SPE for the Internet of Things and Other Real-Time Embedded Systems

Connie U. Smith
Performance Engineering Services
PO Box 2640
Santa Fe, NM 87504-2640 USA
www.spe-ed.com

Catalina M. Lladó
Computing and Maths Department
Universitat de les Illes Balears
07071 Palma de Mallorca, Spain
cllado@uib.cat

ABSTRACT

When real-time embedded systems fail: patients die, warships shoot passenger jets, airplanes crash, cars stop on freeways or accelerate uncontrollably, and other documented problems. Preventing these problems saves lives, money, enables faster delivery, improves architectures, and improves performance. Performance engineering enables developers to predict performance, identify, and correct problems before products are built that contain serious potential failures.

This paper examines current technical and performance issues in real-time embedded systems (RTES) including software and systems developed for the Internet of Things (IoT). We describe the model interoperability framework that uses Model Interchange Formats (MIFs) to exchange performance models among modeling tools. Performance models for RTES or IoT require the representation of additional features, and solution methods beyond efficient, exact model solutions. We introduce the extensions then describe the extended meta-model for the model interoperability framework. We conclude with an evaluation of the approach and how it can be used for performance evaluation of RTES and IoT. While our work specifically focused on RTES and IoT features, the results are applicable to the performance evaluation of many different types of systems.

Keywords
Interchange Format, Real-time embedded systems, Model Driven Performance Engineering, SPE, Tool interoperability.

1. INTRODUCTION

Real-time embedded systems (RTES) “monitor, respond to, or control an external environment. This environment is connected to the computer system through sensors, actuators, and other input-output interfaces.” The Internet of Things (IoT) is “the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.” Performance issues for these objects and systems include:

- Individual actions must meet often-strict performance requirements
- The latency from the time a request arrives until the final action is complete must be responsive to users
- Sizing of hardware devices and components influences cost and thus competitiveness in the market
- Networks of devices and systems collect and consolidate data; the number of sources and amount of data to be processed requires dramatically increasing speed and capacity.

SPE models have been demonstrated to be valuable in predicting performance of software and systems [22, 23]. RTES and IoT need models that assess both high-level system design issues as well as models that evaluate more detailed implementation options. They often involve hardware/software co-design [17]. This work extends the software performance model technology previously reported [22], to include new modeling primitives needed to evaluate these types of systems. While our work specifically focused on RTES and IoT features, the results are applicable to the performance evaluation of many different types of systems.

This work is based on the model interoperability framework previously reported in [16]. The core of the framework is a model interchange format (MIF) that is a common representation for data required by performance modeling tools. Using the MIF, tools in the framework may exchange models by implementing an import/export mechanism and need not be adapted to interact with every other tool in the framework. In fact, they need not know of the existence of other tools thus facilitating the addition of new tools. Our work uses two MIFs: the Performance Model Interchange Format (PMIF) and the Software Performance Model Interchange Format (S-PMIF) [19, 9, 14, 11].

A Model Interchange Format (MIF) for VLSI designs (EDIF) [1] followed by a MIF for Case Data Interchange Format (CDIF) [2] for software design interchange (based on EDIF) were proposed in the 1980s. The Software Performance Model Interchange Format (S-PMIF) and system Performance Model Interchange Format (PMIF) adaptations were proposed in the 1990s [21, 24]. In 2004 we introduced PMIF2, implemented it using XML, and established proof of concept [19]. Revisions to S-PMIF first updated and imple-
mented with XML [14], then later added features for component based and real time systems (S-PMIF2) [11].

Other model interchange formats and model representation extensions have since been proposed. Examples include:

- UML2 - Unified Modeling Language (UML) is a language for modeling software. Its sequence diagrams and activity diagrams represent software processing steps in different formats [7].

UML may be used as the source for performance model evaluation, or it may be supplemented with MARTE described next.

- MARTE - Modeling and Analysis of Real Time Embedded Systems is a language that adds specifications to UML for model-based design and analysis of real time and embedded systems. Performance Analysis Model (PAM) is the subprofile of MARTE that supports early analysis of performance [6, 12, 10].

