Defining Execution Viewpoints for 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

An execution view is an important asset for developing large and complex systems. An execution view helps practitioners to describe, analyze, and communicate what a software system does at runtime and how it does it. In this paper, we present an approach to define execution viewpoints for an existing large and complex software-intensive system. This definition approach enables the customization and extension of a set of predefined viewpoints to address the requirements of a specific development organization. The application of this approach has helped us to identify a set of execution viewpoints that we are currently using to construct execution views of an MRI system, a large software-intensive system in the healthcare domain.

1. Introduction

The usage of multiple views is a common practice to construct and document the architecture of large software-intensive systems [4, 8]. The ISO/IEC 42010 standard provides a widely accepted conceptual definition of architectural views, viewpoints and models [8]:

- *An architectural view* is a representation of a set of system elements and relations associated with them, conforming to a specific viewpoint.
- *An architectural viewpoint* addresses particular concerns of the system stakeholders and consists of the conventions for the construction, interpretation, and use of an architectural view.
- A view may consist of one or more *architectural models*. Each such architectural model is developed using the conventions and methods established by its associated viewpoint. An architectural model may participate in more than one view.

In this paper, we focus on the stakeholder concerns related to system evolvability and the corresponding views that can address them. As part of our research on the evolvability of large software-intensive systems [16], we observed that suitable architectural views are important assets to facilitate system evolution [11, 12]. Such views help practitioners to understand the existing system, to plan and evaluate intended changes, and to communicate them to others.

In particular, we are interested in *execution views*, which consist of a set of models that describe and document *what a software system does at runtime and how it does it*. The term runtime refers to the actual time that the software system is functioning (during testing or in the field). Obviously, it is very important to understand this runtime behavior of the software, but in practice documenting it often does not receive enough attention. Thus, our particular focus is to support practitioners in how to construct execution views for large and complex software-intensive systems. Such systems often have a heterogeneous implementation and consist of multiple processes, each with multiple threads, deployed across several computers.

In our initial work, we constructed an execution view of an existing large software system [2], which addressed specific stakeholder concerns. However, a development organization of such a large and complex system has several stakeholders with numerous concerns. Therefore, the organization needs to be able to define a number of execution viewpoints addressing the needs and matching the characteristics of its particular system. To achieve this, an organization may either reuse the predefined viewpoints available in the literature (e.g. [3, 5, 11, 14]) or define new ones.

In this paper, we present an approach to define execution viewpoints to address the requirements of a specific organization developing a large and complex software-intensive system. This approach includes the identification of the organization's requirements (in terms of concerns related to system evolvability and development activities) and the definition of a set of specific execution viewpoints. The organization's requirements are derived from interviews with key practitioners. The specific execution viewpoints are defined (including the customization and extension of some predefined viewpoints) to address the derived requirements.

We have applied this approach as part of the documentation of the execution architecture of a Magnetic Resonance Imaging (MRI) system. This system is a representative large and complex software-intensive system, developed by Philips Healthcare [1]. This application has helped us to identify how to use (customize and extend) predefined viewpoints and to extend our approach to construct execution views, supporting more practitioners by extending our initial set of models. We expect that other organizations and researchers can reuse our definition approach as well as some of the execution viewpoints we define here.

The organization of the rest of this paper is as follows. In Section 2, we summarize how we identified some predefined viewpoints from the literature. In Section 3, we describe the interviews to identify the requirements of a particular development organization. Section 4 summarizes the identified concepts and concerns to define execution viewpoints. In Section 5, we present a set of specific viewpoints resulting from the application of this approach. Finally, in Section 6, we provide some conclusions and future work.

2. Predefined execution viewpoints

In this section we describe our motivation to search for predefined viewpoints and the result of our search.

2.1. Motivation

To define specific execution viewpoints, we searched the literature for predefined viewpoints that address somehow what a system does at runtime and how it does. In doing so we conform with the conceptual model from the ISO/IEC 42010 standard [8]. Figure 1 illustrates the part of the conceptual model that describes the definition of specific viewpoints, the concepts of viewpoints, views and models with respect to execution. According to this model an execution viewpoint can cite a predefined viewpoint, in the sense that the former can be defined reusing (customizing or extending) the latter.

