Technical Report: Documenting a Catalog of Viewpoints to Describe the Execution Architecture of a Large Software-Intensive System for the ISO/IEC 42010 Standard

Trosky B. Callo Arias, Paris Avgeriou
Department of Mathematics and Computing Science
University of Groningen
The Netherlands
trosky@cs.rug.nl, paris@cs.rug.nl

Pierre America
Philips Research and
Embedded Systems Institute
The Netherlands
pierre.america@philips.com

I. Introduction

This paper presents the documentation of a set of viewpoints to construct views that describe the execution architecture of a large software-intensive system: what the software of the system does at runtime and how it does it. The documentation of the viewpoints is performed using a proposed template. The documented viewpoints can be used as real examples in the definition of the ISO/IEC 42010 standard. The rest of this document is divided as follows. Section II introduces the proposed template, Section III contains some comments and observations on the proposed template, and Section IV contains the documentation of the viewpoint catalog.

II. The ISO/IEC 42010 style

The template to document viewpoints is proposed by R. Hilliard, editor of the ISO/IEC 42010 and consists of the sections summarized in Table 1.

Table 1. Proposed template

Section	Description
«Viewpoint Name»	The name for the viewpoint, and any synonyms for the viewpoint
«Overview»	An abstract or brief overview of the viewpoint and its key features.
«Concerns»	A listing of the architecture related concerns framed by this viewpoint. This is crucial information for the Architect, because it helps her decide whether this viewpoint will be useful to apply to a given system of interest, and to communicate with its stakeholders
«Anti-Concerns»	Optional. It can be useful to document the kinds of issues a viewpoint is not appropriate for. Articulating anti-concerns may be a good antidote for certain overly used notations.
«Typical Stakeholders»	Optional. The typical audiences for views prepared using this viewpoint. Who are the usual stakeholders for this kind of view?
«Model types»	Identify each type of model used by the viewpoint.
«Model languages»	For each type of model used, describe the language or modeling techniques to be used. Each model language is a key modeling resource that the viewpoint makes available. Model languages provide the vocabularies for constructing the view. ISO/IEC 42010 does not specify how a modeling language is documented. It could be by reference to an existing modeling language (e.g., SADT or UML) or technique (e.g., M/M/4 queues); by providing a metamodel for the language to define the language's core constructs; via a template that users fill in; or by some combination of these methods.
«Viewpoint metamodels»	Optional. A metamodel presents the conceptual entities, their attributes and the relationships that comprise the vocabulary of a type of model. There are different ways of representing ontologies (entity-relation diagrams, class diagrams, [OMG MOF].) Any metamodel should capture: *Entities:* What are the major sorts of elements are present in this type of model? *Attributes:* What properties do entities in this type of model possess? *Relationships:* What relations are defined among entities within this type of model? *Constraints:* What kinds of constraints are there on entities, attributes or relationships within this type of model? [NOTE Entities, attributes, relationships and constraints are all architecture elements in the sense of ISO/IEC 42010]
«Conforming Notations»	Identify an existing notation or model language to be used for this type of model.

«Model correspondence rules »	The viewpoint may specify model correspondence rules. Each one may be documented here.
«Operations on views»	Operations define the methods which may be applied to views and their models. Operations can be divided into categories: Creation methods are the means by which views are prepared using the viewpoint. These could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type); heuristics, styles, patterns, or other idioms. Interpretive methods provide the means by which views are to be understood by readers and system stakeholders. Analyses methods are used to check, reason about, transform, predict, apply and evaluate architectural results from this view. Implementation methods capture how to realize or construct systems using information from this view.
«Examples»	Optional. This section provides examples for the reader.
«Notes»	Optional. Any additional information users of the viewpoint may need.
«Sources»	What are the sources for this viewpoint, if any? This may include author, history, literature references, prior art, etc.

III. Comments on the template

This section provides some comments (see Table 2) on the proposed template. The comments are based on our experience defining and using viewpoints to construct architectural views of an existing large software-intensive system.

Table 2. Comments on the template

«Typical Stakeholders»	▪ We propose that this section should be *required*. A viewpoint should have at least one stakeholder to show its targeted audience and value. ▪ Furthermore, we propose to include a brief overview of the stakeholders' role to clarify their wider concerns either within the development or use of the system. This is important since different organizations may assign different responsibilities for the same stakeholder, e.g., due to the size and complexity of the system or the organization.
«Viewpoint metamodel»	▪ We consider that a viewpoint metamodel should be a mandatory element in the description of a viewpoint or a catalog of viewpoints (a set of more than one viewpoint). ▪ Such a metamodel describes the ontology, i.e. the set of entities, attributes, and relationships that the involved stakeholders may use to describe and decompose the system at hand with respect to the given viewpoint(s). ▪ In case of a common metamodel for a set of viewpoints, the metamodel is crucial in providing explicit information about the consistency and relationships between the involved viewpoints and their respective views and models.
«Model types»	▪ We consider that the description of each model type should be composed of two subsections: model language or conforming notations (appear as separate sections in the proposed template) and viewpoint metamodel subset. «Viewpoint metamodel subset» ▪ Since all the different models of a viewpoint aim to describe the same system, we consider that all models should share a common metamodel. ▪ Then, each model type may use the complete common metamodel or a subset of its elements. ▪ However, we observed that in some cases (see resource usage viewpoint), a viewpoint description can extend the metamodel, (e.g., adding or specializing its elements) to frame concerns related to information with finer level of abstraction or detail. ▪ Extensions or specializations of the metamodel's elements can be incorporate if the stakeholder of the catalog agree on that or simply when the extensions are relevant for more than one single model and viewpoint. «Model languages» or «Conforming Notations» ▪ We don't see a clear difference between model language and conforming notations. Both, model language and conforming notations can be understood as the syntax and semantics used by each model type to describe instances of the common metamodel's elements and relationships.
«Examples»	▪ At least one example for each model type should be mandatory to illustrate and describe the use of the model language, the viewpoint metamodel subset, and the conforming notations.
«Model correspondence rules »	▪ Our assumption is that model correspondence is primarily necessary about inter-view and not intra-view relations. If this is the case, it should be clear in the template description ▪ For intra-view correspondence rules, i.e. within models of the same viewpoint, the underlying common metamodel's relationships will already express the correspondence rules. ▪ For inter-view correspondence within views of different viewpoints definitions, the metamodels of the different viewpoints should be related. Otherwise, the definition of model correspondence rules across their models may be difficult or impossible ▪ Thus, we consider that the idea of a common metamodel is also valid to help the description of and enforce inter-view correspondence among views and models of different viewpoints.

