Constructing Resource Usage Models of a Large and Complex Software-Intensive System

Trosky B. Callo Arias[1], Pierre America[2], and Paris Avgeriou[1]
[1]*Department of Mathematics and Computing Science - University of Groningen*
[2]*Philips Research and Embedded Systems Institute*
The Netherlands
trosky@cs.rug.nl, pierre.america@philips.com, paris@cs.rug.nl

Abstract

Resource usage models are important asset to analyze and ensure the adequate usage of the system platform resources such as processors and memory elements. In this paper, we present how to construct resource usage models using actual execution information of a large software-intensive system. We have constructed this type of models for an MRI system (a representative large software-intensive system) to describe resource usage information already using system specific elements such as execution workflow, system software components, and processes. Our observations point out that these models provide useful insights and overviews to support the identification and evaluation of the adequate use of the system platform resources in a top-down fashion, which is desired when developing a large and complex software-intensive system.

1. Introduction

Typical resources that need to be considered when developing and maintaining a software system are network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory. Often, an inappropriate or unpredicted usage of these resources can compromise non-functional properties (e.g., performance and reliability), triggering the execution of expensive corrective maintenance, and even redesign activities. In the literature, there are various methods and approached that use resource usage information to predict and analyze performance and reliability issues. Some of these methods use resource usage information on design models and specifications [2, 6] and others rely on measurements from an existing system [4].

As part of our research on the evolvability of large software-intensive systems [7], we observed that the adoption of prediction methods and approaches using resource usage information is not a common practice, especially when resource usage information is unavailable, hard to describe, or inaccurate because the system at hand is an existing large and complex software-intensive system (composed of multiple processes with multiple threads and deployed across several computers with particular resource configurations). Nevertheless, we also observed that measurement-based descriptions of resource usage is often use in practice and supported by various tools [LTTng, xPerf, SysInternals]. However, these measurement-base descriptions are often generic and low-level of detail. Such descriptions are not immediately useful for practitioners developing and maintaining a large and complex software-intensive system following a top-down or architecture driven approach.

It is obvious that is necessary to describe the resource usage of a software system, especially in a top down fashion (providing insights and overviews) to manage complexity when dealing with large software-intensive systems. Thus our focus is to support practitioners in how to construct resource usage models that describe the actual system resource usage in terms of their specific software system. This includes what and how to measure, how to describe or present the measurement to provide insights and overviews.

In this paper, we present how to construct resource usage models customizing a dynamic analysis approach presented in our previous work [3]. We are using this customization to create resource usage models for the software of a Magnetic Resonance Imaging (MRI) system, a representative large and complex software-intensive system developed by Philips Healthcare [1]. Our observations and findings, from using these models in practice, show that these models support the identification and evaluation of opportunities to improve and

ensure an adequate use of the system platform resources in a top-down fashion.

The organization of the rest of this paper is as follows. In Section 2, we briefly describe the dynamic analysis approach and summarize its customization to construct resource usage models. In Section 3, we describe how to identify the required input and process it to construct resource usage models. Section 4 we present the resource usage models constructed with our approach. In Section 5, we describe our observations and findings from using resource usage models in practice. Finally, in Section 6, we provide some conclusions and future work.

2. Customized dynamic analysis approach

In our previous work [3], we presented a dynamic analysis approach to construct execution models of a large software-intensive system. This approach is an iterative process that allows us to cope with complexity providing means to collect and analyze high-level information first, and then dig down for details when needed. This has been probed useful especially to support top-down and architecture-driven activities to analyze the runtime of the software of a large software-intensive system. Figure 1 illustrates the main elements of our approach customized for the construction of resource usage models and which we describe in the rest this section.

2.1. Resource usage viewpoint

By definition, a viewpoint addresses particular concerns of the system stakeholders and consists of the conventions for the construction, interpretation, and use of an architectural view [5]. We have identified a set of viewpoint interacting with key practitioners of our industrial partner that includes a set of organization and system specific concerns and guidelines. We are including this set of viewpoints as part of the overall process of our approach, especially to focus and ease the identification of requirements and reduce the iteration phases when constructing specific types of models.