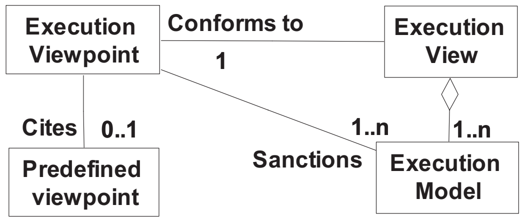


Figure 1. Reuse of predefined viewpoints for an execution viewpoint

2.2. Identified predefined viewpoints

Our search of predefined viewpoints resulted in the identification of five candidates, which are the most comprehensive and elaborated available predefined viewpoints that can be reused to define specific execution viewpoints. Table 1 lists these predefined viewpoints along with their names, as presented in the literature, and the set of concerns and system elements that their execution models describe. These predefined viewpoints can be classified into two groups based on their concerns:
The first group includes:
- The *concurrency viewpoint* of [14], which describes the concurrency structure of the system, mapping functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently
- The *behavior description* of [3], which proposes a language-independent way to document behavioral aspects of the interactions among system elements

The second group includes:
- The *deployment viewpoint* of [14], which addresses how to describe the environment into which the system will be deployed including the dependencies the system has with its runtime environment
- The *deployment style* of [3], which also addresses how to describe the allocation of components and connectors to execution platforms

In addition, another predefined viewpoint is the execution architecture of [5], which spans the two groups, describing the mapping of functionality to physical resources and the runtime characteristics of the system.

Table 1. Predefined viewpoints for execution views

Viewpoint	What it describes (concern)	System elements
Concurrency [14]	- Task structure and mapping of functional elements to tasks - Inter-process communication and state management - Synchronization and integrity - Startup, shutdown, task failure, and reentrancy	Processes, process groups, threads, inter-process communication
Behavior description [3]	- Types of communication - Constraints on ordering - Clock-triggered stimulation	Use cases, structural elements, processes, states, applications, and objects.
Deployment [14]	- Hardware required (specification and quantity) - Third-party software requirements and technology compatibility - Network requirements and capacity and physical constrains	Processing and client nodes, network links, hardware components, and processes.
Deployment style [3]	- Allocation, migration, and copy relations between software elements and computing hardware. - Properties of computing hardware, e.g., bandwidth, and resource consumption.	Software elements (processes) and computing hardware (processor, memory, disk, etc.)
Execution architecture [5]	- Execution configuration and its mapping to hardware devices - Dynamic behavior of configuration - Communication protocol - Description of runtime entities and their instances	Processes, tasks, threads, clients, servers, buffers, message queues, and classes

3. Identifying the organization's requirements for execution views

Asking stakeholders for their concerns should be a common practice, especially for choosing views [3] and identifying which views to recover from an existing system [17]. In order to identify the requirements for execution views, we conducted a series of interviews with key experts of our industrial partner using specific questionnaires. In this section, we summarize the key aspects of the questionnaire design and interviews.

3.1. Questionnaire design

The main goal of the specific questionnaires was to collect information on which execution views to create, what to describe in a particular model, how to choose the abstraction level, and how it should be described. Often, asking these broad questions to practitioners does not provide precise or useful answers. To overcome this, we designed two types of questionnaires (overview and model-specific). To design them, we summarized predefined viewpoints in the literature and our own research observations, and applied guidelines on reviewing software architecture descriptions [13].

Overview questionnaires help us to estimate the value of an execution viewpoint and get an insight on how a given interviewee may use it. To focus the questionnaire, we centered the questions on a set of existing documents containing some execution models that were authored or often used by the interviewee.

Model-specific questionnaires help us to assess how a specific execution model created or often used by the interviewee aligned to descriptions of similar models of predefined viewpoints. Thus, with each model-specific questionnaire we attached at least two models: the one used or created by the interviewee and a related example from the literature. Table 2 summarizes the group of questions for both types of questionnaires, overview and model-specific. For an example of a full questionnaire, see appendix I.

Table 2. Questionnaires structure

Group of questions	Overview	Model-specific
1. Authors and contributors	X	X
2. Creation and maintenance	X	X
3. Intended and actual users	X	X
4. Usage in daily activities (predefined viewpoint)	X	X
5. Usage in other activities (observations & experience)		X
6. Description of concerns (predefined viewpoint)		X
7. Representation language and level of detail		X

3.2. Interviews

To conduct the series of interviews, and keep them manageable and productive, it is necessary to identify a set of representative practitioners. We initially involved two stakeholders of the development organization who are actual consumers and producers of execution views. First, a senior designer who documented an execution view in the past using as a main reference the 4+1 View Model [10] aiming to support the analysis of the system performance. Second, an architect in charge of architecting and designing software interfaces for system-specific hardware devices. Later, we selected additional stakeholders who were mentioned as major contributors or actual users of the chosen document for the interview, e.g., other software architects, designers, platform support engineers, and managers. After conducting an interview, we validated the collected information sending the questionnaire (with answers and comments) to the interviewee who corrected and sometimes extended the captured information.

