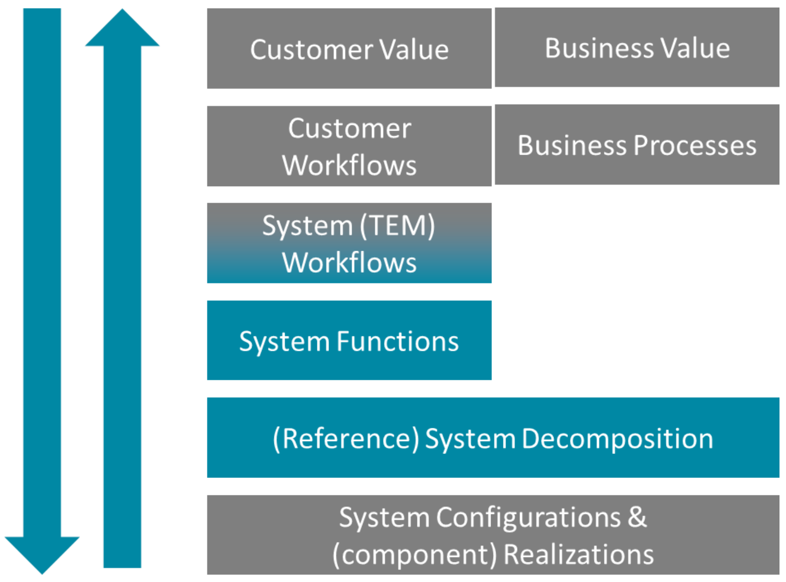
Reference Architecting

A methodology for distilling and documenting a reference architecture for complex high-tech systems



This whitepaper is a "living document". The current version (1.0) reflects the status of the research on reference architecture distilling by ESI (TNO) per December 2020. As the research continues, new versions of this document are expected to become available.

If you are interested in our research, please contact the authors of this whitepaper.
Contact information is available on page 2.

ESI

Document administrative details

Document details

Authors	Richard Doornbos Jelena Marincic Alexandr Vasenev Jacco Wesselius	Richard.Doornbos@tno.nl Jelena.Marincic@tno.nl Alexandr.Vasenev@tno.nl Jacco.Wesselius@tno.nl	
Reviewed by	ESI (TNO) Thermo Fisher Scientific	Teun Hendriks Wouter Tabingh Suermondt Sezen Acur Jamie Mc Cormack Olivier Rainaut	
Report number	2020-10183		
Project number	060.36233 060.47616	PaloAlto PaloAlto2	
Version	1.0		
Date	2021.10.21		
Document Location	TNO SharePoint	TNO Project SharePoint TNO Team SharePoint	*PaloAlto2 - Projectdossier* *RA Distilling Methodology*

Contact details TNO

Contact name	Jacco Wesselius
Department	ESI
Address	High Tech Campus 25, 5656 AE Eindhoven (NL)
Telephone	+31 6 46 119 721
e-mail	Jacco.Wesselius@tno.nl

Acknowledgments:
This whitepaper is a product of the research carried out under the PaloAlto program under the responsibility of ESI (TNO) with Thermo Fisher Scientific as the carrying industrial partner.
The PaloAlto project is supported by the Netherlands Ministry of Economic Affairs (Toeslag voor Topconsortia voor Kennis en Innovatie).

Table of contents

1 Introduction

This document results from the PaloAlto-project of ESI (TNO) and Thermo Fisher Scientific (Thermo). This project started on October 2018 and ran until December 2020. The project researched methods and processes for distilling[1] reference architectures for complex high-tech systems. To develop, apply and validate these methods and processes, the project has been performed in a close cooperation between ESI and Thermo in an *industry as a lab*-setting. At Thermo, the methods and processes have been applied for developing a reference architecture for a range of high end to mid-range Transmission Electron Microscopes (TEMs). The intent is to improve the level of re-use across the product range in order to enhance R&D effectiveness and efficiency.

A reference architecture is a top-level description of the essence of existing architectures, and of the vision of how to fulfill future business needs. As such it provides guidance in the development new architectures and helps to direct strategic design choices.

Reference architectures aim to resolve a variety of issues occurring in companies: a lack of a shared baseline about the why the what and the how, next to a lack of guidance of where to go. In the PaloAlto-project it was confirmed that these issues exist. They are often seen as misalignments in the company and expressed as a strong need for improved effectiveness and efficiency, and a strong focus on reducing cost and time.

A generic life cycle for a reference architecture in a high-tech company is shown in Figure 1. It comprises of three phases: the creation, the dissemination and embedding, and the maintenance phase. In the creation phase, *reference architecting* takes place: distilling and capturing reference architectural information and its validation and alignment in the organization. The dissemination and embedding phase entails the education and explanation of the concepts and use of the reference architecture, and the embedding in the company's processes to be able to actually use the reference architecture. The last phase is that of maintenance, in which the established reference architecture (views, models, rationales, guidelines and restrictions) is evolving and moving with the changes over time at the company's pace.

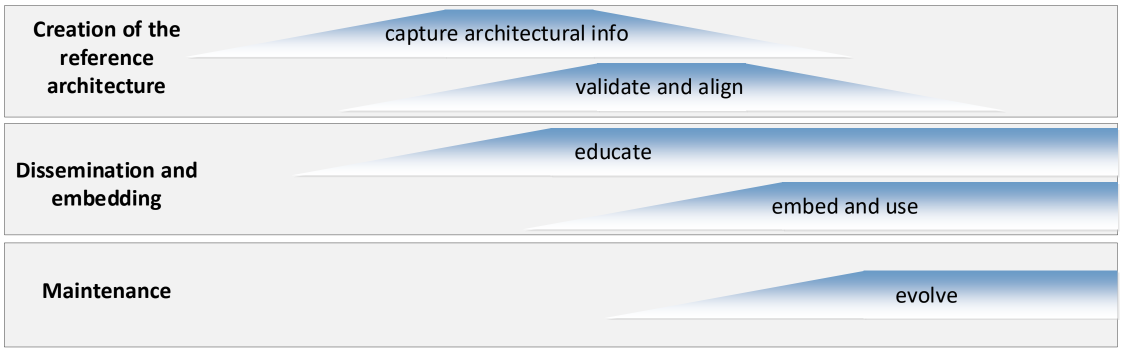

Creation of the reference architecture
capture architectural info
validate and align
Dissemination and embedding
educate
embed and use
Maintenance
evolve

Figure 1 - The generic lifecycle of a reference architecture

[1] To distil: "to get or show only the most important part of something." [79]. The word distilling has been used (and not "creating") to emphasize that a reference architecture is meant to capture the essence of the architecture of a product portfolio. This method focusses on distilling from an existing product portfolio, extension is needed for green field situations, or improve existing system architectures.

In the PaloAlto-project, an approach for *distilling a reference architecture* (the creation phase) has been developed, inspired by the CAFCR[2]-methodology [1]. Market and business *values* are main pillars of CAFCR. It provides a framework for reasoning from these values via system specifications to architectural concepts and implementation choices (and vice versa). Its principles are applied accordingly:

- The process starts with analyzing *customer and business values*.
- To clarify how customers use a system to create these values, workflows are analyzed:
    - *customer workflows* to understand the processes in which the system is used;
    - *system workflows* to understand in much more detail how the functions of the systems are used in these customer workflows.
- A *functional decomposition* is completed in order to understand and capture the system functionality. System design choices are not described in the functional decomposition.
- Based upon (i) business drivers and (ii) knowledge of the design of existing systems, a *system decomposition* is created
    - defining strategic system components and their functionality, interfaces and properties;
    - mapping all the functions from the functional decomposition onto these components.
- The analysis of the current system realizations (modules and their interfaces and dependencies) puts the strategic system decomposition in the context of current system variations and commonalities (and intended use) as:
    - a sanity check of the system decomposition and the other used system abstractions;
    - the starting point of creating an architecture roadmap towards compliance with the reference architecture;
    - the starting point identifying platform components and for creating their roadmaps.

The resulting approach has been applied and validated in the high-tech industrial context or Thermo Fisher Scientific (Thermo). This document describes the status of the methodology as of February 2021. As the research and the application/validation of the methodology continues, newer revisions of this document will be provided and updated.

The document is intended to share the developed methodology and to provide an overview to researchers, systems architects/engineers, and Research and Development (R&D) managers to understand and utilize the approach. As the document describes the results of ongoing research, ESI invites the readers to participate in applied research projects to apply, validate, extend and improve the methodology.

[2] CAFCR stands for: **C**ustomer Values, **A**pplications, **F**unctions/Features, (Architectural) **C**oncepts, **R**ealizations. These form the five main views in the CAFCR-methodology.

2 “Reference Architectures”

The term *reference architecture* needs a definition, as it is widely used for varying concepts. To understand the methodology described in this document, it is important to define some terminology first.

Before introducing the definition of *reference architecture* used in this whitepaper, the differences between *product architectures*, *platform architectures*, and *reference architectures* will be outlined; they differ in their goals, scope, subjects, and intended users. In this section the main concepts of such architectures and their main users will be illustrated.

2.1 Product Architectures

An architecture is “*fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*” [2].

When applied to a product, one speaks of a *product architecture*.

In literature more narrow definitions are used, such as “the scheme by which the function of a product is allocated to physical components” [3]. Such a scheme can include (as suggested by Laperrière and Reinhart [4]):

(1) the arrangement of functional elements;
(2) the mapping from functional elements to physical components;
(3) the specification of the interfaces among interacting physical components.

For the latter, several *kinds* of interfaces are typically seen in product architectures, including those identified by Sanchez [5]: attachment interfaces, spatial (volumetric) interfaces, transfer interfaces, control and communication interfaces, user interfaces, and environmental interfaces.

A product architecture includes the description of these functional and physical structures. The primary users of product architecture descriptions are product *designers and engineers.* They design their (sub)systems based on the allocation of functionality and on the defined interfaces.

2.2 Platform Architectures

To create a right solution efficiently and effectively, *product architects* can benefit from re-using components. *Platform architectures* can help by providing sets of reusable components.

Platform architectures are modularizations of complex systems, where certain components (the platform itself) remain stable, while others may vary [6]. A platform architecture describes which components are generic, which can be specific, and which variation points are available. It prescribes how to compose products or product families out of platform components and specific components.

The reuse of components can help to achieve a competitive advantage by reducing life-cycle costs and creating customer value by reducing training costs, easy extending/upgrading, etc. For instance, a shared car platform enables a car manufacturer to benefit from components reuse across models;

platforms shared across brands and manufacturers create an even larger opportunity for reuse (see [7] for an overview of car platforms). Similar examples can be found in many industries.

Specifying stable interfaces is essential for platform architecture as it allows easy substitution of components on one or both sides of the interface. [8]. As platform architectures differ in their degree of formalism [9], not all interfaces (e.g., from the kinds listed above) will always be completely described.

The *contents* of a platform architecture depend on the purpose and includes components, interfaces, and composition/synthesis rules. While an R&D platform includes *assets* for re-use in product development, a commercial platform helps to create customer-specific product and solutions.

Users of platform architectures are product and solution architects. *Product architects* can create their architectures by mixing and matching stable components and interfaces. Similarly, *solution architects* create solutions for their needs, for instance, using commercial platforms.

2.3 Reference Architectures

When creating a platform or a product line, the platform architect can benefit from guidance provided by a reference architecture. *Reference Architectures* (RAs) capture the essence of existing architectures, and the vision of future needs and evolution to provide guidance to assist in developing new architectures [10]. For the purpose of the methodology described in this document, the need of a reference architecture is stressed to capture the technical *strategy* of the company to relate long-term customer value to long-term business value. For this, it shall provide a common lexicon and taxonomy, common (architectural) vision, rationales, and the modularization and complementarity context. This approach is consistent with the way DoD defines a reference architecture as "an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions" [11, p. 3].