In this case we use a resource usage viewpoint (see "a" in Figure1) summarized in Table 1. In general the concern of this viewpoint is to ensure an adequate use of the system platform resources (e.g., CPU, memory, and cache) at the runtime of system. Within our approach, this viewpoint serves as a guideline to ease and scope the interaction with practitioners: communicate what execution concepts (elements and relationships) build a resource usage model, initial examples of this kind of models, define the requirements and inputs (selection of suitable execution scenarios, required stakeholders, techniques to collect execution data) to construct them.

Table 1. Summary of resource usage viewpoint

Concerns	Identification of bottlenecks, delays, definition of metrics, benchmarks, and budgets.
Models	Scenario-based resources (i.e. processor, memory, and network) usage models, budgets, predictions.
Stakeholders	Software architects, designers, testers, and system platform supporters,
Development activities	System understanding, analysis of alternative design and implementation, testing and conformance of design and implementation, corrective maintenance, and tuning of nonfunctional properties.

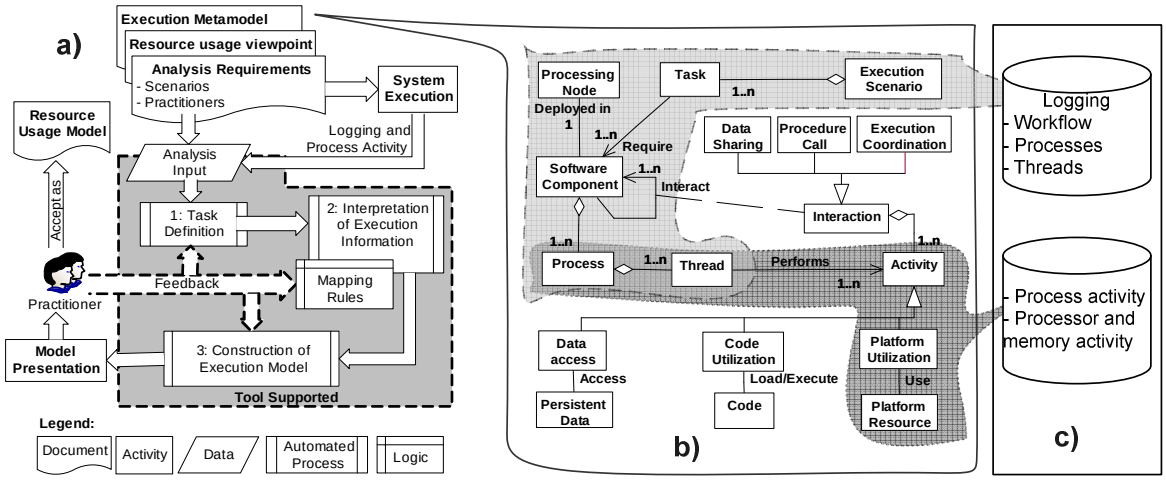

Figure 1. Customized approach for the construction of resource usage models

2.2. Execution Metamodel

Our approach is supported for by a specific metamodel that describe the various elements and relationships that play a role within runtime of a software system (see "b" in Figure 1). We introduced this metamodel in our previous work [3] and progressively extended it with the development of our research. In particular, to support the construction of resource usage models, we have included the concept of *processing node*. This concept represents the hardware devices, e.g., computers, in which software components are deployed. Processing nodes contain resources such as CPUs and memory that form part of the system platform resources that a software component use through the activity performed by its respective threads and processes.

In overall, the various elements in the metamodel play a role at the runtime of a software system and define a number of concepts described in an execution model. In terms of these elements, we can describe that a resource usage model, constructed with our approach, describes in general how the system software components use the platform resources contained in the system processing nodes within an execution scenario and its respective tasks. Thus, for resource usage models, we only use a subset of elements in the metamodel.

2.3. Source of execution information

The source of information of our approach is a combination of system logging and process activity (see "c" in Figure 1). In our previous work [3], we described how we synchronize and a combine these two sources to extract information than aligns with the elements and relationships described in our metamodel. So far, we have explored execution data to determinate dependencies (interaction and relationships) between scenarios, tasks, and software components. However, to construct resource usage models, the source of execution information, we need to include data about activity of resources such as processors and memory of the system's processing nodes.