IV. DOCUMENTATION OF THE EXECUTION VIEWPOINT CATALOG

TABLE 3, TABLE 4, TABLE 5, and TABLE 6 respectively contain the documentation of four viewpoints that form an execution viewpoint catalog. The catalog and its viewpoints aim to support the construction and analysis of views that describe the **execution** architecture of an existing software-intensive system. These viewpoints share a common metamodel (see Figure 1) that decompose a software-intensive system in term of runtime entities. At the same time, the metamodel provides means to link high-level entities with entities such as data and code. The main aspect that enables the latter is the fact that the metamodel maps the concept of software component to a set of actual runtime processes that belong together. The origin of the viewpoints described in this catalog is a research activity to support the construction of views for an existing software-intensive system, the Philips MRI system, thus the documented viewpoints use examples of this existing system.

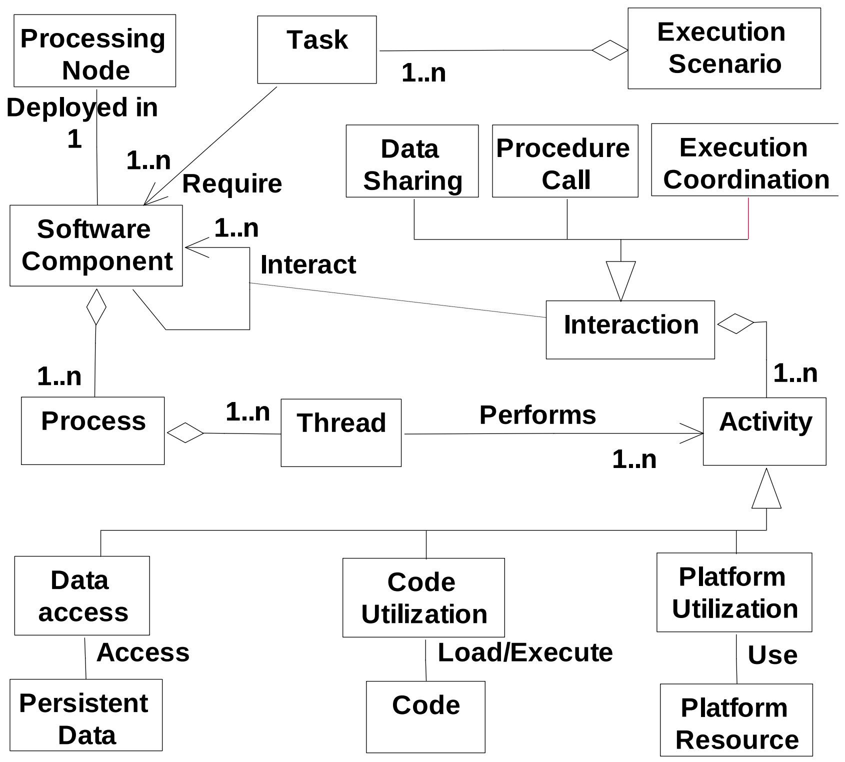


Figure 1. Common metamodel for the execution viewpoint catalog

Table 3. Execution profile viewpoint

«Viewpoint Name»

Execution profile

a.k.a. Scenario overview or functional mapping

«Overview»

An execution profile view is an overview of the interacting runtime elements involved in the execution of the functionality of a system. The system functionality is represented by a set of key execution scenarios identified by the system's development organization.

«Concerns»

- What are the major components that realize a given system function?
- What are the high-level dependencies that couple major components?
- What is the development team that develops or maintains a given system's function?

«Anti-Concerns»

«Typical Stakeholders»

Execution profile views may be of interest to project leaders, architects, testers, operating system supporters, and newcomers in a development organization.

- Project leaders are mainly concerned about the efficient and cost effective planning and execution of development activities.
- Architects are mainly concerned about the maintenance of the system architecture description and facilitate the communication between the technical and business areas of a development organization.
- Testers are mainly concerned about the efficient design and execution of system test cases, and the communication of results to the development organization.
- Operating system supporters are mainly concerned about the efficient design, customization, and maintenance of the underlying system's software platform (operating system).
- Newcomers are new personnel that are mainly concerned about the acquisition of system-specific knowledge to start contributing in the development and maintenance of the system.

«Model types»

Functional mapping and Dependency matrix models:

«Subset of common metamodel elements»

Tasks, software components, processes, and major data and code elements

«Model languages and conforming notations»

- *A functional mapping model* is a graph-based model that describes relationships between high-level elements of a key execution scenario. The notation of a functional mapping model (see Figure 2) is:
  - The scenario is divided in a set of tasks that are link (e.g., using color-coded edges) to the software components that realize each of them. The links, e.g. color-coded edges, identify the trajectory of a task through the rest of elements in the model.
  - Each software component is respectively mapped to its set of running processes (e.g., using record structures where the fields of the record represent the processes).
  - The links continue and describe the activities of a software component (acting as the subject) on objects; the objects represent identified specializations of data, code, or platform resources. For instance, components may read from or write on data repositories.
- *A dependency matrix* describes similarities between the tasks of scenarios, scenarios, software components, and vice versa that can be characterized as high-level dependencies. The notation of a dependency matrix model (see Figure 3) is:
  - Row and columns represent execution elements such as tasks and software components.
  - The cells of a dependency matrix represent the metrics of runtime activity (e.g., reading and writing activity on data repositories) of the elements represented in the respective cell's row and column.