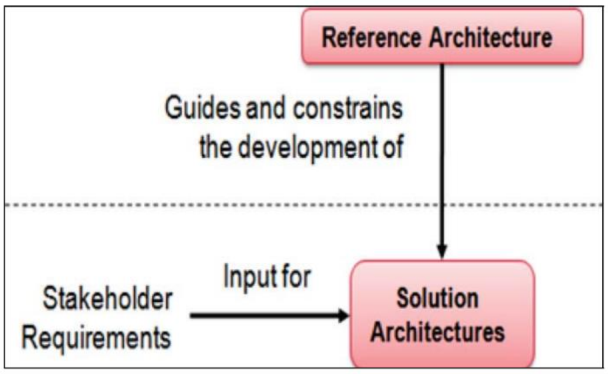


Figure 2 - The Purpose of a Reference Architecture according to DoD (from [11, p. 3]). Solution Architectures in this figure corresponds to product architectures in section 2.1.

A reference architecture does not aim to completely cover a domain or address all architecture details and aspects. There may be multiple RAs within the subject area where each represents a different emphasis or viewpoint of that area [11]. One example is a reference architecture for communication, which can co-exist with another one focusing on decisions for systems safety.

A reference architecture addresses a specific need with a suitable level of details and appropriate abstraction (from concrete to generic). It should not be a complete description, but only capture the core aspects to give *sufficient* guidance for architectural decision making. As a consequence, reference architectures can be very different in different domains and organizations.

Reference architectures guide and constrain *architects* on their *choices of interface types* and instantiations of multiple architectures and solutions. For instance, *platform architects* can use a reference architecture to create and improve components in a way that fits the larger strategic view better; *product architects* can use guidance to address a specific product quality, for instance, security.

RAs focus on strategically important aspects to assure that systems will deliver their core values. Such values differ per domains (e.g., priority of performance or safety), organizations (for instance, innovator vs fast-follower), and business models (consider service or product models).

The *contents* of the reference architecture include aspects relevant to Technical architecture, Business architecture, and Customer context. [10]. These may include elements of strategic purpose, vocabulary, principles, methods (e.g., as technical guidance), patterns, technical architecture (high-level system-wide architecture blueprint), standards, and tools [11]. In practice, a shared description of business architecture and customer context are often missing and the shared assets, e.g., reusable components, often get a lot of focus. This often results in an architecture describing the shared assets (or platform).

	Reference architecture	Platform architecture	Product Architecture
Goals	Guides and constraints instantiations of multiple architectures and solutions	Assists in plug-and-play construction of product architects; Maximizing reuse	Basis for elaboration in design and engineering
Contents	Guidance for architecting decisions in technical, business, and customer contexts	Stable components, interfaces, composition rules	functions of a product allocated to physical components, interfaces
Direct user	Platform architects; Product and solution architects	Product and solution architects	Designers and engineers
Inputs	Business (mission, vision, strategy), customer, technology contexts	product, technology process, production system supply chain [8], reference architectures	Reference and platform architectures; Technologies; product, market, and customer needs.
Approach to Interfaces	Guidance to specify and elaborate interfaces	Overview of stable interfaces, composition rules	Interfaces specified for elaboration

Table 1 - Features of reference, platform, and product architectures

Reference architectures use a number of concepts to address company needs on a level that helps architects. Figure 3 shows generic elements of a reference architecture and its drivers. This diagram was constructed within the ArchWay-project[3] [12, 13] from multiple sources, including [11, 14, 15, 16]. It serves as a generic model for reference architectures and their context.

[3] The ArchWay-project was a project under the responsibility of ESI (TNO) with DAF as the carrying industrial partner. The ArchWay-project was supported by the Netherlands Ministry of Economic Affairs (Toeslag voor Topconsortia voor Kennis en Innovatie).

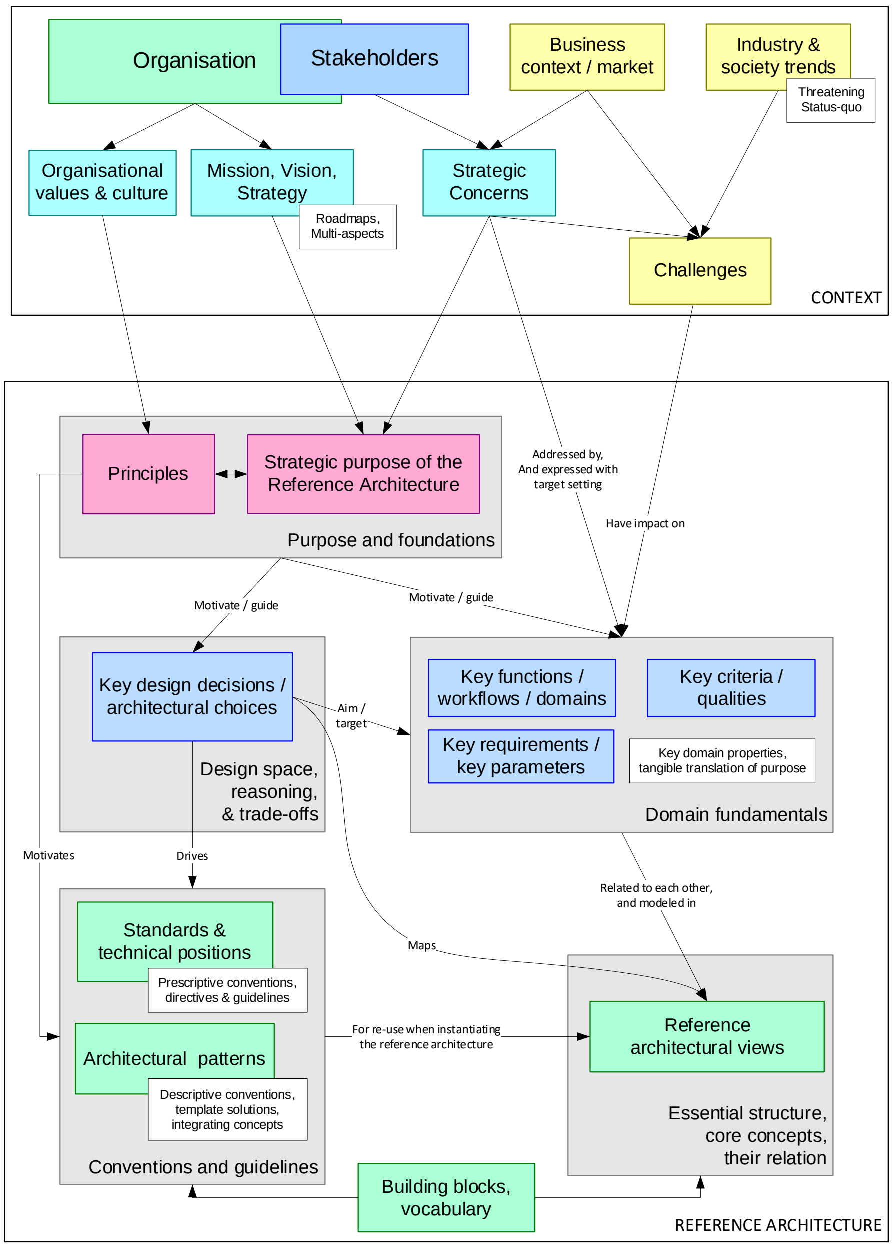

Organisation
Stakeholders
Business context / market
Industry & society trends
Threatening Status-quo
Organisational values & culture
Mission, Vision, Strategy
Roadmaps, Multi-aspects
Strategic Concerns
Challenges
CONTEXT
Principles
Strategic purpose of the Reference Architecture
Purpose and foundations
Addressed by, And expressed with target setting
Have impact on
Motivate / guide
Motivate / guide
Key design decisions / architectural choices
Design space, reasoning, & trade-offs
Aim / target
Key functions / workflows / domains
Key criteria / qualities
Key requirements / key parameters
Key domain properties, tangible translation of purpose
Domain fundamentals
Motivates
Drives
Related to each other, and modeled in
Maps
Standards & technical positions
Prescriptive conventions, directives & guidelines
Architectural patterns
Descriptive conventions, template solutions, integrating concepts
Conventions and guidelines
For re-use when instantiating the reference architecture
Reference architectural views
Essential structure, core concepts, their relation
Building blocks, vocabulary
REFERENCE ARCHITECTURE

Figure 3 - Overview of a generic reference architecture and its context [17]

Summarizing, a reference architecture guides platform and solution architects, who *instantiate* it based on the specifics of the task at hand. Reference Architectures bring value to the company by capturing key value and business drivers and relating these to architectural choices. The focus on a common language and validated patterns brings consistency of implementations and facilitates adherence to standards. As a result, *Product architects* can effectively work on multi-site, multi-supplier, and multi-vendor product creation [18]. *Platform architects,* in turn, can focus on the platform strategy for dynamic environments that impose many changes on the products [19].

3 Scope and Purpose of Reference Architectures

In the previous section the meaning of the term *reference architecture* was discussed, including the roles of its main users. In this section, this will be elaborated by addressing the purpose of reference architectures in more detail.

3.1 Scope

As discussed in the previous section, various types of reference architecture can be identified. The methodology presented in this document has been created for defining a reference architecture *in a company* for *products lines*[4] in:

- software-intensive cyber physical systems (CPS);
- the high-tech industry;
- multi-disciplinary, distributed product development groups.

Examples of such systems are electron microscopes, professional print systems, baggage handling and warehousing systems, wafer scanner systems, etc.

3.2 Purposes

Within this scope, reference architectures are created for several purposes:

- The primary reason for creating a reference architecture is to create the architectural foundation to **assure the company's business needs**. For this purpose, it needs to express the rationale of strategic architectural choices in terms of the company's business needs and strategy.

  Examples of such strategic choices manifold to, e.g.:
  - interfaces that are standardized across the company to enable reuse of core technologies owned by the company, or to enable easy replacement of components by cheaper versions to meet price targets in specific market segments;
  - the identification and definition of variation points to support product diversification and the creation of customer-specific product variants in market segments where *customer intimacy* [20] is the business strategy.

- Another reason for creating a reference architecture is to **capture the essence** of the way in which key customer values are realized within the company (for the given range or products or solutions). For this purpose, it describes strategic choices that are shared across the company or across the product range. This way, it limits the architects' design freedom to assure commonality in product designs. In many cases, the aspects to be standardized are selected based on (i) business aspects as discussed above or (ii) good engineering/architecting practices.

[4] To note, this methodology differs from methodologies for introducing specific system qualities for multiple products in several domains. The latter was adopted by the SECREDAS project [39] that developed a reference architecture for introducing privacy and security into safe automated systems from several domains.

- The third reason for creating a reference architecture is to **consolidate essential technical and architectural/design knowledge** within the company. In many companies, core [5] knowledge is hard to capture. At the same time, this knowledge is one of the most valuable assets for the company. When experienced people leave the company after building up a large base of knowledge, new engineers entering the company lack insights of the system design and means to quickly collect them. A well-described reference architecture can form a stable basis for transferring knowledge.

  In a similar way, a reference architecture can be used to capture a **standardized terminology** in a company (related to its core knowledge). Having such standardized terminology can improve communication within the company significantly.

  Care should be taken that design choices and strategies that were valuable in the past may prove to be a burden in the future. The company runs the risk that the reference architecture becomes an overview of many details without connecting them to customer values or business drivers. Limiting the level of details and recording the rationales and assumptions for past design decisions is needed to avoid this pitfall.

- Finally, reference architecture can be created to serve as an architectural framework for decision-making: **trade-off analysis**. To make architectural choices in product or solution designs, a framework is needed to evaluate the impact of these choices on customer values and business drivers. A reference architecture can, if well setup, provide a framework for doing such trade-off analyses, as it connects architectural decisions to customer values.

Summarizing the above discussion of the purpose of a reference architecture, a reference architecture should:

- cover *strategic* architectural choices (decomposition, standardized interfaces, other standards, standard technical solutions, etc.) and their rationales in terms of their contributions to the company's business strategy (as will be discussed in the next section about factors that influence a reference architectures);
- capture architectural knowledge and the corresponding vocabulary that is crucial for the company and for communication inside the company;
- capture the *essential* qualitative and quantitative relations between the business and technical domains to support trade-off analysis for design choices in products, solutions and platforms based on the reference architecture.