Various monitoring tools, provided by usual runtime platforms, [LTTng, xPerf, SysInternals] support the collection and integration of this sort of data to our source of information. For instance, we have extended our source of information with activity that can describe the behavior of the processor(s) and memory of the system' processing nodes. The processor activity consists of counters that measure the performance of the processor's activity such as arithmetic and logical computations, initialization of operations on peripherals, and execution of threads. The memory activity consists of counters that measure the access to the various sections of the system physical and virtual memory. Additional resource activity can be also collected for the cache, which consists of counters that monitor the file that stores recently used data as long as possible to permit access to the data without having to read from the disk. This is typically used as indicator of I/O operations.

3. Analysis input and interpretation of execution activity

Technically, it is possible to construct all sorts of resource usage models for a given execution scenario. However, the best for the practitioners is to construct first the models that address the main concerns with a

given execution scenario. In this section, we describe how to analyze an execution scenario to identify which resource usage model (see Table 1) should be constructed first and therefore what resource activity should be collected and interpreted.

3.1. Scenario input analysis

For the construction of execution models, i.e. resource usage models, our approach relies on the practitioners' criteria to select key execution scenarios. As the input for our approach, it is important to define as early as possible the resource usage model to be constructed. This is particularly important to decide what execution data to collect and how interpret it. The analysis we follow consist in the identification of the drivers and policies that determinate the resource usage within a given execution scenario.

On the one hand, the drivers are the characteristics of the system functionality, delivered by the execution scenario, that require one or more specific resources. For instance, for an execution scenario that delivers a computation-intensive functionality, the computation-intensive characteristic is the driver for the resource usage. This driver triggers the concern about the proper usage of the available processing nodes' processor and suggests the construction of a CPU usage model using execution activity of the corresponding processor (s).

On the other hand, the policies include design and implementation facts that determine which system processing nodes, software components, and resources (e.g., processor or memory) are involved within the execution scenario at hand. This information can be collected asking practitioners and complemented constructing a scenario overview as we presented in [3].

For a single process and single processing node system, this analysis may be trivial, but when the system at hand is large and complex, this is necessary to manage size and complexity when deciding which software components, processing nodes, and resources should be monitor and perhaps instrument to collect the appropriate execution activity. In resume, the identification of drivers and policies of a chosen scenario help in narrowing down the identification and analysis of the required instrumentation to collect execution activity, stakeholders, and moreover the identification of the model(s) that may address the specific scenario concerns. We will describe more about keys and policies of execution scenarios in Section 5.

3.2. Interpretation of execution activity

Based on our observation on how a large organization develop a large software-intensive system, we consider that presenting resource usage information in terms of the system at hand and in a top-down fashion is appealing for practitioners following top-down or architecture-driven strategies. We have shown in [3] that applying mapping rules it is possible to interpret execution activity, i.e. logging messages and process activity to extract instances of execution elements such as the tasks of a scenario, the involved software component, the respective processes and threads. To map resource activity to higher execution elements we have added two rules to our set of mapping rules.

- *Resource activity information:* The first rule assign unique or combination of resource activity counters (see Section 2.3) to a resource as its most representative execution information. To design this rule we have studied and analyzed the counters that can be collected using the available monitoring tools [LTTng, xPerf, SysInternals] and combine it with the characteristics of the scenario at hand. For instance, to interpret memory activity in data intensive-scenario, we selected the Working Set counter as the most representative value, because it describes the actual amount of memory (rather than the total allocated amount) used at a given period of time (second) within the execution time. For the interpretation of processor activity the type of counter is only one, but when analyzing scenarios using multicore processors, the mapping may consist in using averages and maximums values in case there is not explicit design policy that pointed a particular core.
- *Design information:* The second rule maps text patterns from logging messages and process activity events to design tasks, software components, and threads information activity into numerical values that can be synchronized and combined with the collected resource activity counters' values. Together, the interpreted resource activity and the codified design information are synchronized in a common buffer which is the input for the *Construction of Execution Model* activity of our approach (see Figure 1a). Figure 2 shows a snip of this input. The current tool support that we use to process this input and construct a resource usage models like in Figure 3 is Microsoft Excel and the .netCHARTING library [ref].