4. Identified concepts and concerns

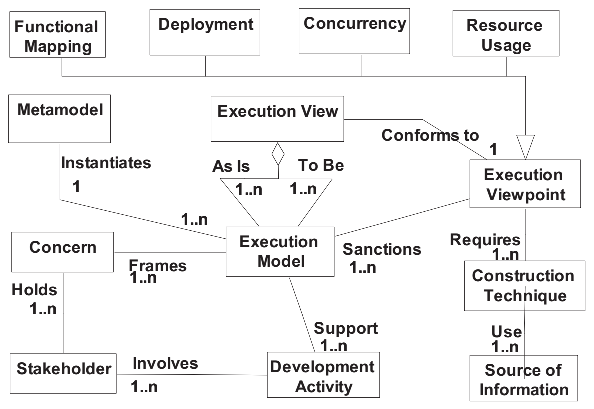


Figure 2. Conceptual model to define execution views and viewpoint

Through the series of interviews, we identified a set of concepts and relationships between them. Figure 2 illustrates the concepts and their relationships. This conceptual model is based on the model presented by the standard [8], but here we limit ourselves to execution views, models, and viewpoints instead of general architectural views, models, and viewpoints from the standard. The functional mapping, deployment, concurrency, and resource usage viewpoints are specific viewpoints that we will describe in Section 5. In addition, we include concepts such as development activity, metamodel, and construction technique to illustrate how execution views and viewpoints fit within the development organization based on the identified requirements. In the rest of this section, we focus on the descriptions of the main concepts (execution model and metamodel) and the identified major concerns related to system evolvability within development activities. Construction techniques and sources of information are presented in our previous work [2].

4.1. Execution models

From the results (answers and comments) of questions in groups 1-4, we identified that a development organization often needs to construct '*As Is'* and *'To Be'* execution models to build an execution view. The concept of *'As Is'* and *To Be'* are also applicable to models of other architectural views, but to keep the focus of this paper, we describe these concepts for models of an execution view.

'As Is' models describe the execution of the current system. These models are often created to support the acquisition of knowledge about key execution scenarios or the interactions between key system components. A *'To Be'* model describes the execution of a system that does not yet exist. Such models are typically created to design and evaluate one or more alternatives for a future system and to communicate the chosen alternative to the implementers. After implementation, a new *'As Is'* model can be created and compared to the chosen *'To Be'* model. Since nowadays a system is rarely ever designed from scratch but is typically based on existing systems (i.e. Brownfield site [6]), it is often a good idea to construct a *'To Be'* model by modifying or taking as a reference an existing *'As Is'* model.

4.2. Metamodel of system execution elements

When identifying the information needs of the practitioners, we found it very useful to describe the various elements that play a role in system execution in a metamodel, which defines a number of concepts that occur in the execution models. Figure 3 shows such a metamodel with system execution elements and relationships between them. We developed this in our earlier work [2] and validated and refined it during the interviews. Most predefined viewpoints (see Table 1) also use several of these elements, e.g., processes and threads, to create execution models. Our metamodel extends the concepts of the predefined viewpoints, including elements and relationships to address the organization's requirements that we identified to construct execution views of a large software system. The particular extensions that we introduce are elements such as execution scenario, task, software component, and activity. These extensions are meant to cope with three major issues: complexity and size of the system, explicit links with other system views, and analysis of resource usage. In section 5, we describe these extensions in more detail in the discussion of the identified viewpoints. We also provide a detailed description of the elements and relationships of this conceptual model in [2].

Note that the metamodel does not apply to an individual execution model, but is *shared* among the execution models. In this way, it indicates important relationships between the models and can help to establish consistency among the models. We expect that using a single, shared metamodel not only in the execution views but also across all architectural views may contribute significantly to their mutual consistency.