- CSM - Core Scenario Model, notation developed for functional analysis and preliminary performance analysis of software systems [26].

- Performance by Unified Model Analysis (PUMA) - a framework into which different kinds of software design tools (UML based) can be plugged as sources and different kinds of performance tools can be plugged as targets, using CSM [25].

- Palladio Component Model (PCM) - for representing software architecture with respect to structure, behavior, resource usage, execution environment and usage profile [3, 8].

- KLAPER - Kernel Language for Performance and Reliability analysis of component based models [5].

- EX-SE - Experiment Schema Extension for specifying the experiments to run with the models [16].

It would be possible to develop a unified superset of these meta-models; however, the resulting meta-model would not be as closely tied to individual modeling paradigms. It seems better to use M2M transformations among them in a model interoperability framework [16]. We used these meta-models in determining features to be included in our MIF extensions. An extensive discussion of this and other related work is covered in [15, 4].

The next section presents extensions to S-PMIF that incorporate modeling features useful for modeling RTES and IoT.

2. S-PMIF+

2.1 Extensions

Our initial MIFs were restricted to Queueing Network Models (QNM) that can be solved by efficient, exact solution algorithms. This scope let us explore the end-to-end process of creating models, exchanging them among multiple tools, running experiments, and comparing solutions. The MIFs and the overall model interoperability approach have been demonstrated to be viable. These extensions broaden the scope to support performance models that can be solved with additional methods such as analytical approximations or simulation solutions. We add a plus sign to distinguish this version (S-PMIF+).

The extensions included in S-PMIF+ are:

1. Wait/Queue/Set Event - An Event may be Set or Cleared. Workloads may Wait or Queue for an event to be Set. When an event is Set, all waiting workloads and one queued workload may proceed.

2. Allocate/Deallocate Resource - When access to a passive resource is restricted, a workload may request access and wait in a queue until the resource is Allocated. When access to the resource is no longer needed the workload Deallocates the resource. A scheduling policy determines the next workload to receive the Allocation.

3. Request/Release/Create/Destroy Token - A Token is a special type of passive resource. In addition to Allocate/Deallocate, it is possible to dynamically Create and Destroy tokens.

4. Get/Put Buffer - A Buffer is another special type of passive resource, with a specified initial size (therefore it uses Get/Put operations instead of Allocate/Deallocate). Get requests the specified quantity from the Buffer and waits until it is available. Put adds the specified quantity to the Buffer and waits if there is insufficient space.

5. Read/Write Shared Variable - A workload can either read or write a variable which is shared with other workloads. It allows multiple readers but writers must have exclusive access.

6. Send/Receive Message - A mailbox is a container for holding messages. A workload can Send a message to a mailbox. A workload can Receive a message from a mailbox; if the mailbox is empty, the workload waits until the next message is sent to that mailbox.

7. Call/Accept/Return Synchronization Point - A workload may Call another workload and wait for the called workload to signal that it has completed the request; the called workload Accepts the request and later Returns to the waiting workload.

8. Allocate/Deallocate/Add Memory - Memory is a special kind of resource, with an initial quantity. A workload can request allocation of a specific amount of memory and may queue if it is not available. Allocate requests a specified amount of a memory, the workload must wait if it is not available. DEALLOCATE releases the specified amount of the specified memory; the waiting workload(s) that will fit are allocated, but a lower priority workload cannot go ahead of a higher priority one. Add increases the amount of a specified Memory.

9. Fork/Split/Join Workload - A workload may Fork or Split into one or more child workloads that execute concurrently. Forked workloads later Join; the parent workload waits until all child workloads Join, then the parent workload resumes execution. Split workloads do not join, they eventually complete and leave the system.
10. Phase Change - A workload may have distinct execution characteristics such as routing, resource consumption, or passive resource usage. A Phase identifier distinguishes the behavior specifications; phases may Change at specific execution points, and execution output metrics may be associated with Phases.

11. Priority - Workloads may have a Priority that controls queue scheduling. A higher priority workload is ahead of a lower priority one. For equal priorities the scheduling is usually first-come, first-served. Priorities may be changed during execution.