Time	Process	PID	TID	Logging Message	Task Code	Available Memory
..	..	..	..	..	..	..
..	..	..	..	..	..	..
00:04.37	MRBootstrap	..	..	----------	2	7.00
00:04.37	MRBootstrap	..	..	Starting Background	15	7.00
00:04.37	MRBootstrap	..	..	----------	15	7.00
00:23.65	MRBootstrap	..	..	completed startup of Background	2	6.44
..	..	..	..	..	..	..
..	..	..	..	..	..	..
00:30.06	MRBootstrap	..	..	----------	2	6.44
00:30.06	MRBootstrap	..	..	Starting Application Software	15	6.44
00:30.06	MRBootstrap	..	..	----------	15	6.44
01:06.98	MRBootstrap	..	..	completed startup of Application Software	2	5.98

Figure 2. Input information to construct a workflow resource usage model

4. Resource usage models

The resource usage models constructed with our approach aim at describing resource usage information in a top-down fashion to support top-down or architecture-driven strategies to analyze the execution of large software-intensive system. We distinguished them as: workflow, functional, and thread resource usage models. In this section we describe the involved execution elements and the used notation for each of them. Color coding is necessary to represent and differentiate the various elements of the models, so we suggest reading the article on-screen or using color-printed versions. The value and benefit of these models are described in Section 5.

4.1. Workflow resource usage models

This type of model is the most coarse-grained representation of resource usage information that we construct. The most important aspect of this type of model is to describe the relation of cause and effect (causality) between the tasks of an execution scenario and the usage of resources such as processors and memory. For instance, Figure 3 shows a resource usage model that describes memory usage within the workflow of the boot of the MRI system divided into two of its major tasks.

A workflow resource usage model contains the tasks of the scenario, the counter values of the monitored resource (s) under analysis, and the period of time which the execution of the scenario takes place. The notation used in this model represent the period of time as a horizontal axis over which the measured resource(s)' counter values are plotted. A vertical axis at right side of the model is a reference for the counters' values. On top of plot of the counter values, the tasks of the scenario are represented as a consecutive sequence of active series along the execution time (horizontal axis).

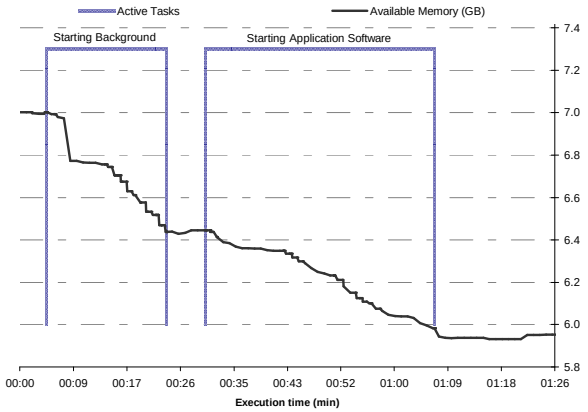

Figure 3. Workflow resource usage model for memory usage

4.2. Functional resource usage models

These models are less coarse-grained than workflow resource usage models, because a model of this type describes the causality between functional components' activity and resource usage. Figure 4 shows a functional resource usage model for the Scan scenario of the MRI system. The elements in this model are the key components that interact to deliver the functionality of the scenario, the counter values of the monitored resource(s) under analysis, and the time of the execution scenario.

The notation in this model is very similar to the one of workflow models, except for the description of software components' activity. According to our execution metamodel (see Figure 1b), a software component or functional component can be mapped to a set of one or more running processes. This abstraction makes it possible to aggregate the activity of various individual process and threads that belong together into a software component activity. This notation makes it possible to describe an overview of software components running in parallel (either in the same or different processing nodes) within the execution of a scenario.

A horizontal segment (consecutive plotted points) represents a software component's activity at a given period. A segment aggregates the interpretation of the various logging messages and process activity event that describe actual activity of the respective set of processes. A space between two consecutive segments represents a period of time that the respective component is inactive, e.g., waiting for some data or control message from other component.