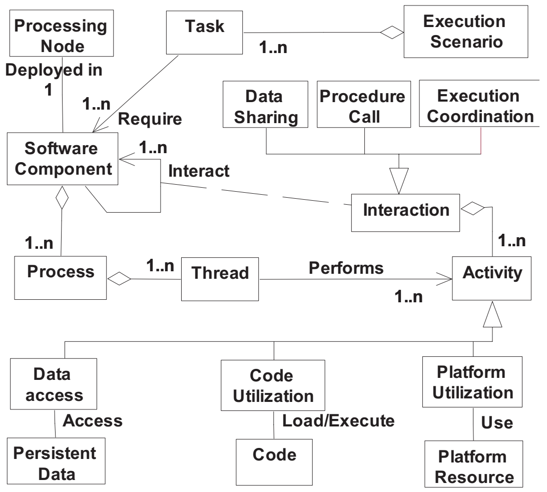


Figure 3. Metamodel of system execution elements

4.3. Concerns related to system evolvability

Based on the result of questions in groups 2-5, we found that the construction of execution models is a goal-driven and often problem-driven activity to evolve an existing system. This means that the concerns of the stakeholders relate to the activities they perform within a given development project towards specific goals. The major stakeholder's concerns and the development activities that need support of execution views are listed in Table 3 and elaborated in the following paragraphs:

- *System understanding:* In addition to the result of questions in groups 2-5, our own observations helped us to identify two aspects of how an execution view supports acquisition of system knowledge. On the one hand, execution models support system-specific education and training of new developers. Often new developers are exposed to execution models before they can start reading and writing code. This practice helps new developers to create a mental model of the overall system, the system components they develop, and their relations (dependencies) with the rest of the system components. On the other hand, *'As is'* execution models help *all* practitioners to constantly refresh, validate, and extend their mental models, in particular to support system corrective maintenance activities that aim to improve the existing run-time structure and manage unpredicted system behavior.

- *Project planning:* Practitioners need to construct *'To be'* execution models to support two particular activities. On the one hand, these models are needed to distinguish and analyze the difference between considered alternative or future architectures and designs that aim to improve quality attributes such as reliability [15], dependability, and safety [7]. This is important, as it is often not obvious how the realization of the alternative design may affect the structure and behavior of the system at runtime and therefore influence other system quality attributes. On the other hand, as we described in Section 4.1, execution models are necessary to describe the overall system structure, its components, and their interactions that make up the system functionality of interest. Often system components are mapped to development units within or outside the organization. Thus describing the involved system components enables the identification of the involved units, and therefore the planning and budgeting of responsibilities, if possible, as a downstream process.

- *Communication:* Another goal of describing the architecture of a software system is to support the communication between system stakeholders. In particular, we identified that besides the mental models that practitioners may have, they need explicit evidence in a common language (i.e. diagrammatic representations of execution models) to supports three links of communication within the development organization. First, execution models are useful to transfer technical knowledge of the system design and implementation. This supports the communication of designers and developers with architects and managers. Second, execution models are needed to describe how the system uses third-party components at runtime. These models will enable the communication of development units (external or internal) with customer designers, developers, and testers. Third, execution models are needed to describe how the software system interacts with and uses the resources of its runtime platform. These models will enhance the communication of the design and implementation units with the (internal or external) unit supporting the system runtime platform.

- *Conformance of design and implementation:* Large and complex software-intensive systems have strict constraints on their non-functional properties such as reliability, safety, and performance. Ideally, the architecture and design should describe how to achieve those requirements, but often the implementation deviates from these requirements at runtime. This usually happens when the implementation uses third party or off-the-shelf components, facilities provided by the implementation technology and the runtime platform, such as dynamic loading of shared libraries, plug-in mechanisms, and mechanisms to manage memory access. Thus, to verify non-functional requirements and properly test the system, it is often necessary to construct *'As is'* execution models to describe changes in the access and utilization of resources such as shared memory, shared code libraries, communication paths, and power consumption. Thus, *'To be'* models can be updated, extended, and analyzed.

Table 3. Concerns and development activities supported by execution models

Concern	Development activity
System understanding	Education and training, dependency analysis, and corrective maintenance
Project Planning	Analysis of alternative and future architecture and design.
Communication	Between development units or teams and with customers and providers
Conformance of design and implementation	Architecture documentation, verification of non-functional requirements, and testing

5. Execution viewpoints

The results of questions in groups 5-7 showed that the predefined viewpoints listed in Table 1 are useful to define execution views. However, they do not optimally address all stakeholder concerns, in particular in dealing with the complexity and size of the system, in making explicit links with other system views, and in describing and analyzing actual resource usage. Therefore, we defined four specific viewpoints addressing the requirements for the execution views. Two viewpoints are based on predefined viewpoints (concurrency and deployment) and two are additional viewpoints (functional mapping and resource usage). In this section, we describe these four viewpoints including some of their sanctioned models.