In general, execution profile models aim to provide high-level information to describe and make it explicit:

- The sequence of tasks within the set of execution scenarios
- The set of software components and their respective set of processes
- Distinction of the execution activity per task and per software component on data repositories, code modules, and system-specific resources.

«Examples»

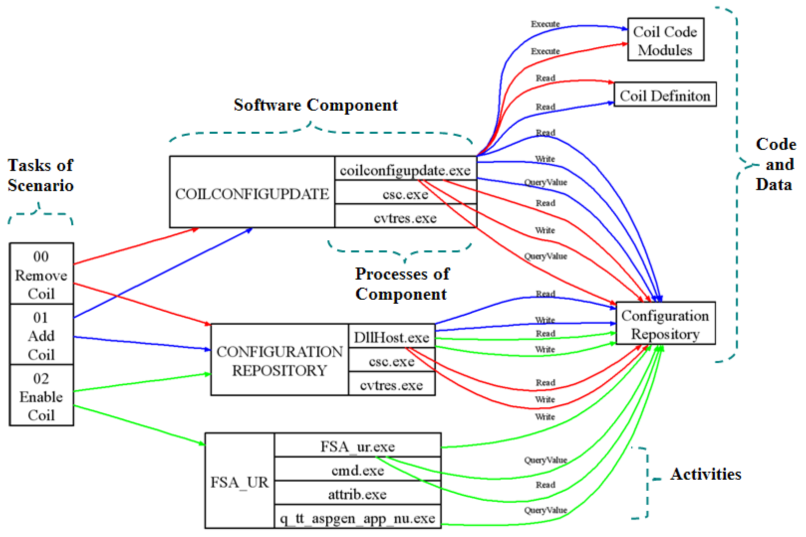
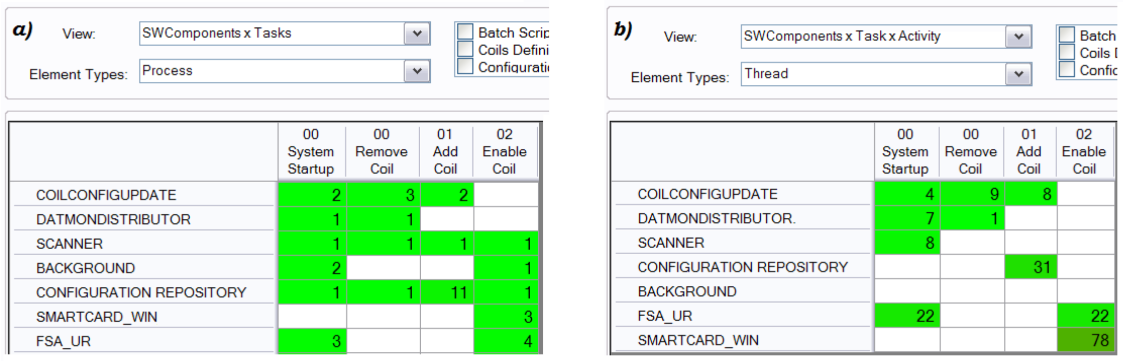


Figure 2. Example of a functional mapping model



a) View: SWComponents x Tasks

Element Types: Process

☐ Batch Scrip
☐ Coils Defini
☐ Configurati

	00 System Startup	00 Remove Coil	01 Add Coil	02 Enable Coil
COILCONFIGUPDATE	2	3	2	
DATMONDISTRIBUTOR	1	1		
SCANNER	1	1	1	1
BACKGROUND	2			1
CONFIGURATION REPOSITORY	1	1	11	1
SMARTCARD_WIN				3
FSA_UR	3			4

b) View: SWComponents x Task x Activity

Element Types: Thread

☐ Batch
☐ Coils I
☐ Confic

	00 System Startup	00 Remove Coil	01 Add Coil	02 Enable Coil
COILCONFIGUPDATE	4	9	8	
DATMONDISTRIBUTOR.	7	1		
SCANNER	8			
CONFIGURATION REPOSITORY			31	
BACKGROUND				
FSA_UR	22			22
SMARTCARD_WIN				78

Figure 3. Example of dependency matrices models

Figure 2 and Figure 3 show execution profile models of one of the functionalities of an MRI Scanner system. The functionality is the configuration of a coil device, a system specific hardware device.

«Model correspondence rules »

- Every software component described in an execution profile model is deployed and run on a processing node. This relationship is a correspondence rule between models of an execution profile and an execution deployment view that describe the corresponding software component and processing node respectively.
- To identify and derive more rules one can use the relationships between tasks, software components, processes, data repositories, and code elements described in the common metamodel.

«Operations on views»

Creation methods: To construct execution profile models using actual execution information one should take into account the following operations:

- Choose a set of execution scenarios (e.g., test cases and integration tests) that are representative for the system functionality that need to be described. For each scenario, define or identify the sequence of tasks or workflow that build them and that should be followed by the end-user or automatically executed by the system.
- The actual sequence of task or workflow of a scenario can be extracted using logging mechanisms that are part of the system infrastructure or monitoring utilities provided by the system platform.
- Similarly from logging or monitored activity, it is necessary to extract:
  - The actual set of involved software components and their corresponding set of processes and threads.
  - Aggregations that represent system data repositories, code libraries/packages, and if possible the system specific hardware devices, e.g. *Configuration Repository* or *Coil Code Modules* as shown in Figure 2.
  - The execution activity that describes how software components' processes use data repositories, code libraries, and system specific devices.