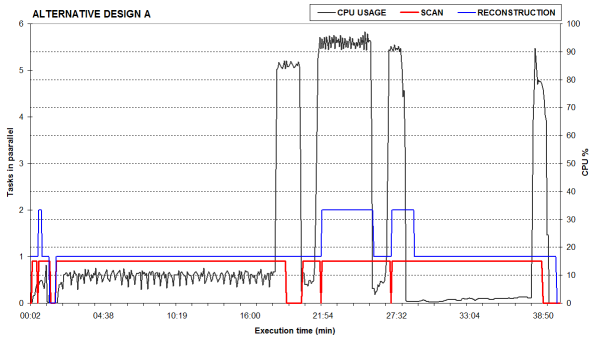

Figure 4. Functional resource usage model for processor usage

4.3. Thread resource usage models

This type of model is the most fine-grained representation of resource usage that we construct. The description of resource usage at this level is important because, base on our metamodel (See Figure 1b), a thread is the immediate and initial link to consistently map resource usage to higher abstractions such as components and tasks.

The notation of this type of model is similar to functional resource usage models. In this case the horizontal segments represent the given thread activity and the spaces between segments the corresponding inactivity. In addition, if the interpreted execution information from the logging and process activity make it possible, gray lines represent the execution flow, aggregate communication and control flow, between threads.

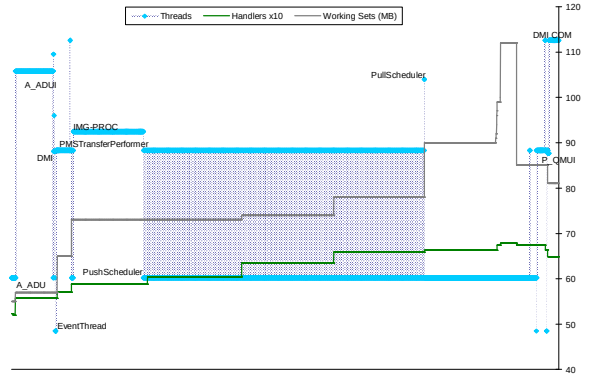

Figure 5. Thread resource usage model for memory usage

To describe and enable the analysis of resource usage at the thread level, it is important to make the role of the involved threads explicit, i.e. their design names instead of ID numbers, their function or role, and even the code elements they execute. This information may not be explicit for practitioner using third party or off-the-shelf components. Thus, it is important to count with information as the one described by the model of Figure 6. This model is a thread and process structure model that we constructed to make the key processes and threads involved in an execution scenario explicit. Further information on why and how to construct this type of model is part of our work in progress. For the context of this paper, we use this model to provide insight and overview of the actual role of the threads in a resource usage model.

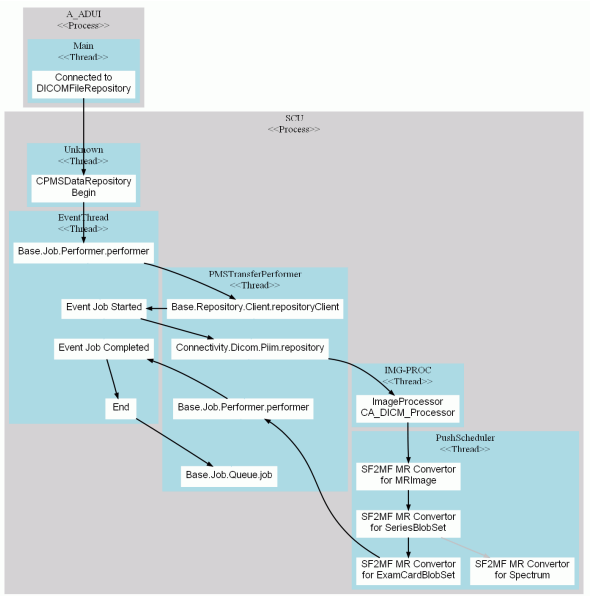

Figure 6. Scenario-based process and thread structure model

Our interpretation has been centered in choosing representative counters, summarize the collected values, and map them to the activity of execution elements such as thread, processes, software components, and task of a scenario. On the one hand, to summarize a processor activity we use the average and maximums values of the counters of each of its cores. On the other hand, to summarize memory activity we choose to use the working set memory counter as a main value because it represent the actual memory that used at a given time, rather than the total allocated.