It is important to note that a reference architecture is not a complete description of a product or platform architecture; it does not give all the details needed to construct a product. In fact, the reference architecture should only contain these elements that constitute the essence of the architectural choices, the key choices that assure the technical strategy to meet the company's business goals. Since it constraints design freedom of the architects of the platform/product architectures, the designer of the reference architecture should assure that not too much is prescribed

[5] The knowledge about the company, its target markets, its users and their way of working, the products, technologies and architectures and their rationales, the manufacturing processes and technologies etc. (and about the relations among these) that allow the company to be successful.

in it; every unnecessary constraint in the reference architecture removes design options for the product architect, hence potentially eliminating promising product architectures/designs.

Key decisions to be made for a reference architecture:

- which constraints to add and when to stop?
- does adding additional constraints/design choices to the reference architecture contribute to realizing the business goals of the organization?
- or does it unnecessarily constraint product architects and designers?

In addition, note that a reference architecture will evolve and that it will require maintenance, the more details the architects put in the reference architecture, the larger future maintenance effort will be.

Making the right decisions about these aspects, requires the architect to reflect on the main concerns to be addressed in the reference architecture.

4 Reference Architecting Concerns

As discussed above, a reference architecture is created for a specific purpose and in a specific context. For a reference architecture to be fit for use, it has to aligned with various aspects of its environment. For the reference architecting methodology described in this whitepaper, the concerns to be addressed in a reference architecture are linked to the BAPO-framework[6] [21, p. 100, 22, 23] (see Figure 6 on page 24). This framework puts the architecture in its context: Business, Processes and Organization, where the business drives the architecture and where proper alignment between the architecture and the organization, and its processes is crucial:

- The **business strategy** of the company drives the creation of its reference architecture. In which markets is it active? What products or services does it provide? Does it serve multiple markets or market segments with different characteristics, e.g., high-end and entry-level market segment? Does it want to grow its position in specific markets or market segments? How does it intend to be successful in these? Is its market strategy based on product leadership, or on operational excellence or on customer intimacy [20]? What are the company's core values in the market? Which aspects of the company's product offerings are critical to its business success?

  Based on such considerations, the architects make decisions about the aspects that should be explicitly addressed in the architecture. Should they focus on standardization for efficiency and cost reduction? Or should they focus on flexibility to be able to offer customer tailored products and solutions? The architects have to decide how to assure that future product designs based on the reference architecture will deliver the company's core values in line with its business strategy.

- In accordance with *Conway's Law* [24], a product design follows the (communication) **structure of the organization**. This is one of the aspects to consider in a reference architecture: although company structures are not carved in stone, it will have stable elements. Architects will design their products structures in ways that are influenced by (and at best consistent with) the organizational structure, geographic dispersion, social boundaries, distribution of tasks, core competencies, equipment etc. This way it will also influence the reference architecture, e.g., by decomposing the system in subsystems based on the geographical dispersion of the company and by standardizing interfaces between subsystems that are delivered by different departments and external suppliers and partners.

  Reference architectures are created with the intention of having a stable foundation for product architectures. In dynamic markets, company structures can change far more frequently. This also shapes the reference architecture as it requires the company to make choices that will have impact on the reference architecture: which elements of the organizational structure are (expected to be) stable and which are not? What would the impact of organizational changes be on the architecture and can a reference architecture be defined that can be stable in times of organizational turmoil?

[6] BAPO stands for: Business, Architecture, Processes and Organization. Initially called *BOPA* in [77]

Finally, the reference architecture is influenced by the way of working in an organization, by its **processes** in sales, research and development, manufacturing, solution and product delivery, service and support and many other processes in a product's life cycle. For instance, other process choices can be identified that can result in specific concerns that the architects could decide to address in the company's reference architecture.

In a similar way to the organizational aspect discussed above, the architect has to consider which architectural choices are essential to assure that the architecture is well-aligned with the company's main processes (which are probably even more subject to change than the organizational structure). An overview of process considerations and their relevance for architectural choices is given in Table 2 (on page 17).

Another methodology for identifying concerns to be addressed by a reference architecture is the CAFCR-framework [1], referred to before. Here, the main driver for creating an architecture is *customer value*. The framework stresses that, in order to define an architecture, architects have to create insight in the way their products and solutions enable the customer to create added value. The value of the product/solution is captured in the way its *quality* is defined: its fitness for use. [25, p. 2.8]

The architect has to identify the essential customer values that the company wants to deliver in its target markets. The reference architecture contains the core elements (e.g., technology choices, standards, structures and interfaces) that the company chooses to assure that products designed conform its reference architecture will deliver the intended customer value, in line with its business strategy.

From the other side of the CAFCR-spectrum, technology and product realization aspects can be an important driver for the reference architecture. Here the architect can identify where technology is moving and decide how the reference architecture should be prepared to support future technology changes to enable future value creation, in line with the company's business strategy. Architects can identify which technologies are stable and which evolve quickly. In the reference architecture, they can incorporate standards to decouple technologies that evolve quickly from the stable core of their systems.

From the BAPO and CAFCR framework, categories of drivers and concerns for a reference architecture have been identified, as discussed above. In a concrete case, companies (and their reference architect) have to choose their own specific drivers and concerns for these categories. Making this choice explicitly is at the core of the methodology described in this document.

Process Choices	Potential relations with architecture
The choice made in R&D for an agile way or working, or for a fully pre-planned waterfall approach;	In case of agile R&D processes, integration of modified components (hardware/software) will be a continuous process. This gives ways to manage their evolutions, but it also requires these integrations to be done easily. In these cases, the reference architecture could identify which components are expected to be modified and re-integrated frequently and define their interfaces more stringently. The reference architecture could also provide tools and methods to do these integrations quickly and without risk of disrupting the complete R&D process in case of malfunctioning of these components. If a waterfall model is used, the continuous integration processes can be less prominent of even completely absent. In these cases, it is also important to assure that system integration can be done without major issues, but given the much lower frequency of these, the architects might decide not to give concerns related to continuous integration a main role in their reference architecture.
The way field maintenance is done, the role of remote access to systems;	If the service processes of the company rely on remote monitoring, and remote service of their installed base, one of the key concerns to be addressed in the reference architecture is: methods, standards and tools for secure access to systems in the installed base. For a company which service processes are fully focused on physical access to the system (with potentially valuable contact moments with its users) this specific security aspect is not a main concern.
The way manufacturing and installations are done, are systems pre-assembled and tested in the factory before shipments or does the company do drop shipments where individual parts "meet for the first time" at the customers site?	For a company performing on-site installations of components that are drop-shipped to the customer's site, the ease on installing the system can be a concern that influences choices made in the reference architecture. The reference architecture might describe testing and diagnostic interfaces for all components as they are shipped in the logistical processes. Understanding the manufacturing, transport and installation processes enables the architects to define appropriate standards in the reference architecture to assure the quality and predictability of on-site installation activities. For a company that pre-assembles its systems in the factory before shipment, such standards could be less important, not giving them a place in the reference architecture at all.

How are purchasing processes organized, is R&D in control of the purchased parts or can supply chains be changed without R&D being in the loop?	If R&D is deeply involved in the purchasing processes and in the selection of parts, the R&D organization can assure that key components are selected that fit well in the system designs and that key components are not modified without need. If, however, R&D is less actively involved in these processes, one could consider how to limit the impact of new component selection. Furthermore, the components that are crucial to the performance of their systems that cannot be changed without their active involvement and recertification need to be identified. The reference architecture could in this case address such aspects specifically.
If the company delivers customer specific systems, how do sales and R&D align to assure that systems offered to customers can be delivered efficiently in terms of costs and time-to-delivery? Does the company use engineer-to-order processes (R&D develops dedicated for a customer) or does it apply configure-to-order processes (the sales department offers solutions that can be created by configuring/composing customer-specific systems from standard products)?	If a company decides to apply configure-to-order processes to avoid huge R&D costs, the reference architecture has to address concerns like platform design, mechanisms to create customer-specific solutions from platform components, etc. [7] For a company that choses to apply an engineer-to-order approach, this is less relevant. In either case, a company that delivers many customer specific product variants needs processes for configuration management that are well supported by its reference architecture. If a company only delivers a fixed set of standard products, such concerns are likely to have a less prominent influence on the reference architecture.

Table 2 - Process considerations and their architectural relevance

[7] In 2019, ESI and Vanderlande investigated the impact of platform-based development and configure-to-order processes on product/platform architectures and on the interaction between sales and R&D. An overview of the status of these discussions can be found in a presentation of Maarten Verhoeven of Vanderlande at a webinar of ESI about MBSE [76].

5 State of the Art / Literature Overview

5.1 Examples of Reference Architectures

The term Reference Architecture is well-known in software-based systems. In literature, one can find many examples of reference architectures for software systems, especially for information systems. Some of them are focused on architecting systems based on specific paradigms and technologies, resulting in reference architectures for cloud computing [26], or IoT (internet of things) [27]. Some focus on a certain architectural style, such as service-oriented architecture [28], without going into the details of the domain in which they are applied.

Reference architectures for specific software application domains do exist, for example for automotive domain [29] or farm management systems [30]. Also in some mature software domains such as compilers and operating systems strictly defined architectures are used [31], although they are not often not explicitly called *reference* architectures.

When it comes to methods to design software-based systems, the automotive domain, as a highly standardized domain, applies architectural frameworks and guidance. A notable example is the collaboration of Volvo with researchers [16] to establish automotive architectural frameworks. To support the increasing role of the software in the system development, they introduced a Continuous Integration and Deployment viewpoint; to enable rapid communication with suppliers and flexible development they introduced the Ecosystem and Transparency viewpoint; finally, to acknowledge the fact that the cars nowadays are part of a network of communicating systems they introduced the System of Systems viewpoint.

As explained in the ISO/IEC/IEEE standard for architectural descriptions [32], an architecture framework is a coordinated set of viewpoints, conventions, principles and practices for architecture description. When such framework is available, and well established in a particular domain (e.g. automotive) its reference models form a strong foundation for designing reference architectures that companies create for their own products and product lines. As described in the previous section, the methodology described in this report focuses on creating the latter.

In the *system* architecting domain, literature does not offer an abundance of reference architectures and architectural framework examples. There are reference architectures for embedded control systems, such as ASRA for avionics [33], but they refer to software systems and related computing architecture on which the software runs and on which the data is stored; they do not cover a whole system or a product. As an illustration, an extensive literature review [34] of architectural guidance for automotive systems, found 13 software- and only 2 hardware-aimed references. As noticed in the article of Cloutier [10], the lack of examples of the reference system architectures indicates lack of maturity of the term Reference Architecture in the system engineering domain.

It comes as no surprise that there is even less written and researched on the *methods* for creating such system reference architectures. Research at ESI (TNO), including the work presented in this paper, aims to fill this gap [35].

5.2 On methods for designing reference architectures

Of the existing literature on the methods for reference architecting, the Reference Architecture report of the U.S. defense department [11] presents a solid basis. Although it does not give a prescriptive method on how to design a reference architecture, it outlines its focus and context. The report offers a standard definition of "reference architecture" and defines in detail the elements it should provide; they are *strategic purpose*, *principles*, *technical positions*, *patterns* and *vocabulary*. Even though the document starts by referring to information systems and provides in its appendices solely the examples of software systems, it is aimed at DoD *systems* architecting and as such it is generic enough to be applicable on a system level, beyond software.

The DoD report positions reference architecture in the context of DoD *enterprise architectures*. The report also provides a conceptual model of enterprise architectures. The model describes different factors and aspects that drive the creation of reference architectures, without going into the details of the relations between these aspects. The conceptual model of the DoD enterprise cannot be applied in its entirety to the context of high-tech systems. Besides entities such as Tools and Technical Standards, the model describes domain specific concepts such as different DoD departments. In a previous project of ESI [36] on reference architectures in automotive domain, this model has been adapted and extended (see Figure 3). It expands the reference part of the DoD model, and adapts the domain specific concepts of defense to those of high-tech systems.