Analysis methods:

- Project leaders and newcomers may use an execution profile view to learn about the system functionality, the set of major components (hardware, software, and data) that realize it, and the high-level dependencies that couple them.

- Execution profile views may be used to support downstream planning of development projects. For instance, identify the development force, i.e. internal and external teams that are in charge of the development and maintenance of the identified components that perform a given system function to be change within a development project.
- For testers and operating system supporters, an execution profile view can provide information to identify the actual processes and execution elements such as data repositories and platform resources that may influence or play a role in the design of test cases, the assessment of test results, and the report for corrective maintenance activities.
- Project leaders can use dependency matrix models to identify interactions between tasks of scenarios or between software components, which can be characterized as horizontal dependencies. Comparing relationships between software components against task can be characterized as vertical dependencies. In both cases, horizontal and vertical dependencies can be characterized as change dependencies to be analyzed within project planning.
- The value in the cells of a matrix can be changed for different purposes. For instance matrix (a) compare components and task using as metric the number of active processes of a component within a task. The second matrix (b) does same comparison but uses a finer grain metric, the total number of created threads by a component within a task.

«Notes»

- Often, when constructing an execution profile view, it will be necessary to use some domain knowledge to filter out any artifacts or noise from the extracted high-level information. Further detail about it and creation methods for an execution profile view see source I.

«Sources»

I. 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.

Table 4. Execution deploymen viewpoint

«Viewpoint Name»

Execution Deployment

a.k.a. Deployment

«Overview»

Execution deployment is base on regular deployment viewpoints (see sources I and II) and focuses on the description of the environment of a system with special attention on the distribution of processing and the level of detail. This is to make it explicit how software components are distributed on to processing nodes and the relation with the environment in which the system is deployed.

«Concerns»

- How are software components distributed among processing nodes at runtime?
- How are the system software components, distributed in different processing nodes, and do they interact or connect at runtime?
- How are the physical links between the system processing nodes allocated to software components?
- What are the system's physical and functional constrains that rule the location of processing nodes and their respective software components?

«Anti-Concerns»

«Typical Stakeholders»

Execution deployment information is recognized to be of a particular interest of system administrators, architects, designers, software engineers, testers, and system infrastructure supporters. They play the following role:

- System administrators and system infrastructure supporters are mainly concerned about the efficient administration and support of a system functioning in the field and within development activities respectively.
- Architects are mainly concerned about the maintenance of the system architecture description and facilitate the communication between the technical and business areas of a development organization.
- Designers are mainly concerned about the design of system components and coordinate their implementation and testing with software engineers and testers respectively.

«Model types»

Physical deployment models:

«Subset of metamodel elements»

Processing nodes, software components, processes, data repositories, and code elements

«Model languages and conforming Notations»

- *A physical deployment model* is a regular deployment model that its notation aims to describe three main aspects: detail of processing nodes, detail of links between processing nodes, and organization of processing nodes.
- *Detail of processing nodes:* Boxes are often used to represent processing nodes in a deployment model. Inside them, the graphical notation should differentiate software components or functional elements from important code libraries, data repositories, and system-specific hardware devices.
- *Detail of links between processing nodes:* Lines can be used to describe links between processing nodes such as network or communication lines. These lines should be complemented with textual description about the function of the link, the link's technology characteristics, and the capacity or bandwidth the connected nodes require from the link.
- *Organization of processing nodes:* Grouping of processing nodes can be used to describe their distribution that should resemble as much as possible to the actual physical and geographical distribution of the system at hand.

«Examples»

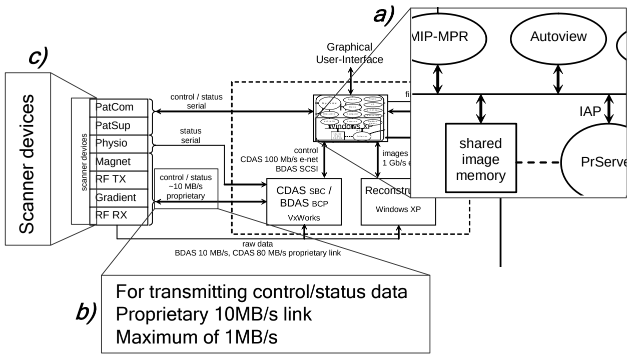


Figure 4. Example of a execution deployment model

«Model correspondence rules »

- Processing nodes, described by execution deployment models, contain software components that run using the respective processing node's resources. This relationship is a rule to identify the correspondence between the models of an execution deployment and models of views such as execution profile and resource usage that describe a given software component and its resource usage on the corresponding processing node.
- To identify and derive more rules see the relationships between processing nodes, software components, processes, and data and code elements described in the common metamodel.

«Operations on views»

Creation methods: To construct execution deployment one should take into account the following operations:

- Processing nodes should include description about the software components (or groups of running processes), important resources (such as data repositories and third-party software), and when possible, the major links (e.g., busses and share memory) between software components (see ***a*** in Figure 4).
- When describing the links between processing nodes one should include: the function of the link, the technological characteristics of the link, and the resources that the software system requires from the link.
- These three aspects are important to make the role of the link within the system runtime explicit (see b in Figure 4). This guideline should also applied when creating deployment views for part of the system or when zooming in a given processing node structure to describe the internal deployment of software components to specific processors or cores (e.g., in a multiprocessor or multicore architecture).
- In addition, for regular deployment view, it is recommended that the description of a processing node should include its current technological characteristics (e.g., operating system, processor speed, and number of cores) and whenever possible a brief description of why this characteristics are required.

