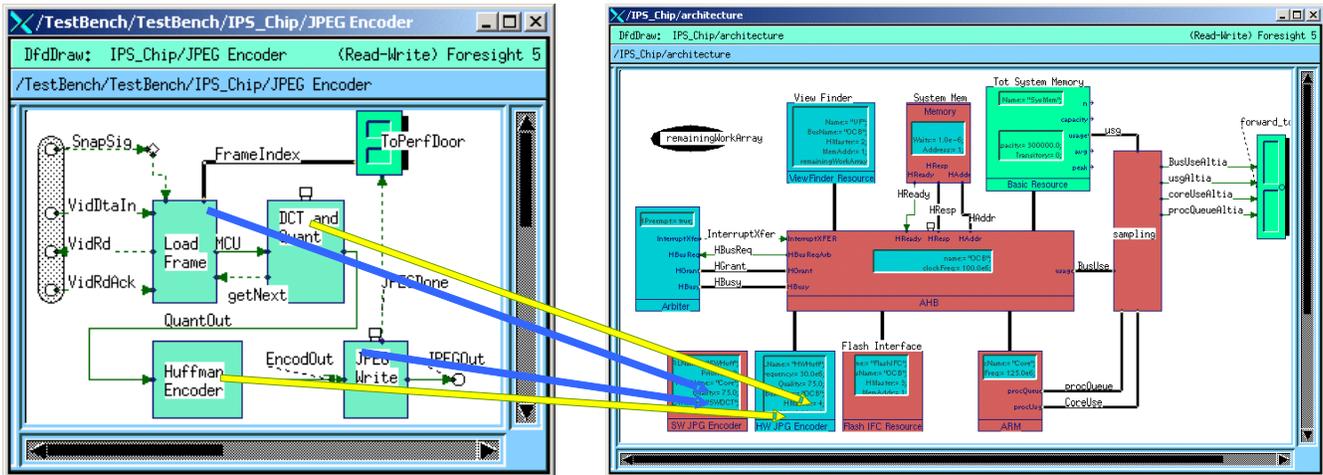
Figure 3: Foresight Composite Model for a Digital Camera SOC

In this mapping, the two most compute-intensive functions in the JPEG encoder (DCT and Huffman encoding) are mapped to the hardware CODEC. The *Load Frame* and *JPEG Write* functions are mapped to the software encoder.

Simulation and Analysis

The objective of the analysis is to determine the best implementation for the JPEG encoder. This is accomplished by simulating three different configurations of the composite model that have different mappings of the JPEG encoder components to the platform. By simply flipping a switch to select each of the different mappings, the performance of the system against the requirements can quickly and easily be evaluated. In addition, processor and bus utilization can be predicted. This analysis makes system optimization extremely straightforward.

In this particular system, an all software implementation of the JPEG encoder did not meet the performance requirements for the system due to the GPP becoming overloaded. An all hardware implementation exceeded the performance requirements, but a mixed implementation (the mapping described above) also met the performance requirements and required significantly fewer hardware gates to implement. This last configuration was chosen because it maximized performance while minimizing cost.

The optimization of this system design was greatly facilitated by the RAMS approach. The tools allowed designers to:

- Easily trade off among implementation alternatives without significantly modifying the model.

- Accurately predict resource utilization for memory, bus and GPP for each candidate configuration.

- Accurately predict key performance metrics for each candidate configuration.

- Verify implementation directly in the context of the system model.