Another document that presents valuable guidelines for reference architecting, is Gerrit Muller's Reference architecture primer [1] based on his CAFCR architecting method and on the results of the System Architecture Forum [37] that gathers prominent practitioners in the field [38]. It provides guidelines on the abstraction level of a reference architectures, as well as why and when they should be used and what they should contain. The ideas of Muller's white paper have been further detailed and solidified in the co-authored article that discusses concepts relevant for reference architecture design by Cloutier et al [10].

Delving into answering the question "How do you create a reference architecture?", it is described and modelled in the process of knowledge discovery by Muller and van de Laar in [23] . Using their analysis tools and skills, researchers (without domain knowledge) worked together with domain experts to find basic reference architecture structures. The value of this article is that it performs a "meta" analysis of the reference architecture discovery process. The steps of this process, once made explicit, can lead the design of a reference architecture and help navigate through plethora of details.

A reference architecture can focus on system qualities, such as the approach described in a EU project [39] where ESI led the developing of a reference architecture for introducing security and privacy to safe automated systems. That reference architecture provided a common language for developers of secure solutions [40].

The methodology described in this whitepaper extends the concepts presented in the standard of DoD and guidelines of Cloutier. It focuses on *distilling* reference architectures for high-tech systems. These systems are complex not only because of their intrinsic, functional (necessary) complexity, but they also have built due to time-to-market pressure, a huge legacy, leading to, so called avoidable complexity in software [41], but also in the system as a whole. This presents a huge obstacle for fast innovations and introductions of new features.

6 The Structure of a Reference Architecture

In the previous sections, the definition and the purpose of reference architectures have been discussed. Before discussing the methods and techniques to create a reference architecture, first a way to structure a reference architecture will be described: what are the building blocks of a reference architecture, how are the related and how do they contribute to its purpose?

6.1 Introducing a 6-Layer Model: relating it to CAFCR

Elaborating on the model in Figure 3, a way to organize the information of the Reference Architectural Views (bottom right) will be described in this section. Inspired by the CAFCR methodology [42], a 6-layer structure to organize the information in a relevant way has been setup, see Figure 4.

In the 6-layer structure the relations between the layers are made explicit. Models at a layer are linked to models at the next higher and next lower layer. This structure enables top-down and bottom-up reasoning (as indicated by the arrows in Figure 4) by creating *chains of reasoning*:

- *top-down* they explain how customer and business values are realized: via a workflow description, the involved functions are described, which in their turn are mapped on system components and realizations;
- *bottom-up* they express how properties of components (realizations in the lowest layer) influence values, via their impact on functions and workflows.

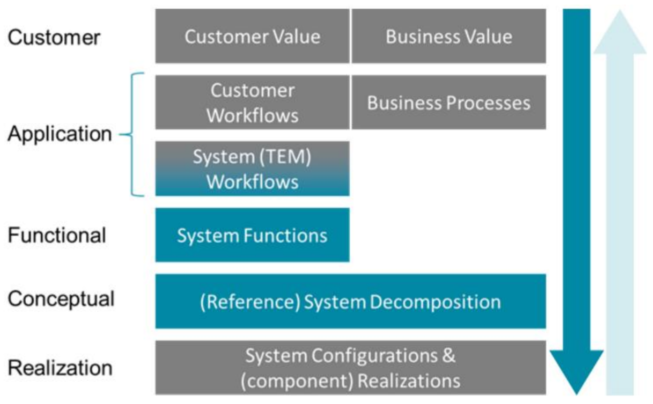


Figure 4 - The 6-Layer reference architectural model. On the left, the CAFCR aspects are shown for reference.

6.2 Purpose of the 6-Layer Model

The purpose of the 6-layer value model is to show the relations between the domains of business, users and customer, and that of the architecture and product realizations.

This model can be constructed for a product architecture, for a platform architecture or for a reference architecture. In the context of the reference architecture, the business and the costumer domain reflect the future and answers to the questions such as “Where do we see ourselves in 5 years?” and “What is our strategy to get there?”.

The model relates the reference architecture, functional and system decompositions to (i) customer and business values and workflows on one side, and (ii) to system realizations on the other side. It captures the rationale for architectural decisions and visualizes the relations in an abstract way, understandable for both technical domain experts and business and marketing managers.

Each layer abstracts from much more detailed and complete models, descriptions, formulas, and documents. The most important and most difficult task here is finding the right level of details – where product architecture should be complete and detailed, a reference architecture should only describe and constrain [43] the elements of strategic concern. Deciding which elements to describe and which constraints to apply are a matter of architectural strategy closely linked to the business strategy of the organization.

In these choices the business values (right side of the 6-layer model) drive the description of the system decomposition. Without this drive from the business side, the system decomposition will typically closely follow the functional decomposition. As the reference architecture serves a long term strategy (see Section 3), it is essential, before or while creating it, to revisit the business model, and to have clear decisions on:

- what the business strategy is;
- what the strategic customer values are that should be addressed in the targeted market segments and applications;
- how the business will organize itself to add the customer value and earn profit at the same time.

These will guide the decisions regarding what the reference architecture will describe and to which level of details.

6.3 Layer 1: Customer Values and Business Value

Customer Values

For a business, it is crucial to understand the goals, wishes and ambitions of the customers in the markets targeted by the company. Based on this knowledge, the company can create a good Customer Value Proposition – a convincing argument that shows the value the customer gets when using a product or a solution the company offers. In this section a definition of customer values will be given. Based on that, the notion of the Customer Values Layer will be explained and in addition to the relations with the other 6 layers (in Figure 4).

The concept of customer values is addressed in the field of *marketing research* and in a branch of modern system engineering called *Value-Driven Design (VDD)*. The latter field has adopted the value definitions from the former. An overview of customer value definitions in the marketing literature, including the authors' own definition, can be found in an influential article of [44] . Most of these definitions recognize that values are subjective measures, a *perception* of what is received for the price paid. To provide a customer with a convincing evidence of the worth they get when their requirements are fulfilled, values are expressed in monetary terms [45].

When applying Value-Driven Design (VDD) techniques, system engineers analyze the trade-offs between certain features or improvements, from the perspective of customers and how much they ‘value’ them [46]. At the heart of this approach is a value function that calculates the monetary value of different designs.

Both the approach of marketing research and that of VDD offer concepts and means to compare products, features, services in terms of how much the customers are willing to pay for them. The techniques developed in VDD aim to map customer needs to technical product characteristics. The methodology described in this whitepaper does not aim to replace these techniques. Rather it complements them and benefits from them if they are applied in the organization.

Based on [47], *customer value* is defined as **technical, economic, service and social benefits** that customer receives in exchange for the price they pay.

In *reference* architecting, those values will be collected that will be, or continue to be addressed in the future as part of the business strategy. They will be grouped into: (1) Values for customer in different markets; (2) Market segments; (3) Applications in such a segment and (4) Types of users doing something in these applications (workflows): e.g., researchers versus service centers. An example of the categories made to evaluate Customer Values for TEM electron microscope are given in the Table 3.

Markets	Life science, Material science, Semiconductors
Applications	SPA (Life science), Tomo
Market segments	Universities, Service centers, Industry
Types of users	Experienced operators who know the microscope inside and out, Operators using the microscope as a push-of-a-button tool

Table 3 - Categories of values (for TEM)

With the information gathered from the customer or from marketing, the values will be structured and the following aspects will be analyzed: (1) the end goal of the customer (if they are a business themselves: their differentiators at the market for their customers) and (2) the relative importance of each of these values.

To structure the values within each of the categories in the *Customer Values layer* and to relate the customer goals with the system architecture, the “5 why’s” technique [48] can be applied. For example, if the customer in a research institute specialized in Single Particle Analysis (SPA) is asking for a certain image resolution of the microscope, one can ask why this is valuable to them, because usually behind a technical requirement there is another goal, such as to make an image of a protein, and behind this value can be another goal, for example to be a pioneer in biomedical imaging of newly discovered viruses. With the end goal known, the company can decide in which innovations to invest and can also make an assessment how much a customer is willing to pay for an improvement that makes it even better or closer to their end goal.

The company strives to create the design choices that support all the values, but if a choice needs to be made, one would like to know which values do prevail. To cover this a relative importance is assigned to each of the values, on the scale of Low, Moderate, High and Very High. For example, the electron microscope is designed to have the best possible image quality and the best possible speed, but for the research institute in the Life Science market, the image quality has priority over the acquisition speed. This balance could be different in other markets (e.g., in semi) or for other types of users (e.g., for service centers).

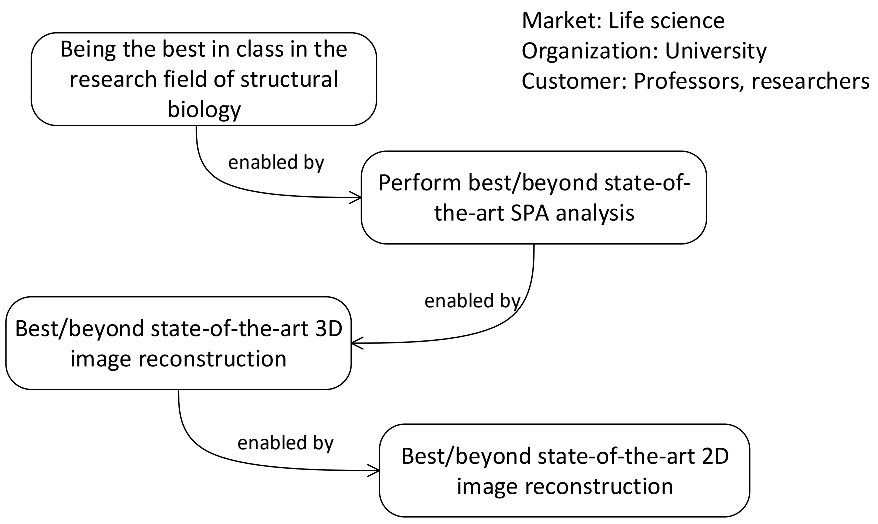

Market: Life science
Organization: University
Customer: Professors, researchers
Being the best in class in the research field of structural biology
enabled by
Perform best/beyond state-of-the-art SPA analysis
enabled by
Best/beyond state-of-the-art 3D image reconstruction
enabled by
Best/beyond state-of-the-art 2D image reconstruction

Figure 5 - Structuring customer values from top-level goals to system properties that are related to customer workflows

The customer values are structured in a causal diagram (like Figure 5), relating the customer values to customer and system workflows, system functions and system decomposition. These relations guide decisions on reference workflows, abstract function and system decompositions, and reference interfaces.

In the system and product design phase, these relations will be the basis for creating models for quantifying the design decisions in terms of the values they bring to the customer.

The most important relations between the technical domain and the values are made explicit, and by doing this the intuition of the architects or their back-of-the-envelope calculations can be checked. By making the most important relations explicit and visualized, a means is provided for communication between system architects responsible for different system aspects, and between system architects and marketing managers.

Business Values

The BAPO-model [49][8] explains the various dimensions of software development and its role in industry. This model can be extended to systems and positions the architecture with respect to business, process and organization, see Figure 6. The arrows indicate an order to traverse the concerns: e.g., the business concerns influence the architecture, which in turn influence the processes and organization. In practice the concerns will be traversed iteratively and in an order that fits the challenges at hand.

[8] Initially called BOPA [77]

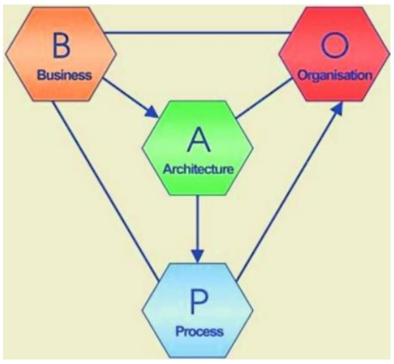


Figure 6 - The BAPO concerns, from [49]

In this section one of the business processes will be considered: the product development process (in which the reference architecture is mainly used) which should ultimately provide value to the business, so it is important to know which aspects are influencing this business value. The reference architecture, once developed, has a large impact on 'how the business is run'. Examples of aspects that provide business value in the current context are the amount of component reuse, the development efficiency, the independence of development teams, the configurability of product (families), the reduction of variability (e.g. the number of different products), and the standardization of internal processes.