Analysis methods:

- For the particular case of the execution deployment, architects and designers can use it to describe the overall architecture of the system highlighting the allocation of software components (including major data repositories and specialized third-party libraries) on to processing nodes or computers.
- Analysis of the details about links between processing nodes is useful for software engineers and testers to identify how these links may be used to connect the software components that they develop, maintain, and test respectively.
- Analysis of the organization of processing nodes is particularly recommended to make some design decision explicit, such as safety issues and rules to manage the influence of physical phenomena (e.g. magnetism and levels of temperature) on processing nodes. For instance, Figure 4 shows how processing nodes and the software components they contain are distributed as a group of scanner control devices (see c) or user interface elements. This is relevant for infrastructure supporters and stakeholders involved in development phases to learn about the physical and functional constrains (e.g., magnetic fields, noise, levels of temperatures, and required shimming) that benefit or limit the inclusion and redistribution of system components in the system environment.

«Notes»

«Sources»

Deployment Viewpoints :

I. N. Rozanski and E. Woods, Software Systems Architecture: working with stakeholders using viewpoints and perspectives: Addison Wesley 2005.
II. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures. Views and Beyond: Addison Wesley, 2002

Table 5. Resource usage viewpoint

«Viewpoint Name»

Resource Usage

«Overview»

The typical hardware resources that are used by software system are processors, memory, disk, and network bandwidth. Hardware technology changes very often and development organizations regularly evaluate the consequences of these changes, in particular because inappropriate or unpredicted resource usage can compromise non-functional properties, e.g. performance and reliability, triggering the execution of expensive corrective maintenance and even redesign activities.

«Concerns»

- How to assure adequate resource usage and justifying the development effort needed to accommodate hardware resources changes?
- What are the metrics, rules, protocols, and budgets that rule the use of resources at runtime?
- How software components and their respective processes consume processor time or memory within key execution scenarios?
- Does the realization of the system implementation have an efficient resource usage?
- What are the bottlenecks and delays of the system and their root cause?
- How is the third party software behavior using resources such as processor and memory?

«Anti-Concerns»

What is the designed or required resource usage?

Resource usage is different from required resources, which is covered by the deployment viewpoint. For instance, usual deployment models describe network connections with the capacity of the physical network link. Instead, the resource usage viewpoint shows how to describe the actual capacity used overtime and enables the analysis of the difference between the required (budgeted) network capacity and the provided capacity.

«Typical Stakeholders»

Resource usage information is of a particular interest of system administrators, platform/infrastructure supporters, architects, designers, software engineers, and testers. They play the following role:

- System administrators and system infrastructure supporters are mainly concerned about the efficient administration and support of a system functioning in the field and within development activities respectively.
- Architects are mainly concerned about the maintenance of the system architecture description and facilitate the communication between the technical and business areas of a development organization.
- Designers are mainly concerned about the design of system components and coordinate their implementation and testing with software engineers and testers respectively.

«Model types»

Task, component, and thread resource usage models (i.e. processor, memory, and network), budgets, predictions.

«Subset of metamodel elements»

Processing nodes, software components, processes, thread, hardware resources, and the specializations described in Figure 5.

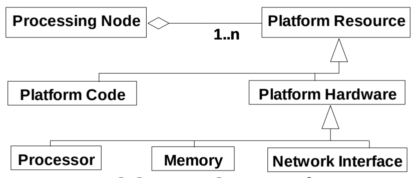


Figure 5 Metamodel specialization for resource usage

«Model languages and conforming notations»

- *Task resource usage models* are the most coarse-grained representation of resource usage information. The purpose of this type of models is to describe the correlation between the tasks of key execution scenarios and the activity of hardware resources.
- *Component resource usage models* are more fine-grained than task resource usage models. The purpose of this type of models is to describe the correlation between the activity of each system software component (set of one or more running processes) and the usage of hardware resources within key execution scenarios.
- *Thread resource usage models* are the most fine-grained representations of resource. The purpose of this type of model is to describe the correlation between thread activity and the activity of hardware resources.

These three types of models share the following common notation:

- The workflow or activity of the runtime element, e.g., task, component, or thread can be represented as consecutive segments along the execution time (horizontal axis), which can be called execution periods.
- Execution period of software components should be presented along a horizontal time axis and distributed in groups that assemble their mapping or distribution onto processing nodes. For instance, Figure 7 illustrates the distribution of execution periods into computers named scanner and recon.
- Two separate vertical axes are needed. The first at the left side as a reference for the execution period of components and threads. The second at the right side as reference for the resource usage values.
- To ease the visualization of the correlation, plot the resource usage measurements values below the horizontal execution periods of components or threads.

«Model correspondence rules »

- Every thread described by the models of a resource usage performs runtime activities using various code and data elements. This relationship is a correspondence rule that helps to identify the models of a concurrency view that describe the threads described in a given thread resource usage model (e.g., this rule apply to models in Figure 8 and Figure 10).
- To identify more rules, see the relationships between processing nodes, software components, processes, thread, and hardware resources described in the common metamodel

«Operations on views»

Creation methods: To construct resource usage models one should take into account the following operations:

- Resource usage descriptions should be based on actual resource usage measurements (e.g., using tools such as Process monitor or Windows Performance Analyzer)
- Choose a set of execution scenario as a benchmark of the system functionality under analysis and use representative input, e.g., data sets, to stress the hardware resources involved in their execution.
- The collected data should include workflow information, e.g., logging and the respective resource usage measurements.
- To correlate the measurement with descriptions in terms of the system design or architecture, the following information should be extracted from the collected data:

For descriptions at the software component level, i.e. Figure 7:

  - The actual set of involved software components and their corresponding set of processes.
  - The execution periods of each software component, that is involved in the execution scenario.