5.1. Functional mapping

The functional mapping viewpoint addresses the concern about the relation between the system functionality, system functional components, and execution elements. Thus, it shows how to describe the mapping of the runtime elements (including software and hardware elements) to the functional system components that interact together to deliver the system functionality. For a large and heterogeneous system, this viewpoint should show how to describe the mapping consistently and without being overwhelmed by the size and complexity of the system. To achieve this, the set of most important execution scenarios should be chosen and for each of these a functional mapping model should be constructed. Moreover, for each such model, the most relevant elements should be determined, so that the others can be filtered out.

The model in Figure 4 is sanctioned by this functional mapping viewpoint. It shows how the individual tasks in a scenario are supported by a set of software components and how the processes that belong to them perform activities, such as data access and code utilization. We observed that models like this one support all concerns and development activities in Table 3. For instance, functional mapping models are necessary to enable practitioners that are less familiar with execution elements to understand the system execution. Certain practitioners, such as managers and architects are typically more familiar with the functionality and the main components of the system. By contrast, designers and platform support engineers are often more familiar with processes and threads. A functional mapping model such as Figure 4 helps them to relate these concepts to other, less familiar ones.

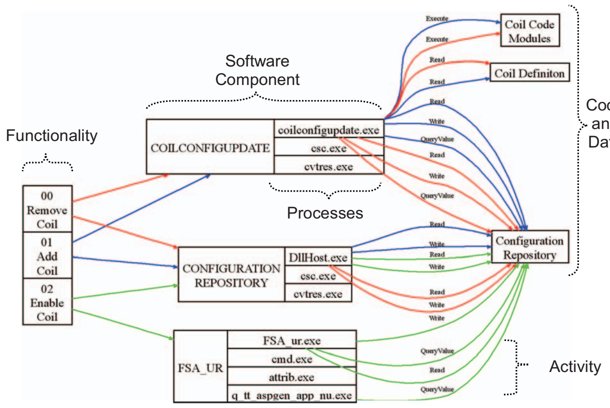


Figure 4. Execution model of the functional mapping viewpoint

5.2. Deployment

This viewpoint is a customization of predefined deployment viewpoints [3, 14]. This viewpoint addresses the concern about the allocation of system execution elements to processing nodes and the environment into which the system is deployed. Compared to predefined deployment viewpoints, the requirements that we identified indicate that such a deployment view should show additional information on three aspects (see Figure 5):

a) Detail of processing nodes: Boxes that describe processing nodes in a deployment model should describe more consistent and useful information. For instance, the predefined deployment viewpoint [3], describes that runtime platform and network models should include information about the characteristics of the processing nodes and the functional elements inside them. To do this for a complex system, while keeping an overview, we decided to represent functional elements with software components (groups of processes) thereby reducing complexity when the number of processes is large and details are not necessary. In addition, we identified that it is required to describe the allocation of important code libraries, data repositories, and system-specific hardware devices to processing nodes, making explicit distinctions between these elements and software components.

b) Detail of links between processing nodes: Often deployment models use lines to describe links between processing nodes such as network or communication lines. However, these links often lack descriptions about what they actually serve for at runtime. We identified that for an execution view, links should describe at least three aspects: the function of the link, the link's technology characteristics, and the capacity or bandwidth the system requires from the link.

c) Organization of processing nodes: We identified that the diagrammatic representation of a deployment model should resemble as much as possible the actual physical and geographical distribution of the system. This is particularly required to make some design decision explicit, such as safety issues and rules to manage the influence of physical phenomena (e.g. magnetism) on processing nodes. For instance, the diagram can indicate how processing nodes and the software components they contain can be located close to user interface elements or scanner control devices.