12. Arrival and Service Distributions - a broader set of stochastic distributions can be used when solving models with simulation and approximation methods including: exponential, hyperexponential, uniform, normal, erlang, constant, and a generic other.

13. Queue Scheduling Disciplines - additional disciplines can be used to determine the next workload selected from a queue: to FCFS, IS, PS, RR, LCFS, FCFSPR, FCFSPPR, RRP, and LCFSPPR.

2.2 Meta-model

The S-PMIF meta-model is in Figure 1. In the following discussion new or changed features are indicated with (R).

Software performance is represented with a Project that is composed of one or more Scenarios, one or more Facilities, and zero or more Passive Entities (R). The specification of a ProbabilityDistribution (R) different from Exponential (which is the default) is allowed for interarrival specifications for PerformanceScenarios (R) and for serviceTime of ActiveService (R).

Nodes may be either Processing Nodes or Compound Nodes. CriticalSection (R), a new SyncNode, represents processing that must be uninterrupted, i.e., it is guarded with a semaphore. Other nodes are unchanged from earlier versions of S-PMIF.

There are some revisions to S-PMIF to better represent the OverheadMatrix and its parts. The ServiceSpec (R) shows three types of service: ActiveService (R), CalculatedService (R), and PassiveService (R). The CalculatedService makes explicit that it is computed from the SWResourceRequirement and OverheadMatrix. This is the typical ServiceSpec for software performance models. The ActiveService and PassiveService are added to correspond to the PMIF+ ServiceRequestPlus [9]. They specify a combination of active and passive service requests that can be made, with an optional sequenceNumber which specifies the order of execution when ordering is required. ActiveService requests specify a computer resource requirement for a specified server. They may use a special ProbabilityDistribution (R) or a load dependent service time (R). The latter is specified as a string that will be interpreted by the tool. PassiveService requests specify the command, the quantity, and reference the PassiveEntity (R). The specifications in Table 2.2 are included in S-PMIF+

Queue scheduling disciplines are extended (R) as above. Other parts of the meta-model are unchanged so they are not explained here.

The S-PMIF meta-model allows for easily adding additional communication and coordination model features. If additional needs are identified in the future, a new PassiveEntity can be added along with its PassiveService commands. Other parts of the meta-model are unaffected.

3. EVALUATION

We have validated S-PMIF meta-model by constructing test models of all features, then creating and solving those models in two very different modeling tools. Qnap [13] and a new prototype tool with a working title RTES Analyzer created from the SPE-ED [23] solver [20] to handle the S-PMIF+ extensions.

We have also created and solved more complex models of actual systems that use most of the new features in combination, and compared simulation results. We confirmed that it is feasible to represent and solve all the included features, that we have defined the features correctly with all necessary data specified, and that it is feasible to automatically translate models that conform to the meta-models into different modeling tools.

It is beyond the scope of this paper to cover all these examples. Details may be found in [18]. Instead, we selected one example that requires a solution with analytical approximation or simulation and illustrates passive resources and other model features.

The example has a pipe and filter architectural style. Data arrives from an external source at a constant arrival rate of 1 unit per second, it is processed by the first (open) workload, Get, then put in a buffer. The second (closed) workload, Spatial, after a thinkTime of 0, begins with a get from that buffer, when the data arrives Spatial processes it, then puts it to another buffer, and the cycle repeats. Three other “downstream” closed workloads, Temporal, Threshold, Paths, do the same. Each workload executes on its own processor so the workloads can execute in parallel. Workloads may have to wait on a get for data to arrive in the buffer, or at a put if there is no space in the buffer because a downstream process has not yet processed earlier data.

Each buffer is declared as a PassiveEntity. To specify buffer access, the ServiceSpec for the scenario at its CPU specifies:

- a PassiveService with the command “get” and a reference to the buffer (which may cause a wait if the data has not arrived)
- an ActiveService for its CPU service (with normal service distribution)
- a PassiveService with the command “put” and reference to the appropriate buffer (which may release a waiting workload, or cause a wait if the buffer is full).