5. Resource usage models in practice

Within the current validation of our approach, we have constructed resource usage models to support the analysis of the execution architecture of the system of our industrial partner, the Philips MRI scanner. In overall, resource usage models are considered as plots that describe resource activity along a time axis. The key contribution of this representation for practitioners

is that it enables the analysis and understanding of variations (pikes and deeps) on resources activity already in terms of their specific system elements such as tasks, software components, and design threads, and more over address the aspect within the resource usage viewpoint (see Table 1).

5.1. Addressing concerns

Each type or a combination of the resource usage models that we describe in Section 4 provide information that one can use to address the concerns described in Table 1 in the following ways:

- The model in Figure 3 helps to measure the actual memory budget for the task under analysis. Constructing similar models for key scenarios, i.e. test cases scenarios, help to the definition of system benchmarks. Furthermore, it is possible to zoom in on the task structure, divide a task into smaller tasks, to conduct a more accurate analysis of the resource usage variations within a given task.
- In figure 4, the information described by the model, i.e. processor activity, and components active and inactive periods plus the context of the scenario and the domain knowledge of the given analyst will help to determinate the precise nature and causality of inactive periods that can be considered as bottlenecks and delays situations.
- The information provided by the model in Figure 5 also enables the analysis of bottlenecks and delays but at a finer granularity. This granularity enables a downstream development activity, because one can communicate or share the results analysis to the internal or external provider that develop or maintain the code elements executed within the given threads.

5.2. Supporting development activities

As part of the setting of our research project, we have constructed resource usage models within actual development projects observing that resource usage models can support the following activities:

- In overall, the resource usage models that we have constructed contribute to system understanding. On the one hand, we as researcher acquired domain knowledge about the system functionality, design, and implementation. On the other hand, we observed that a resource usage model is often a medium to discuss and transfer technical knowledge between practitioners analyzing it.
- As the part of our early experimentation, we have constructed a set of models for the scenario in Figure 4 using different configurations of the system (alternative designs and implementations). There we learned how practitioner could use resource usage models to analysis alternative designs and implementations towards the detection and correction of bottlenecks and waiting times to improve the performance of the system.
- We observed that some designs rely on the efficient use of resources, i.e. multicore processors to achieve certain requirements. Thus, similar to the support of analysis of alternative designs and implementations, one can construct resource usage models within the testing and verification phase, analyze if the implementation is actually using the given resource as is stated in the design specification.

6. Conclusions and future work

The representation of the resource usage models that we present in this article is not much different from visualizations provided by the existing monitoring tools [LTTng, xPerf, SysInternals]. However, we consider that our contribution is that we provide a structured approach that a software development organization or a tooling provider can use to describe resource usage using high level and specific system information.

Our ongoing work focuses on using execution models, including resource usage models, to identify opportunities of improvement in a top-down fashion to ease corrective maintenance or tune nonfunctional properties such as performance. Thus, we expect to report on this activity as part of our future work.

Acknowledgments

This work has been carried out as a part of the Darwin project at Philips Healthcare under the responsibility of the Embedded Systems Institute. This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

References

[1] Philips Healthcare - Magnetic Resonance Imaging, February 2009 http://www.healthcare.philips.com/main/products/mri/index.wpd

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, Model-based performance prediction in software development: A survey, *IEEE Transactions on Software Engineering*, vol. 30, pp. 295--310, 2004.

[3] T. B. Callo Arias, P. Avgeriou, and P. America, Analyzing the Actual Execution of a Large Software-Intensive System for Determining Dependencies, in *15th Working Conference on Reverse Engineering*, 2008.

[4] M. Devarakonda and R. Iyer, Predictability of Process Resource Usage: A Measurement-Based Study on UNIX, *IEEE Transactions on Software Engineering*, vol. 15, pp. 1579--1586, 1989.

[5]ISO/IEC-JTC1/SC7, ISO/IEC 42010 Systems and software engineering - Recommended practice for architectural description of software-intensive systems 2007.

[6]J. Muskens and M. Chaudron, Prediction of run-time resource consumption in multi-task component-based software systems, in *International Symposium on Component-Based Software Engineering*: Springer LNCS, 2004.

[7]P. van de Laar, P. America, R. Rutgers, S. van Loo, G. Muller, T. Punter, and D. Watts, The Darwin Project: Evolvability of Software-Intensive Systems, presented at *3rd International IEEE Workshop on Software Evolvability* 2007.