The information concerning the business value is described in the Business Value section (top-right) of Figure 4.

6.4 Layer 2 and 3: Workflows

The details of how the customer achieves their goals are addressed in the Customer Workflows layer. Here, the "what" of the customer is translated into the "how". And in this "how", the system is part of a system-of-systems, in which multiple (diverse) systems and users work together to produce the desired result.

The question here is – which aspects of the system-of-systems (SoS) do need to be considered and described when relating the system to the customer values? The SoS has its own architecture; it expresses a behavior, it transforms the inputs into the end result; it has a structure, as well as internal interfaces through which the systems communicate. Depending on what the customer value is, one may need to describe some of these aspects or a combination thereof.

At the customer, a high-tech system typically performs a task or a number of tasks as part of *a (business) process*. This process is called *a customer workflow*. It will be described in the **Customer Workflow layer**. A large number of customer values depend on properties of these workflows. One needs to understand:

- customer workflows: the overall workflow of a customer to realize a value;
- system workflows: the part of customer workflows in which the system of interest is being used, describing the workflow of the system and clarifying the contribution of the system to the customer's value creation.

Customer Workflows

A **workflow** or a **workflow process** is a predefined set of steps and their partial ordering. Each step has pre- and postconditions: the preconditions must be met before the step is executed and the postconditions will be fulfilled after execution of the step. Every step is executed by *resources*, such as people and machines.

The ordering of steps can be sequential, in parallel, conditional or iterative [50], as shown in Figure 7. In sequential ordering, one step is followed by another, that is, upon completion of one step, the next one starts. In parallel ordering, steps are executed in parallel and when both steps are completed, the next step can start. Conditional ordering means that there is a choice to be made between the steps, and only one can be executed. Finally, a step may have to be executed one or more times, which is described as iterative ordering.

Every step can be either a compound step containing another workflow process, or an elementary step. An elementary step is a basic unit of work, which must be a sequential set of primitive actions executed by a single resource.

In value modeling, the value is a function of the steps' properties, which depend on the properties of the resources. To model the effect of different resources, workflow steps are composed until elementary steps are reached that are executed by the resources in question. Workflow steps are decomposed to the level of the desired granularity of the resources. The result is a hierarchy of workflows, a tree of steps, as sketched in the example in Figure 8.

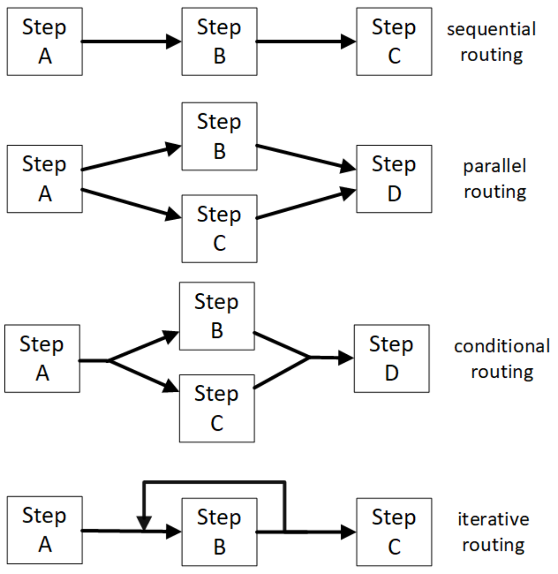


Figure 7 - The routing of step in a workflow – typical workflow patterns redrawn from [50] with "Task" renamed to "Step.

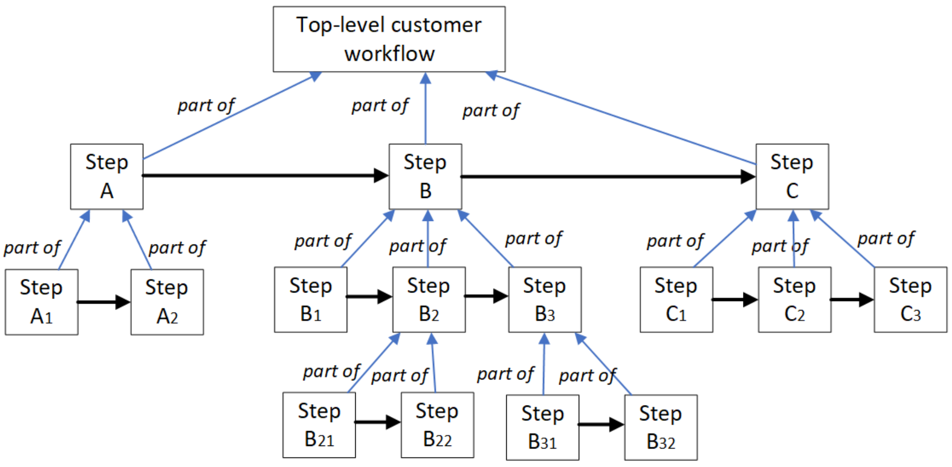


Figure 8 - A tree of workflows, decomposed until the effect of the resource to be modeled is reached.

In reference architecting, a small number of typical, representative customer workflows is chosen and their relationship with a selected set of customer values will be modeled.

System Workflows

The steps in the customer workflow that are executed by the system of interest can be further decomposed into system workflows.

High systems such as electron microscope, a printer, a wafer scanner, transform a workpiece (a sample, a paper, a wafer) into a desired result. What happens inside a machine, how the workpiece gets transformed, can be described as a sequence of steps that the system performs. This sequence is called a system workflow. It will be described in the **System Workflows layer**.

Here the same rules apply as for the customer workflows, there are different orderings of steps as shown in Figure 7 and every step can be an elementary or a compound step. This way one can show the decomposition of the top-level customer workflow, in a tree of “sub-workflows”, which are the steps in the main workflow, and being workflows themselves, consisting of other steps/workflows.

The same formalism can be used to describe customer and system workflows. No distinction is made between the customer workflow layer and system workflow layers because of a framework or a formalism change. The distinction between the two layers is there to distinguish between the systems-of-systems view of the customer, and the view of the system itself. Changes in the system workflow can result in changes in the overall customer workflow increasing the customer value.

Some steps in the top-level customer workflow may be executed with systems that one is not architecting. In such cases, the architect still might want to understand their influence on the customer values, in which case these workflow steps can be decomposed further.

An example of modeling values and workflows is the timing performance of creating an image of a number of viruses. To model this value, the architect needs to understand the basic process at the customer, such as sample preparation, and sample transport. Furthermore, the architect will model the sequence in the microscope that handles the sample and creates images. A simplified of such customer and system workflows is sketched in Figure 9.

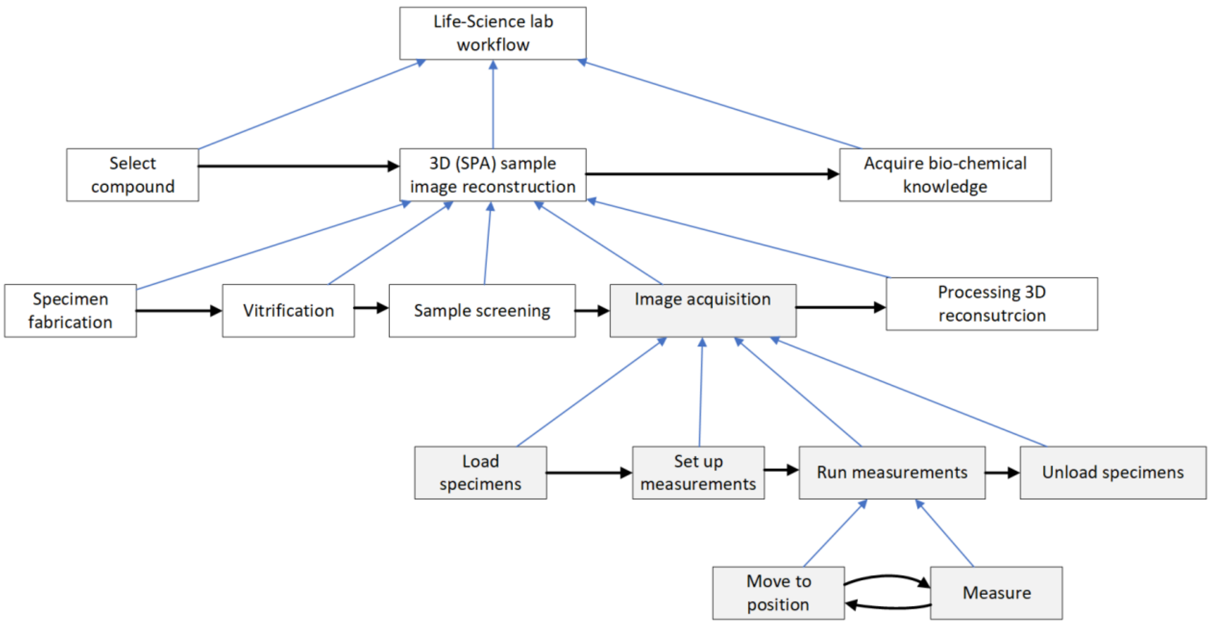


Figure 9 - Simplified workflow in Life-Science lab to acquire knowledge of a virus. Blue arrows show "part of" relations, black arrow show "precedes" relations in a sequence. The workflow within the system itself is shown with steps in grey color.

Business Workflows

When decomposing processes in a company analogous to the workflows of the customer, one expects to see that business workflows have relations with aspects of the system decomposition.

In product development, various business processes are executed simultaneously and overlapping. For (development of) the reference architecture it should be clear in which process it plays a role. To give some examples: in road mapping new products, decisions about strategic outsourcing, and (sub)system or feature development. As noted in the table in section 4, in case of application of a guideline (which is part of the reference architecture) the improvement of a certain component cannot be done unconstrained. The reference architecture defines that it should, e.g., be backwards compatible, or use extensions of a mechanical interface in a prescribed way. Another example is that the system decomposition might be optimized for independence of development teams, i.e., by matching team composition, and competence distribution with the system decomposition. Especially for multi-site situations this may prove to be critical.

The information concerning the business processes is described in the Business Processes section in the 6-layer reference architecture structure (second layer, on the right of Figure 4). The actual technique to create such a workflow is part of the future work. This includes workflows that go beyond product creation process.

6.5 Layer 3: Functional Architecture

The functional architecture is a view on the system which provides a description of *what* the system does. In general, it is a crucial central perspective that links to many other system views. For instance, a system workflow step involves the *execution* of a system function (or of multiple functions).

A *system function* is a transformation of inputs (information, energy, material) into outputs, performed by the system-under-study. Each function can be decomposed into smaller functions, that can be described separately, which is typically called a functional decomposition. The functional decomposition is usually expressed graphically as a tree of functions (as in Figure 10). Tools may also use tables to describe these.

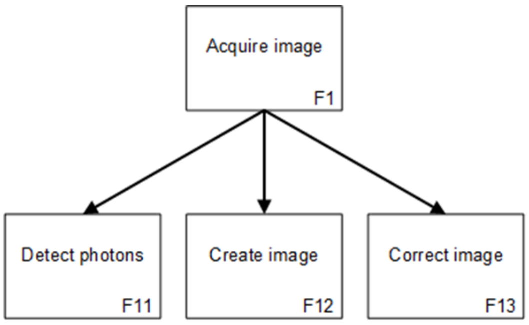


Figure 10 - Function Decomposition of a Acquire image function into subfunctions.

One can continue to split functions until a high level of detail has been reached. In case of developing a reference architecture it is important to capture the functions that are common for all products in scope. When decomposing functions further and further, one will reach a level of detail for which the products are different. This is the typical boundary indicating when/where to stop modelling, because from here onwards things are not common anymore. Depending on the purpose of the reference architecture, going to this detail level might not be needed.