The following shows an excerpt of the xmi specification with a ServiceSpec for one of the scenarios at its CPU:

```xml
<scenarios xsi:type="spmif:PerformanceScenario" name="Spatial">
  <mainEG>
    <startNode xsi:type="spmif:BasicNode" name="Spatial" isMainEG="true">
      <nodes xsi:type="spmif:BasicNode" name="SpatialNode">
        <serviceReq xsi:type="spmif:PassiveService" executesOn="/@server.0" sequenceNumber="1"/>

      </nodes>
    </startNode>
  </mainEG>
</scenarios>
```

Note that the ServiceSpecs are not necessarily in order thus the requirement for sequenceNumber attribute
### Table 1: PassiveEntity options with their associated commands

<table>
<thead>
<tr>
<th>Passive Entity</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>start/stop</td>
</tr>
<tr>
<td>mailbox</td>
<td>send/receive</td>
</tr>
<tr>
<td>resource</td>
<td>allocate/deallocate</td>
</tr>
<tr>
<td>token</td>
<td>wait/queue/create/destroy</td>
</tr>
<tr>
<td>event</td>
<td>wait/queue/set/clear</td>
</tr>
<tr>
<td>buffer</td>
<td>get/put/create/destroy</td>
</tr>
<tr>
<td>sharedvar</td>
<td>read/write</td>
</tr>
<tr>
<td>memory</td>
<td>allocate/deallocate/add</td>
</tr>
<tr>
<td>syncpoint</td>
<td>callreturn/accept/return</td>
</tr>
</tbody>
</table>

The Buffer implementation in Qnap is easily done with Semaphores, which consist of a queue and a counter. The counter is the number of pass grants available if positive, and the number of customers waiting if negative. The workload that puts data to the buffer produces a pass grant for the semaphore, and the workload that gets data from the buffer asks the semaphore for a pass grant and it waits if the value of the counter is ≤ 0.

The implementation in RTES/Analyzer is also easy. Its simulation engine, CSIM, has a buffer type that is declared and a size specified; operations sendRequest and receiveRequest have the desired semantics and CSIM manages the waiting/activation of processes when appropriate. Both constant and normal probability distributions (and others) are supported by CSIM.

The results are shown in Table 2. Because of the constant arrival rate, and normal service time distribution, the results are identical except for the precision reported thus confirming that the models have been correctly implemented.

This example illustrates using a fixed-size buffer for sending data between processes which includes requesting a passive resource, possibly waiting, then releasing it which may schedule another waiting workload. It uses two different ProbabilityDistributions, and buffers for synchronization between processes, which is supported differently in the two tools. A similar technique is used for most of the PassiveEntity/Command combinations in Table 1. The timer for recording end-to-end response time, and syncpoint for synchronization of concurrent processes have different behavior [18].

### 4. CONCLUSIONS

This paper describes performance modeling challenges of RTES and IoT. Additional types of evaluations are needed, such as end-to-end latency and hardware/software codesign. Extensions that model additional types of communication and coordination that require new types of model solutions were also described. The S-PMIF+ meta-model was presented and explained. We then described our validation process. We have used these model features to represent actual RTES and IoT case studies and found the modeling power to be sufficient to evaluate their system performance.

After years of creating test models and comparing results, we are confident that:

- Key performance-determining features are included in the MIF
- They are correctly represented with the meta-model
- The behavior of the performance models corresponds to the meta-model
- The model solutions are correct

The extended S-PMIF+ is ready to be used by other researchers and practitioners.

### Acknowledgments.

Smith’s participation was sponsored by US Air Force Contract FA8750-15-C-0171.

### 5. REFERENCES

[1] EDIF, Electronic design interchange format. en.wikipedia.org/wiki/EDIF.
<table>
<thead>
<tr>
<th>Example</th>
<th>Latency</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qnap</td>
<td>0.486</td>
<td>0.0869</td>
</tr>
<tr>
<td>RTES/Analyzer</td>
<td>0.486</td>
<td>0.087</td>
</tr>
</tbody>
</table>

Table 2: Buffer Results


Figure 1: S-PMIF+ Meta-model