For descriptions at the thread level, i.e., Figure 8:

  - The actual set of involved process and their respective threads. To identify the set of actual threads, it will be useful to have at hand a concurrency model (see concurrency viewpoint) of the scenario under analysis.
  - The execution periods of the identified threads, i.e. aggregations of consecutive thread execution activity, and when possible the control and dataflow between threads.

Analysis methods:

- Software architects, designers, and platform supporters, can analyze task resource usage models to predict, and tune resource usage budgets. For instance, the model in Figure 6 helps to identify the actual memory required for the described scenario and its respective tasks. This information can be used to reformulate the designed memory budged of the given scenario.

- Architects and designers may also use resource usage models, e.g. Figure 7, to analyze alternative architectures or designs and compare them based on how efficient processor or memory are used to deliver key computation- or data-intensive system functions.
- Resource usage information is also useful for designers and software engineer to identify opportunities to tune and match design and implementation. For instance, models like Figure 7 and Figure 8 helps to identify correlations between delays or dead times with peaks and valleys of the resource usage and match to the designed or desired interaction between elements such as software components and thread.
- Resource usage information in terms of design elements as in the models in Figure 7 and Figure 8 are useful evidences that ease the communication and sharing of knowledge between internal and external teams, e.g. between designers, platform support engineers, and external providers without drilling down in the implementation code. For instance, the model in Figure 8, was constructed to describe the memory usage of a third party infrastructure used to support the execution of data-intensive system function.
- A resource usage view also eases the definition of benchmarks for the design and execution of test and verification procedures. For instance, having resource usage models of a given execution scenario before and after it is changed, serve as evidence to track and communicate the desired or undesired variations of the runtime of the system.

«Examples»