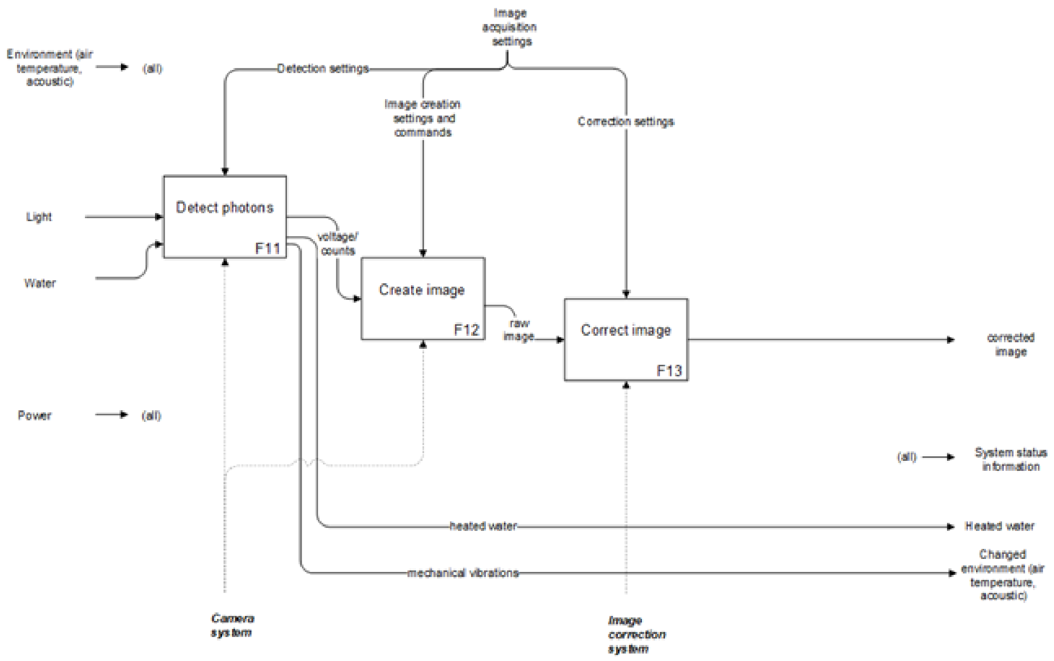


Figure 11 - The functional flow graph of a 2D Imaging System (using IDEF0 [51, 52])

When an explicit description of inputs, outputs and control inputs is added to this decomposition, it becomes very useful for understanding *what* a system and its parts are doing. When inputs and outputs are added, a more complex graphical network will be needed, see Figure 11. This figure (using the IDEF0 graphical language [51, 52, 53]), describes the decomposition of Figure 10 in more detail; it describes how the subfunctions are connected and how the overall function is delivered by the combined subfunctions. In the method described in this whitepaper, the richer decomposition style of IDEF0 will be used instead of the decomposition trees (see the section 7.5 for a more detailed description)

6.6 Non-Functionals

At the start of this section, it was stated that the functional view of a system describes *what* the systems do. It is crucial to also address the *how well* the systems do this. These aspects characterize the performance of the system (in the broad sense of the term *performance*). To differentiate them from the functional view on the system, they are often called *the non-functionals* (to emphasize what they are not), or the qualities (see, for instance ISO25010 [54]) or -ilities.

Depending on the customer and business values, non-functionals typically describe properties that are primarily related to the customer/business value layer and the workflow layer or secondarily to the functional decomposition layer:

- properties of system workflows as discussed in section 6.4 - System Workflows (page 26):
    - *productivity*: how long does it take to perform a workflow and how many people are needed for it? how often can a workflow be performed per year, taking into account, for instance, system down-time for maintenance, repair, consumable replacement, cooling, etc?
    - *effectivity*: what is the ratio between successful and unsuccessful executions of the workflow?
    - *safety*: how safe is the execution of the workflow?
    - *security*: how secure is the (confidential or sensitive) data/information acquired during the workflow?
    - etc.

- properties of business workflows as discussed in section 6.4 - Business Workflows (page 27):
    - how efficiently can the systems be diagnosed and serviced?
    - how efficiently can R&D processes be executed, e.g., how much reuse between systems is possible and how much benefit does that bring?
    - how much effort is needed to upgrade a system with new functionality?
    - how much effort is needed to update a system with, for instance, security updates?
    - how much effort is needed to create customer-specific products?
    - what are the costs of manufacturing, transporting, installing and calibration a system?
    - what the cost of good shipped?
    - what are the expected cost of resolving future component/subsystem obsolescence?
    - etc.

- properties of the functions used in such workflows:
    - how fast is the function performed?
    - how reliably is it performed?
    - how easy/difficult is it to use the function, is dedicated training or specific expertise needed?

  - what is the cost of (performing) the function, e.g., in terms of the number of people needed to execute the function or in terms of the usage of consumables or even in terms of system wear and tear?
  - what is the quality of the output of a function, e.g., image quality?
  - etc.

When products are delivered for multiple market segments, the target values for these non-functionals typically determine in which market segment a specific product fits. For an entry-level system, it might be acceptable to deliver images with a lower resolution, when the system is less expensive and very easy to use and to service. For a system targeting the high-end segment the balance can be completely different: extreme resolution and a very rich set of features are needed for a system that is more complex to use and that is much more expensive. The reference architecture and the system decomposition that is part of it create the stable structures across those markets and provide the variation points to create products that provide the right level of performance for the right price. The architects need to identify: (i) the essential non-functionals, (ii) their target value ranges and (iii) how these can vary between systems targeted for specific markets and applications.

In the next section the System Decomposition layer will be described. The functional model describes which functions need to be allocated to a sub system in the system decomposition. The non-functionals determine which decomposition that completely captures all functions is the best, this requires the architect to perform trade-off analyses to balance potentially conflicting non-functionals.

In the current description of the Reference Architecting methodology, modelling the non-functionals is not yet fully addressed. Some aspects are covered by what has been depicted as the cross-layer arrows in Figure 4 on page 20 and in the related models discussed in section 6.9.

6.7 Layer 4: System Decomposition: (Sub-) Systems and Interfaces

Once the functionality and key non-functionals have been captured and connected to market-values and workflows, the architect has to make key design decisions: how will the functionality be implemented and how will the non-functional qualities be realized? Decomposing the system into sub-systems and their interfaces lays the foundation for the reference system architecture.

A system decomposition consists of:

- a collection of subsystems, where each subsystem has a collection of defined interfaces;
- a collection of connections between the interfaces of these subsystems;
- an allocation of functions to the subsystems;
- an allocation of the flows between the functions to connections between the subsystem interfaces.

Making such a system decomposition, the architect has to assure that:

- each element of the lowest-level function decomposition has been allocated to exactly one subsystem;
- each flow between functions has been allocated to exactly one connection between subsystem interfaces;
- all flows map on interfaces between subsystems with equivalent subsystem connections;
- all interface connections are between compatible subsystem interfaces.

When the architect has completed this task, he has created a “correct” system decomposition that realizes the system functionality specified by the functional decomposition.

Note that the systems in the reference system decomposition are abstract - they generalize from product or platform architectures. As such, they have less hierarchy levels in the system tree and they do not necessarily describe all the interfaces in full detail.

System Hierarchy

One of the main guiding principles in system decomposition is the hierarchy principle - the system is decomposed into sub-systems; each sub-system is decomposed into its constituent sub-systems. The process continues until the desired level of modularity is achieved. Figure 12 shows a hierarchical decomposition of a system S into subsystems. The hierarchy allows for separation of concerns, i.e., one can for example talk about how S1, S2 and S3 interact with each other without going into the details of S23.

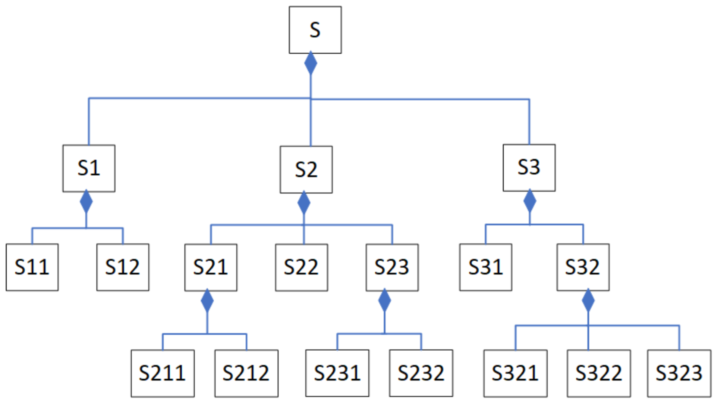


Figure 12 - One possible decomposition of the system S. The filled diamond relation is borrowed from UML notation to describe “part/whole”, composition relation.

In addition to dividing the system into its constituent parts, the architect decides on *how* these parts interact. Best practices offer design patterns to choose from for some decomposition problems. For example, a group of subsystems realizing a communication function, may be arranged in a pipeline structure or in a service-oriented structure.

In reference architecting, the system decomposition will be kept abstract enough to describe multiple realizations (implementations) of the system. For example, in the TEM microscope reference architecture, there is the Stage sub-system, that describes all realizations. The decomposition is done until adding a next layer does not add any value for addressing the strategic concerns of an organization. Typically, the architect continues until reaching a level where the connection can be made to concrete realizations in product architectures.

Since a reference architecture is meant to cover a range of products, the reference system decomposition will also describe the structure of a range of products and their variants. Following a *150% modeling approach*, the structure represents all realization of all possible functions, even some of them might be optional for certain market segments and applications and may not be present in a range of product configurations once they are manufactured.

In practice, for organizational reasons, the sub-systems in the decomposition tree are referred to as modules, components, units. In terms of the nature of the decomposition process, each sub-system is treated as a system. So, one could say that a system's constituent elements are also systems. They can all be described with a concept of a *block* that will be introduced in the following sub-section.

System interfaces

An interface is a boundary where, or across which, two or more systems interact. As Muller [1, p. 47] succinctly puts it - every time a decomposition is performed the resulting components will be decoupled by interfaces.

System engineering textbooks classify interfaces into the following types: (1) physical connection, (2) energy flow, (3) mass flow and (4) information flow. Within each category, sub-categories (aspects) are defined that refer to engineering disciplines designing them. An example of more detailed interface categories that still fit into the four main categories is as follows:

- functional flow realization (a system interface realizes one or more functional interfaces);
- workpiece interface describing the
  - electro-mechanical aspects of material flow and
  - the protocol of the flow;
- communication protocol;
- data interface described with a conceptual diagram that represents the data and its structure being exchanged;
- electrical, electro-mechanical, mechanical interfaces as more refined categories of physical interface;
- volume interface, specifying the physical dimensions of the physical interface;
- software interfaces describing software aspects of the interface;
- control interfaces describing the flow of the control;
- interfaces used in different machine modes (service, production, testing mode);
- quality aspects assigned to aspects above (e.g. data speed).

A system interface definition typically covers different aspects from the list above. Also, an interface realization typically realizes multiple kinds of flows.

In the reference architecture not every single interface not all its aspects will be described. The architect describes the (guidelines for) interfaces and those aspects that are strategic and need to be standardized. Take for example the HT system described in Figure 16 and the high-voltage interface. To achieve reusability, reference architecture points that this interface is standard and refers to the (reference) design document, that details it out. For example, it can say something about the power specs including quality as well as the physical position of the port. Another example is that of a CAN bus, in which the communication interface has been standardized, and there is a rationale why this brings a value.

To achieve separation of concerns and modularity, the system should have clearly defined external and internal interfaces and apply the information hiding principle introduced by Parnas [55]. In Figure 16, a power supply interface for HT tank has been standardized for all TEM platforms to satisfy power requirements. The architect can discuss interactions with the power unit, without going into the details of how the power is distributed internally in the system.

External interfaces also reflect the organizational aspect of the reference system architecture. If the two teams/groups departments are responsible for a (sub-) system X, then the external interfaces represent not only a “contract” between the system X and the system with which it interacts, but a contract between two organizational units.