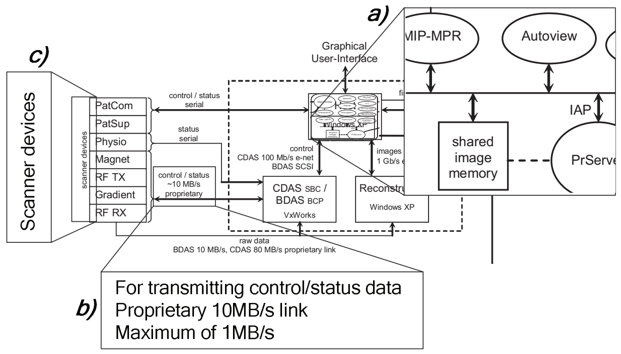


Figure 5. Customized deployment model for an execution view

5.3. Resource usage

This viewpoint addresses the concerns how to ensure and adequate resource usage. This includes the metrics, rules, protocols, and budgets that define and describe how the system actually accesses or uses available resources such as data, system code artifacts (software), and runtime platform resources (hardware and software). Describing resource usage is different from describing 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. Thus, it enables the analysis of the difference between the required (budgeted) network capacity and the provided capacity.

Figure 6 presents an execution model that describes CPU time usage. The resource usage in the scenario is described together with the activity of the two main functions (scan and reconstruction) of the system subject of our research. Resource usage can be described in terms of the processes or threads, especially when performing a top-down analysis. For instance, we constructed models like this one to analyze the difference between alternative designs of the major system functionality. There, we observed that the main activities supported by models sanctioned by a resource usage viewpoint are analysis of alternative architectures, conformance of design and implementation, and communication (in particular between designers and platform support engineers).

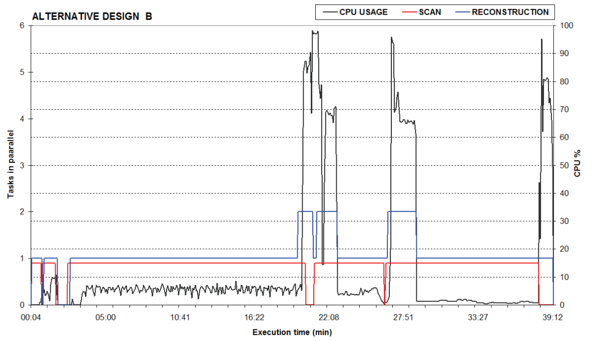


Figure 6. Resource usage models to analyze alternative designs

To construct resource usage models, it is expected that a system architecture and design should provide benchmarks and budgets for resource usage, e.g., CPU usage, but this is not often the case in current practice. Thus, this viewpoint should also show how to create and describe benchmarks and budgets to steer the construction and analysis of resource usage models. A set of *'As Is'* execution models of stable execution scenarios, preferably obtained from measurements on an actual system, can serve as benchmarks for resource usage. Based on those, budgets for future designs can be expressed as *'To Be'* models. Our experience is that this helps practitioners to agree on benchmarks and to define budgets based on specific context and actual system information.

5.4. Concurrency

This viewpoint is a customization of the predefined concurrency viewpoint [14]. For the execution view, we identify that it is required that the main concern that a concurrency model should address is the actual control flow and data flow between software components. On the one hand, control flow defines the order of execution and synchronization between software components to use or access the various system resources. On the other hand, data flow describes how data is processed and flows through software components and other system elements such as data repositories. Together control and data flow creates the runtime behavior of a system in terms of order of interactions, situations of concurrency, communication channels, and time-based interaction dependencies between processes, threads and other system elements, such as data repositories and the runtime platform.

For a large system, this viewpoint shows how to describe actual control and data flow at an overview level (software components) and a process and thread level of detail. We identified that to describe control and data flow between software components, it is necessary to define abstractions at the level of software components to represent the types of interactions between them, such as data sharing, procedure call, and execution coordination (see Figure 3). In addition, those abstractions should be mapped to actual execution activities performed by the corresponding processes or threads of the interacting software components. In this way, it is possible to construct control and data flow models at the process and thread level of detail.

Figure 7 illustrates the control flow and dataflow for a given execution scenario. In this model, control flow and dataflow is described between processes (grey boxes) and threads (parallelograms). The control and data flow edges between threads are labeled with numbers (1 to 4), which identify the tasks of the scenario.

Figure 8 shows a matrix model that describes situations of concurrency for the same scenario, but at the overview level. In this matrix model, the tasks of the scenario are distributed horizontally representing the time dimension and software components are distributed vertically. The value in each cell is the number of active threads, which might be interacting creating control and data flow.