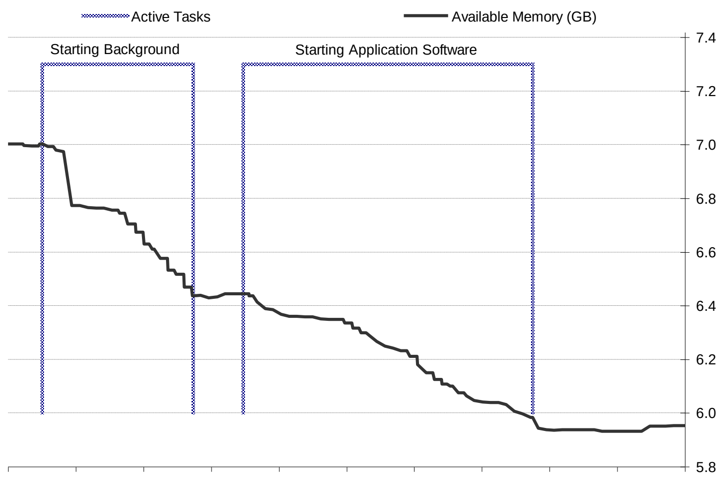


Figure 6. Example of task resource usage model

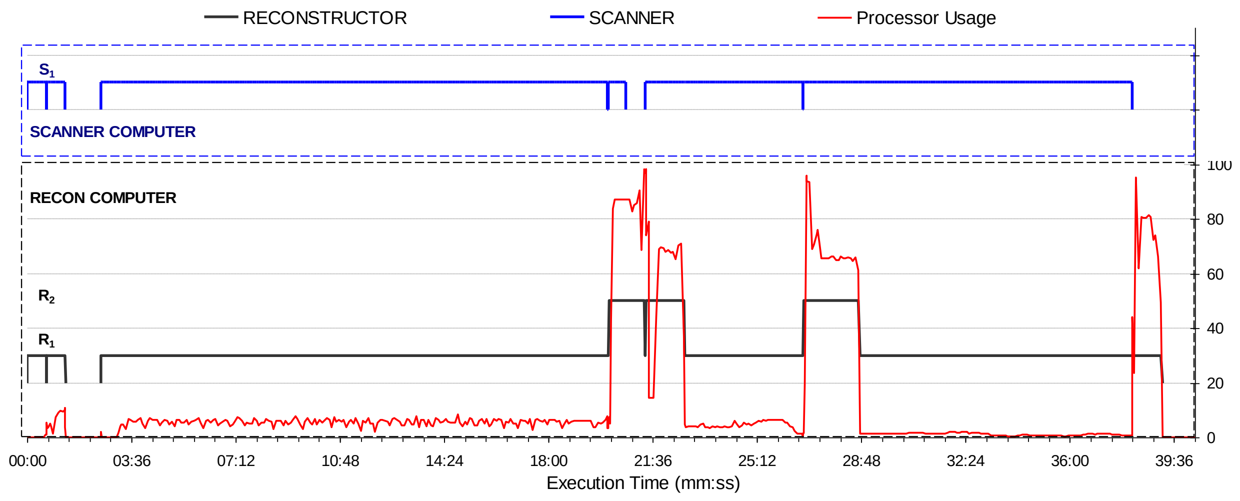


Figure 7. Example of a component resource usage model

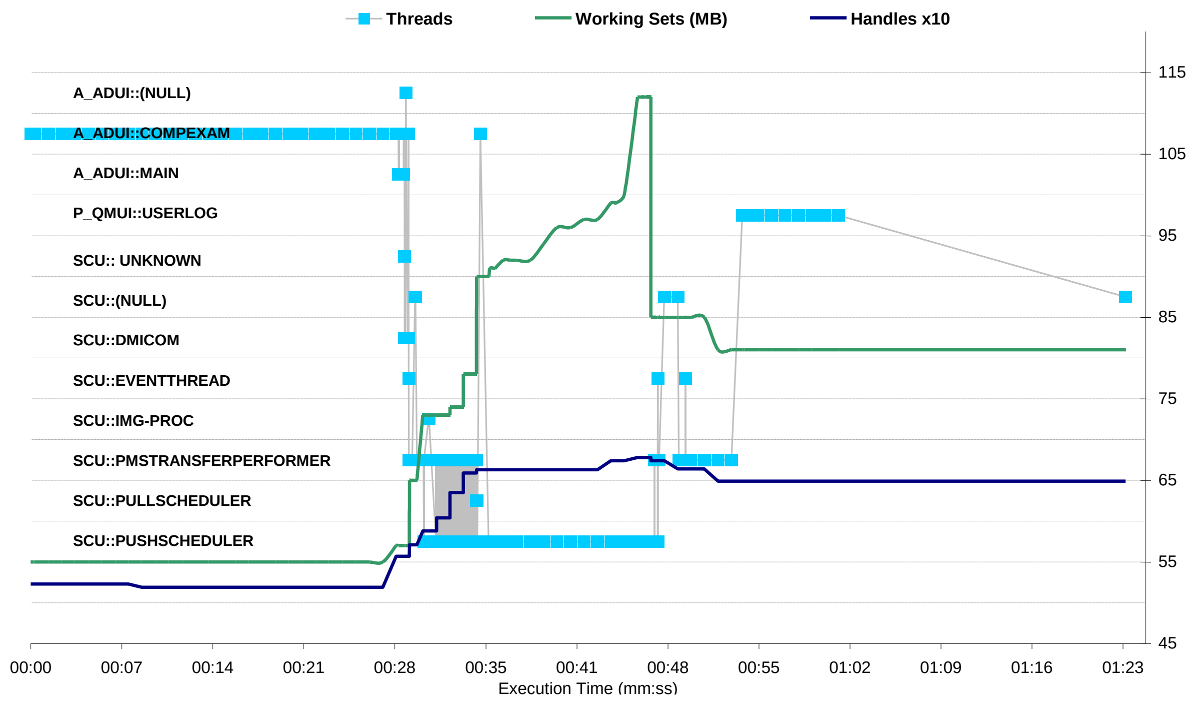


Figure 8. Example of a thread resource usage model

«Notes»

- Often, when constructing an analyzing a resource usage view, it will be necessary to use domain knowledge to understand and use implicit high-level information, such as the communication mechanisms or protocols between software components and system physical constraints. Further detail about it and creation methods for a resource usage view see source I.

«Sources»

I. T. B. Callo Arias, P. America, and P. Avgeriou, Constructing a Resource Usage View for a Large Software-Intensive System, presented at *16th Working Conference on Reverse Engineering*, 2009.

Table 6. Execution Concurrency Viewpoint

«Viewpoint Name»
Execution Concurrency
a.k.a. Concurrency

«Overview»
An execution concurrency view is an overview of how the runtime elements of a system execute concurrently to deliver the system functionality. Runtime concurrency often drifts from designed concurrency in complex and large software-intensive systems, especially when this type of systems includes the integration of components with heterogeneous designs and implementations.

«Concerns»

- Which runtime elements execute concurrently?
- How does the runtime concurrency match the designed concurrency?
- What are the aspects that constrain, coordinate, and control the runtime concurrency of the system?
- What are the opportunities to improve the concurrency of the system?

«Anti-Concerns»

«Typical Stakeholders»
Execution concurrency views may be of interest to architects, designers, software engineers, testers, and operating system supporters. They play the following role:

- Architects are mainly concerned about the maintenance of the system architecture description and facilitate the communication between the technical and business areas of a development organization.
- Designers are mainly concerned about the design of system components and coordinate their implementation and testing with software engineers and testers respectively.
- Testers are mainly concerned about the efficient design and execution of system test cases, and the communication of results to the development organization.
- Operating system supporters are mainly concerned about the efficient design, customization, and maintenance of the underlying system's software platform (operating system).

«Model types»
Workflow concurrency and process-thread structure:
«Subset of common metamodel elements»
Processing nodes, software components, processes, and thread
«Model languages and conforming notations»

- *A workflow concurrency* is a Gantt-char like model that illustrates temporal relations between high-level runtime elements (e.g., tasks or software components). The notation of a model of this type (see Figure 9) is:
  - Elements such as scenario's tasks are represented as segments horizontally and vertically distributed. The horizontal organization corresponds to a time axes that represent the occurrence and duration of the tasks over time. The vertical organization corresponds to location axes, which for instance represent the processing node where the task is executed.
  - Color coding can be useful to distinguish the function or nature of the involved tasks and the borders between their locations.
- *A process-thread structure model*, also know as concurrency model (see resource I) describes the distribution and mapping of functional elements to actual runtime elements such as processes and their threads. The notation of this type of model is:
  - Runtime processes are represented as container of threads. At same time, threads are represented as containers of functional elements such as executable code, runtime events, and interfaces to data and hardware resources.
  - The notations of containers can be usual boxes, and lines connecting them as representation of control and data flow relationships. Richer notations such as UML and stereotyping can be used as well. For instance, the model in Figure 10 use boxes as containers and stereotypes to distinguish between process and threads.

«Examples»

Figure 9 shows a workflow concurrency model of the startup of an MRI Scanner system. The startup is a key scenario to analyze the integration of the various system components. Figure 10 shows a process-thread structure model to analyze the runtime structure of a key data-intensive function of the MRI Scanner system.