Together with the system decomposition comes budgeting the system specifications, like power consumption, heat dissipation, system weight, system costs, EMC, vibration. These are not necessarily assigned to already defined functional interfaces, but they are equally important to specify.

Language

This subsection describes main concepts of the graphical notation and informal language for describing system decompositions. The notation is based on already existing languages, such as SysML 1.x [56]. The concept of Blocks will be introduced, followed by two views: (1) structures of blocks and (2) connections between blocks.

A system decomposition consists of two diagram types: *block hierarchy* (structure view) and *block connections*. These two have to be consistent in block naming and block hierarchies. In a tool that contains all the views, these relations can be traced.

BLOCKS

A *Block* represents an abstraction of (a component of) a system, like for example a ‘Camera system’ or ‘Retraction’. A Block can describe any type of entity within a system of interest or the system’s external environment. This document shows abstract System Decomposition Blocks as blue boxes (as shown in Figure 13) to differentiate them from the Blocks that are drawn in realization diagrams (in Section 6.8).

A block as an SD element represents a concept, a class, that can have different realizations. For instance, a ‘car’ block may imply a vehicle in general, not necessarily a specific car (e.g., a Golf that leaves the factory). In TEM SD, one has a “Stage system” that can be realized as two Stage types, each with specific features. A specific configuration selects one of the two realizations.

Blocks are used to describe sub-systems on all levels of the decomposition tree. For example, they refer to large systems, such as energy delivery system, or a safety system built into it, consisting of a sensor and a switch; and sensor and a switch are also described with a block.

Side note: Comparison with SysML
The SD blue boxes reflect SysML’s notion of Blocks that describes the structure of an element or system. In SysML block can reflect a System, HW, SW, Data, Procedure, Facility, or Person. The description can include categories (e.g., constraints and values) chosen by the architect.

Figure 13 shows an example of a block named ‘High Voltage System that can be further elaborated in SysML.

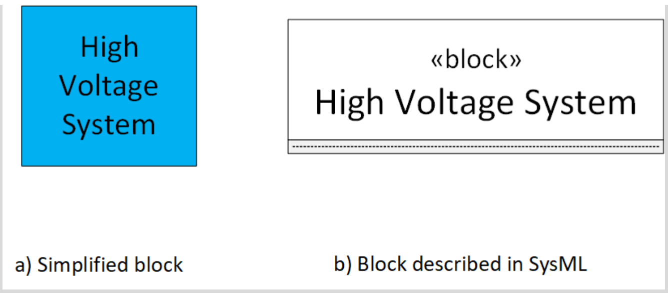


Figure 13 - A comparison of a blue block and SysML's concept of a Block

STRUCTURAL VIEW: STRUCTURES OF BLOCKS

A structural decomposition, such as one shown in Figure 14, describes the system decomposition tree, and its sub-trees. This view can help stakeholders with all sorts of technical and organizational tasks. It can, for example, support the system requirements decomposition and specification, it can be used in an FMEA process. Or, it can be used to organize development teams.

In the diagrams, the “part/whole” relations is shown with a non-directed line. The hierarchy is represented by the position in the diagram, where the subsystems are positioned below the system. An example is shown in Figure 14. To visualize the relations in the decomposition tree, the blocks in the diagram are extended to the length taken by the sub-systems.

The logic behind the diagram is as follows: blocks on higher levels are composed of lower-level blocks. For example, Figure 14 shows that ‘High Voltage System’ consists of a ‘HT generator’ and ‘HT cable’.

As a whole, these blocks realize a particular system function (Provide Source Energy in this case). The tree structure allows picking the level needed for analysis, organization, realization. A system engineer can for example define High Voltage System’s external interfaces, without addressing its internal elements.

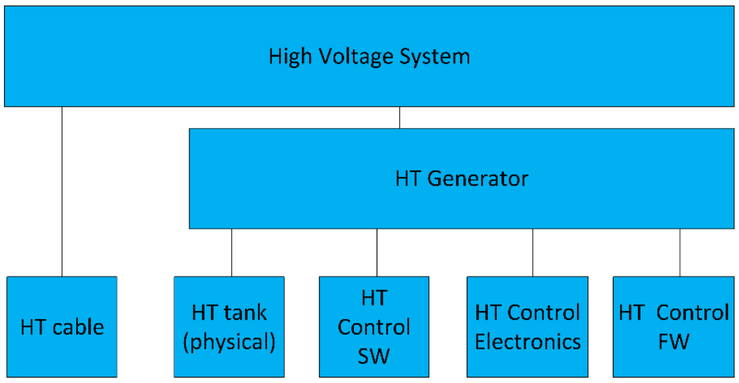


Figure 14 – Reference Architecture System Decomposition of TEM microscope's 2D Imaging System

Side note: Comparison with SysML.

The lines connecting the blue blocks that denote "part/whole" (consists of) relation are described with an association with filled diamond. The relations in SysML also allow to depict when an element is optional with the number of elements from 0 to 1 (0..1) as well as the number of systems.

Figure 13 shows an example of a block named 'High Voltage System that can be further elaborated in SysML.

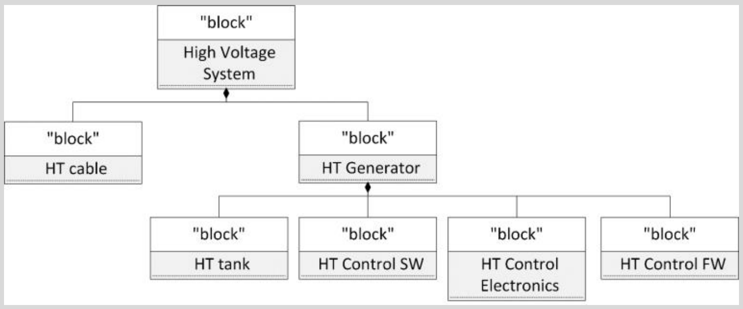


Figure 15 - High Voltage system decomposition described using SysML

CONNECTIONS BETWEEN BLOCKS

This view complements the system structural view. It shows how the subsystems are connected and how their internal and external interfaces are organized. Its syntax and semantics are described using the example diagram in Figure 16:

- This view represents the system tree in the "box in a box" manner, with different shades of blue for readability.
- The interfaces are represented as small rectangles (▭) located on blocks' boundaries. Note that what is internal interface on one abstraction level, becomes an external interface on a lower abstraction level.
- When an interface has a unidirectional flow, the direction is indicated using arrows. Bidirectional flows are represented with lines.
- Color coding can be used when the architect wants to emphasize a specific aspect. For example, in Figure 16 :
    - obsolete interfaces (legacy that has to be maintained) are colored orange.
- Notes can be used to clarify definitions, for example when there is a new realization, but the old interface still has to be maintained, this can be explained in a text connected to the interface with a line.
- How every external interface is implemented internally is not depicted. In some cases, when an external interface exposes internal interface and this is important to show, this is depicted in a 'daisy-chain'. For example, in Figure 16, 'Incoming safety interlocks' is linked to 'HT Control FW'
- Optionally, for keeping the diagram readable, when there is a clear distinction between input and output directions, of interfaces, inputs are described on the left side, and outputs on the right. This kind of layout is similar to the one of the functional decomposition diagrams.

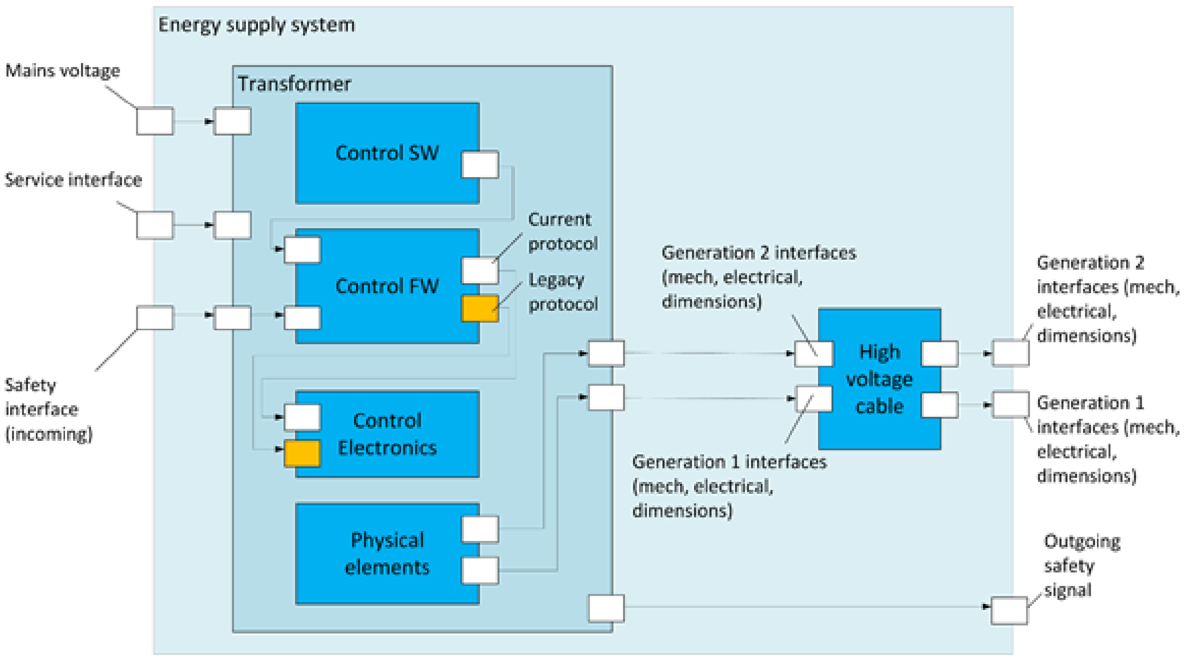

Energy supply system
Mains voltage
Service interface
Safety interface (incoming)
Transformer
Control SW
Control FW
Current protocol
Legacy protocol
Control Electronics
Physical elements
Generation 2 interfaces (mech, electrical, dimensions)
Generation 1 interfaces (mech, electrical, dimensions)
High voltage cable
Generation 2 interfaces (mech, electrical, dimensions)
Generation 1 interfaces (mech, electrical, dimensions)
Outgoing safety signal

Figure 16 - The *connections between blocks* view of Reference Architecture highlights reference and standard interfaces.

Side note: Comparison with SysML.

In SysML, a Port shows an interaction point. Ports can show required or provided services, as well as types of matter, energy, or data that can flow in and out [57]. Ports can represent any interaction point types, including Physical object on the HW level (an HDMI jack, a fuel nozzle); or a SW interaction (a TCP/IP socket, a data file, …). Figure 17 shows how the 'DellSta-77 Satellite' Block describes the port between an external object (a solar simulation chamber).

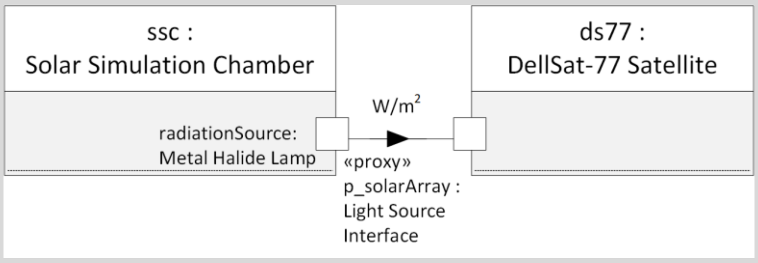


Figure 17 - Redrawn from [57]. A proxy port example that hands-over the power density from external source to an internal solar panel

Structures like shown in Figure 16 can be directly converted into SysML's Internal Block Diagram (IBD) that depict parts linked via interfaces. One example of such a diagram, redrawn from [57] is shown in Figure 18. Such a view is particularly useful when the architect wishes to highlight interconnections and interaction details of complex systems.