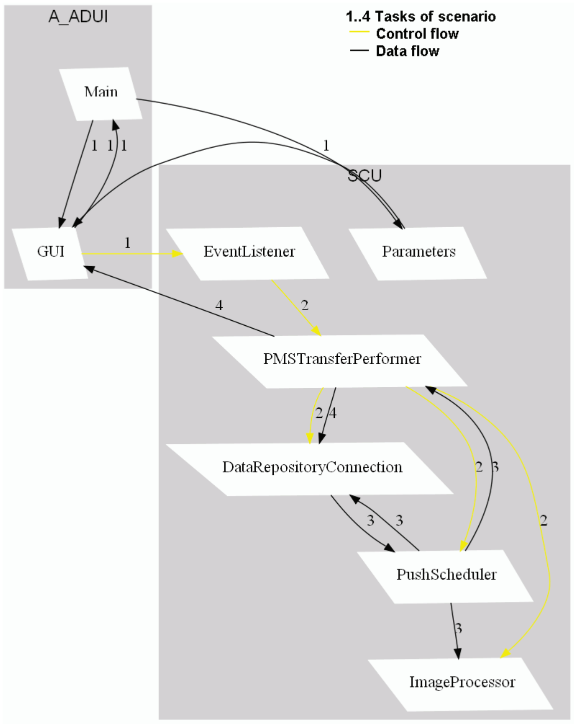
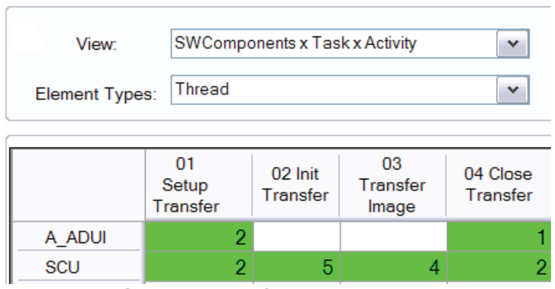


Figure 7. Control and data flow model between processes and threads



View: SWComponents x Task x Activity

Element Types: Thread

	01 Setup Transfer	02 Init Transfer	03 Transfer Image	04 Close Transfer
A_ADUI	2			1
SCU	2	5	4	2

Figure 8. Overview of concurrency between software components

Practitioners will often decide for informal representations [5, 14], but we have identified that most practitioners will associate boxes and lines with software components or processing nodes rather than processes and threads. Therefore, when constructing diagrammatic representations of concurrency models at the detail of processes and threads, it is required to use distinctive notations, e.g., using stereotypes in UML diagrams or representing threads with parallelograms instead of boxes (as in Figure 7).

6. Conclusions and future work

We described how to define a set of execution viewpoints to support the construction of execution views for an existing large software-intensive system based on the requirements of its development organization. The contribution of our approach is three-fold. First, we have shown and conceptualized how to use (customize and extend) predefined viewpoints in practice. Second, the definition approach using predefined viewpoints is a valuable complement (e.g., to scope and guide) to more general-purpose definition methods such as [9]. Moreover our approach is repeatable in other organizations and research groups. This was validated by the key practitioners involved in the approach: they confirmed that a similar approach could be used to upgrade or define other viewpoints for views of their specific system. Third, our set of defined specific execution viewpoints can be reused or cited to construct views in other organizations, because they address specific concerns that stakeholders may have.

We have shown how execution views can be constructed as useful sources of information that describe what a software system does at runtime and how it does it. On the one hand, such a view describes the actual realization of the design and implementation on the targeted platform (in *'As Is'* models). On the other hand, the view describes the desired behavior of a possible future system at runtime (in *'To Be'* models). As part of our future work, we aim at investigating and reporting how such execution views can be efficiently maintained and used to support specific architecting and design activities. Moreover, we intend to study how execution views can be related to other architectural views, with special emphasis on identifying or preferably avoiding inconsistencies.

Acknowledgments

We would like to thank the Software Architecture Team and the software designers of the MRI system in Philips Healthcare, in particular Krelis Blom and Danny Havenith. We also extend our gratitude to Rob van Ommering, Wim van der Linden, and our Darwin colleagues for their feedback and joint work.

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, http://www.healthcare.philips.com/main/products/mri/index.wpd, 2009, Visited February 2009

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

[3] 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.

[4] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, A general model of software architecture design derived from five industrial approaches, *Journal of Systems and Software*, vol. 80, pp. 106-126, 2007.

[5] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Boston: Addison-Wesley, 1999.

[6] R. Hopkins and K. Jenkins, *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*: IBM Press, 2008.

[7] G. Hunt, M. Aiken, P. Barham, M. Fähndrich, C. Hawblitzel, O. Hodson, J. Larus, S. Levi, N. Murphy, B. Steensgaard, D. Tarditi, T. Wobber, and B. Zill, Sealing OS processes to improve dependability and safety, presented at *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.