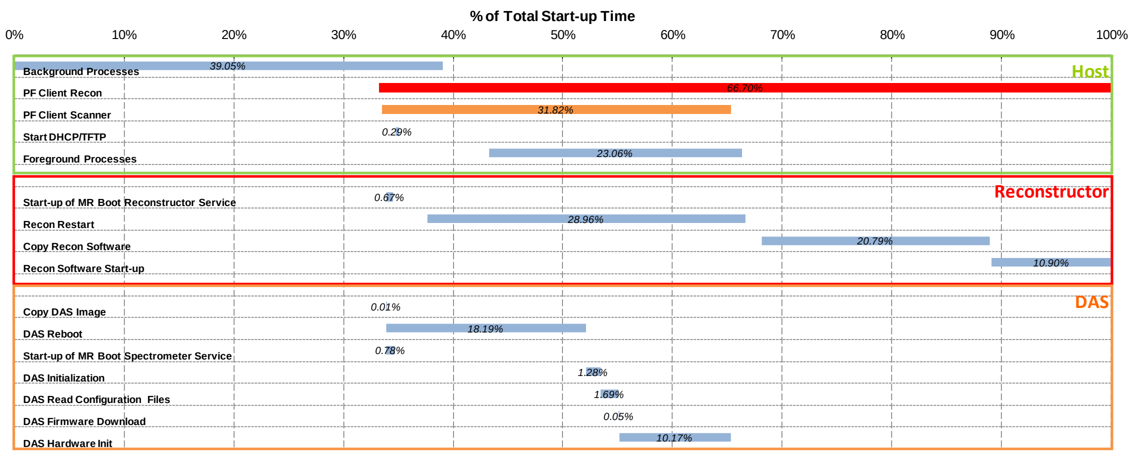
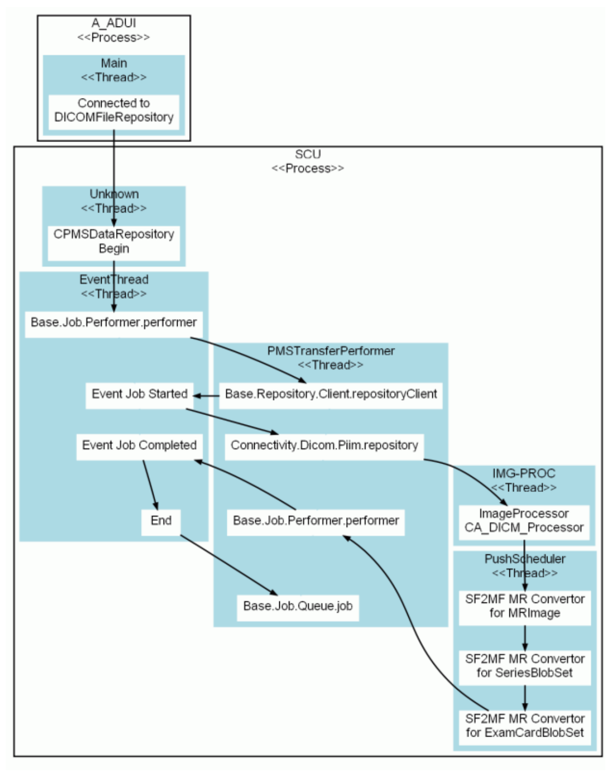


Figure 9. Example of a worflow concurrency model



A_ADUI <<Process>>
Main <<Thread>>
Connected to DICOMFileRepository
SCU <<Process>>
Unknown <<Thread>>
CPMSDataRepository Begin
EventThread <<Thread>>
Base.Job.Performer.performer
PMSTransferPerformer <<Thread>>
Event Job Started
Base.Repository.Client.repositoryClient
Event Job Completed
Connectivity.Dicom.Piim.repository
IMG-PROC <<Thread>>
ImageProcessor CA_DICM_Processor
End
Base.Job.Performer.performer
PushScheduler <<Thread>>
SF2MF MR Convertor for MRImage
Base.Job.Queue.job
SF2MF MR Convertor for SeriesBlobSet
SF2MF MR Convertor for ExamCardBlobSet

Figure 10. Example of a process-thread structure model

«Model correspondence rules »

- Every task and process, described by a model of a runtime concurrency view is part of a given scenario and belong to a given software component respectively. This rule helps to identify the corresponding execution profile model (and vice versa) that describe the execution profile of the given scenario.
- To identify and define more, see the relationships, between software components, processes, threads, and code elements described in the common metamodel.

«Operations on views»

Creation methods: To construct runtime concurrency models using actual execution information one should take into account the following operations:

- Choose a set of execution scenarios (e.g., test cases and integration tests) that are representative for the system functionality that need to be analyzed.
- To construct a concurrency workflow model, define or identify the important tasks that build the chosen scenarios. Then identify the relative start time and duration of each task with respect to the time of execution of the scenario under analysis. This information can be extracted using logging mechanisms that are part of the system infrastructure or monitoring utilities provided by the system platform. Logging can be used to additionally identify the location of the task, but design knowledge can be enough at this level.
- To construct a process-thread structure model, identify the important runtime processes involved in the scenario using design knowledge. Additional information about threads and the code elements that they execute can be identified from logging or monitored process activity. It is important to identify a meaningful name for each thread rather than numeric identifications. This is important to match runtime thread to design threads.
- In addition, analysis of data and code utilization activity collected with logging or monitoring mechanism can be used to characterize and describe control and dataflow relationships between process and between threads

  Analysis methods:

to architects, designers, software engineers, testers, and.

- Architect and designer may use concurrency workflow models to gather high-level information about elements that run concurrently as input for the downstream planning of development activities.
- Concurrency model also eases the definition, design, and execution of test and verification procedures. For instance, testers may use concurrency workflow models as evidence to track and communicate the desired or undesired variations of the runtime concurrency of the system.
- Software engineer may use concurrency models to learn and analyze how the system functions and the pieces of code they implement are instantiated and deployed at runtime. Furthermore, models like Figure 10 will help software engineers as links to understand models from different viewpoints such as Figure 8.
- Similarly, operating system and platform supporters may use concurrency model to acquire and communicate system design knowledge.

«Notes»

«Sources»

I. N. Rozanski and E. Woods, Software Systems Architecture: working with stakeholders using viewpoints and perspectives: Addison Wesley 2005.