[8] ISO/IEC-JTC1/SC7, ISO/IEC 42010 Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems 2007.

[9] H. Koning and H. van Vliet, A method for defining IEEE Std 1471 viewpoints, *The Journal of Systems & Software*, vol. 79, pp. 120 - 131, 2006.

[10] P. Kruchten, The 4+1 View Model of Architecture, *IEEE Software*, vol. 12, pp. 42-50, 1995.

[11] G. Muller, CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity, *PhD Thesis*, Technical University Delft,The Netherlands, 2004

[12] G. Muller, Gaudí System Architecting - A Reference Architecture Primer, http://www.gaudisite.nl/info/ReferenceArchitecturePrimer.info.html, 2007, Visited April 2009

[13] H. Obbink, P. Kruchten, W. Kozaczynski, R. Hilliard, A. Ran, H. Postema, D. Lutz, R. Kazman, W. Tracz, and E. Kahane, Report on Software Architecture Review and Assessment version1.0, http://philippe.kruchten.com/architecture/SARAv1.pdf, Visited November 2008

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

[15] H. Sozer and B. Tekinerdogan, Introducing Recovery Style for Modeling and Analyzing System Recovery, presented at *7th Working IEEE/IFIP Conference on Software Architecture*, 2008.

[16] P. van de Laar, P. America, J. 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.

[17] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva, Symphony: View-Driven Software Architecture Reconstruction, presented at *4th Working IEEE/IFIP Conference on Software Architecture*, 2004.

APPENDIX I. Example of a model-specific questionnaire

AD Project name: Building the Execution Architecture of the MRI System	**Date:**
Domain:	**Team:**
Activity: Review of Execution Architecture Documentation	
Purpose of the activity:	

Review Session: Runtime Structure or Concurrency Models

In this session, we review in detail the section Runtime structure of the document Execution Architecture and the concurrency or behavior viewpoints from the literature. The review is centered in discussing in detail the concerns addressed by this section and some of the diagrams of the runtime structure of the MRI system execution.

1. Creation and maintenance overview:

- **Is there any specific contributor or source of information?**
- **Besides the guidelines of the 4+1 model, what triggered the creation of this section?**
- **What was the validation of the information of this section?**
- **How often is this section going to change?**

2. Intended audience: (roles*)	**3. Actual audience: (roles*)**
Hardware and Software designers and architects	* Roles within PH-MRI e.g. architect, designer, implementer, maintainer, etc.

4. Usage w.r.t. architecting and design activities

The tailoring of the list of activities is based on the overview review (previous session)

Activity	**Intended**	**Actual**	**Desired**	**Comments and brief answers on how the activity is addressed**
Communication among development units				
Conformance of downstream design and development				
Analysis & Design workflow				
Education and training				
Communication with customers and/or providers				
Analysis of system quality attributes				
Analysis of alternative architectures/designs				
Other specific activities for an improved version of this section				
Planning and creation of vision and roadmaps				

5. Usage w.r.t. specific (architectural and design) concerns addressed by a concurrency viewpoint

Concerns are collected from the literature, nevertheless we expect that the interviewee may add some specific concerns

Concern	**Intended**	**Actual**	**Desired**	**Comments or brief answers on how the concern is addressed**
Process/Thread Structure				
Show the mapping of functional elements to Process/Thread(s)				
Describe the mapping of functional elements to Process				
Explain the mapping of functional elements to Process				
Inter-process communication (Which are/why)				
State management (states, transitions, causes, and effects)				
Synchronization and integrity (e.g. mutex and shared data)				
Startup and shutdown of unit and the aggregate system				
Failure (Thread level and process crash) and propagation				
Reentrancy and priorities (critical sections, shared code)				

Notes:

6. Description and representation of information

(in the provided runtime views: Figure 1 and Figure 2)

Question	**Possible alternatives**			**Comments and brief answers**
What is the abstraction level of the diagram?	System	Overview	Detail	
Do you recognize the type or class of elements described by edges and nodes?				
Do you recognize interactions between elements?				
Do you understand what happened due to interactions?				
Do you identify the sequence of interactions				
Do you recognize what is inside of the nodes?				
Can you describe the reason for grouping elements inside nodes?				
Can you recognize the semantic of the different edges?				

Additional Comments

- *Attached models (System level, Overview level, Detail level)*