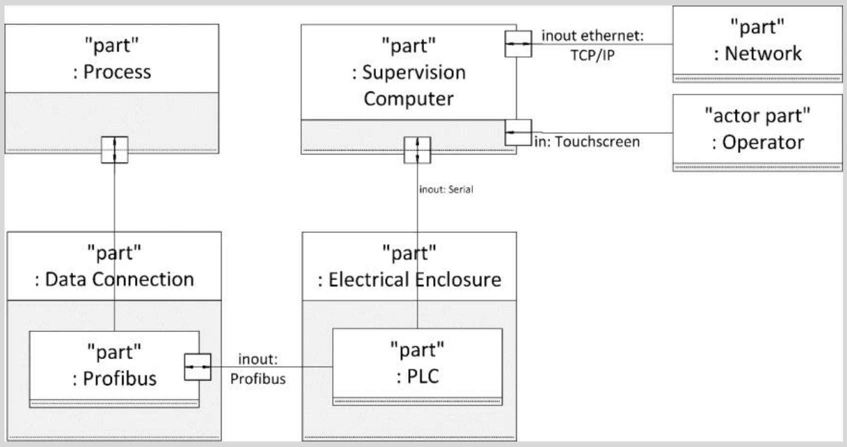


Figure 18 - Ports example. A proxy port example that hands-over the power density from external source to an internal solar panel. Redrawn from [57].

6.8 Layer 5: Realizations

In the System Decomposition layer, abstract building blocks have been described. The Realization layer describes concrete hardware (HW) and software (SW) components as they are used in a system to realize these building blocks. This layer shows which individual components can form groups to seamlessly work together. The layer is not part of a reference architecture, but it provides context for discussions on reference architectures and architecting. These individual components can be further configured during the production of the system.

Preferably, realization elements of a (sub)system can have their own lifecycle and roadmaps and are modular and easy to exchange. This requires having the same interfaces. An example of a reusable element is a HW (like a processor, battery) or a SW (library) component. Sometimes adapters are used, e.g., to adjust physical dimensions (re-package).

The diagram in Figure 20 describes different system *realizations* (system configurations) and how they are realized by combining *realizations* of subsystems. The figure has the following features:

- a vertical line from lowest blue boxes shows possible realizations (e.g., Supply A and Supply B for the Block ‘Power supply’);
- each horizontal line is a system configuration;
- if a system configuration needs no realization of a blue box, it is mentioned as ‘None’;
- if a system configuration can use different realization of a single blue Block, those options are shown next to each other (e.g., Camera 2 can use Sensor B or Sensor C);
- realizations have unique names; reusing the same name across configurations is enough to show that the same realization is reused (for visual readability, if a realization is reused across several configurations located next to each other, the realization block can be extended);
- realizations have an abstraction level suitable to talk about configurations, e.g., when the interfaces are re-used. Internals of the same realization on this level (e.g., realization version) can be different; a changed interface calls for defining a new realization box;
- realization boxes can be color-coded to indicate specific properties, e.g., to depict their development state: Obsolete (orange), In production (light green), or in development (yellow).

Even though the realization layer is, strictly speaking, not part of the reference architecture, it is a valuable source of information. It couples the reference architecture (and its system decomposition) to the reality of systems that have been manufactured in the past (the installed base) and about systems that are (or will be) manufactured. Apart from being a source of inspiration for creating the reference system decomposition, it is used to

- identify gaps and differences between the system decomposition and the reality of current systems; the discrepancies can be used to identify (i) errors or over-simplifications in the abstract system decomposition; (ii) elements that need to be put on the architectural roadmap to improve the design of systems that are currently being manufactured and shipped;
- identify variation points needed in the reference architecture, by creating an overview of the full system configuration landscape, providing an overview of system variants in the field and those rolling out of the factory;
- make informed decisions about the product variants that will be commercially supported (as new shipments and in service) in configuration management processes (e.g., triggering decisions to upgrade fields in the installed base to reduce the number of configurations to be supported, to remove commercial options if this would reduce the number of configurations to be supported etc.).

Side note: Comparison with SysML.
Elaborating an SD blue box to a Realization box represents SysML's generalization connection type. This can be read as a Realization Block M is a 'type of' SD Block A.
Generalization relationship conveys inheritance between two elements: a more generalized element, called supertype (e.g., SD level), and a more specialized element, the subtype. For instance, gyroscope is a type of sensor (see Figure 19).

In SysML, subtype inherits all defined features of its supertype: the structural features (properties) and the behavioral features (operations and receptions). For example, a supertype sensor can have values of mass and cost and can have operations like initialize, calibrate, and acquire data. Its subtype sensors (e.g., Magnetometer, gyroscope, and earth-horizon sensor) inherit these values and operations.

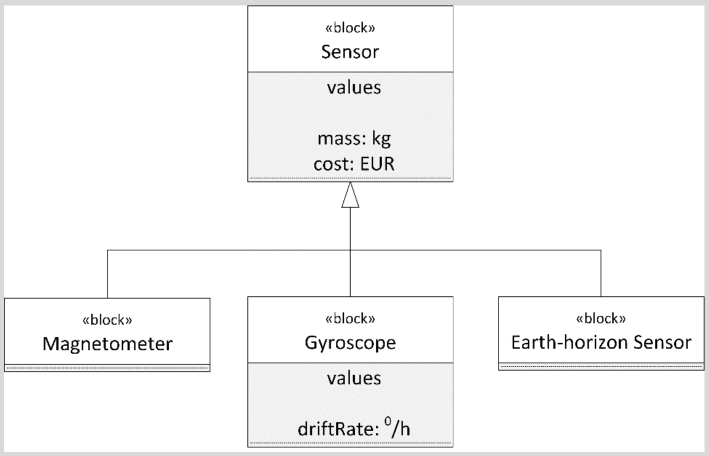


Figure 19 - Sensor described as an abstraction (reconstructed from [57]).

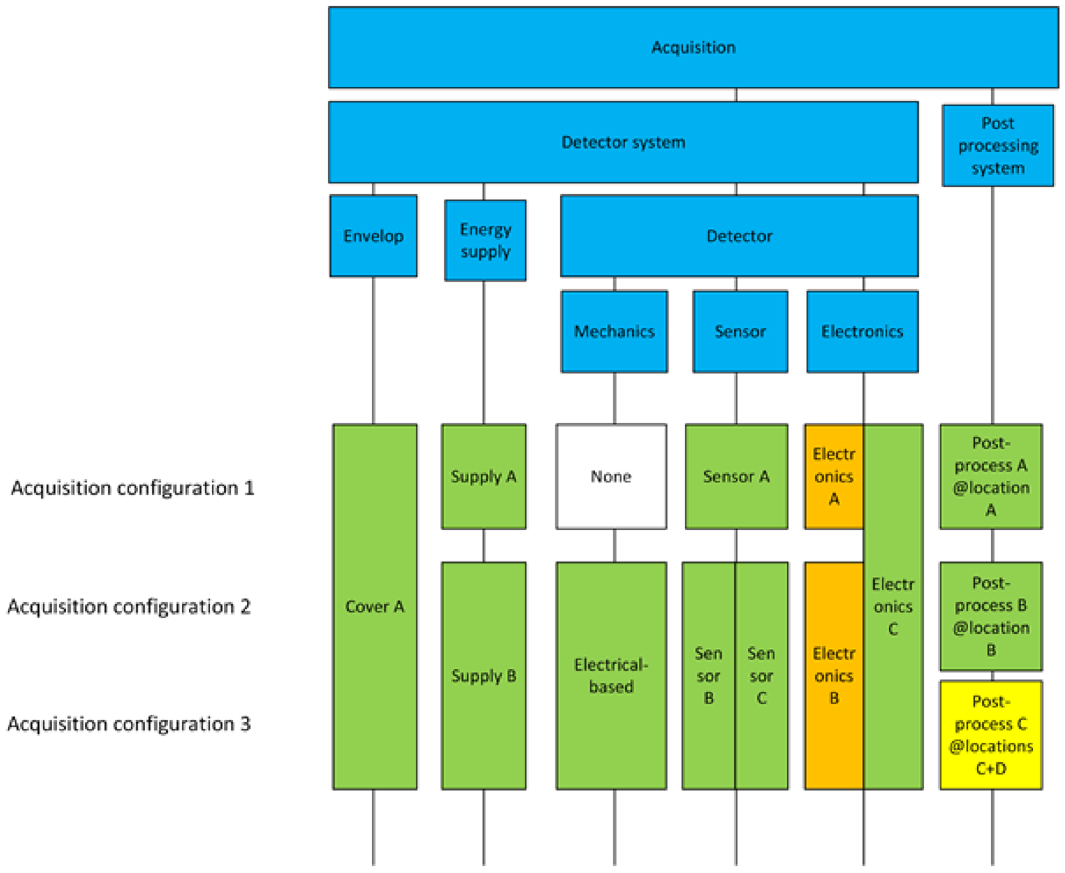


Figure 20 - Realization and SD layers (in blue): An example how an SD is realized and used in different configurations

6.9 Relating Design choices to Customer and Business Values

In section 6.6, non-functional aspects (qualities, -ilities) were discussed. These are modeled, and if possible, quantified, as properties of workflows that express a customer or business value. For instance, P_{wf} is a property of workflow *WF* (a non-functional). It reflects Customer or Business value *V*:

- P_{wf} is a function of properties of steps within the *WF*:
- if a step in *WF* is a compound step, a workflow itself, its property can be a function of its steps' properties;
- if system functions are used in a step, the value of P_{wf} can depend on properties of those functions;
- the properties for those system functions, in their turn, depend on design choices that are captured in the System Decomposition and on properties of the components that are used in system Realizations.

These relations between values, workflows, functions, system decomposition choices and properties of realized system components are depicted as the vertical arrows in Figure 4 on page 20. In methodology described in this whitepaper, formulas and models are used to calculate the value of non-functionals and of the corresponding values in terms of properties of customer/business workflows, system decomposition and realization properties.

Modelling and assessing the non-functionals of systems and products are typically difficult tasks, as they require accurate insight in the behavior of the systems. Especially early in development one has to create approximate models to calculate rough estimates. Later on, one can create more detailed and accurate models and combine them to assess the system's behavior (e.g., using a digital twin) and other properties (e.g., cost models, maintenance models)

As an example, we assessed an important customer value: the productivity, throughput, or the time-to-result. This example is explained in section 6.1 and worked out as a use case in section 8.3, using the structures of the reference architecture (as described in previous sections).

Assessment of non-functionals, requires not only the structural information, but also other types of system models, e.g., optical models, dynamical models, behavioral models. For instance, to calculate the image quality of an electron microscope one needs to model all processes that lead to the final image. The set of models would include the following knowledge areas: electron beam - specimen interaction, electron-optical magnification, electron detection using cameras, and image processing algorithms. These models should specifically include the disturbances that influence the ideal image. Typically, these disturbance models are developed by a handful of experts in each of the mentioned areas, and therefore usually focusing on a very small domain of the system. For an architect who needs an approximation of a non-functional for the entire system, this is obviously a difficult starting point. The DAARIUS methodology [58] for multi-expertise-team modelling and can be used for non-functional trade-off analysis.

6.10 Vocabularies

In industry it is of utmost importance to communicate efficiently and effectively. Miscommunication leads to loss of time and workarounds which are the costly consequences and therefore, needs to be prevented . Normally already a vocabulary is used in a company, but it is sometimes not made into a formal standard. The vocabulary used should be standardized to some extent and in some scope (e.g., company, site, department, expert group in two directions):

- the company language: English, French, German, etc. (especially in multi-site situations);
- a vocabulary or taxonomy defining the naming of system parts, company activities and processes, etc.

When creating (domain) models one automatically uses ad hoc names for concepts. When such models have to be shared, one must take care the concept names correspond to the standardized vocabulary, or that they are added to it when necessary. In case of the reference architecture, all used names must be part of the standard because of its company-wide use.

Note that also international standards exist for terminology in certain areas. As an example, for the domain of architecture, the ISO42010 standard [2] introduces strictly defined 'viewpoints', 'views', 'stakeholders', etc. It is important for a company to adhere to these standards, for the simple reason to not re-invent the wheel and to improve the communication in multi-site, multi-language situations.

7 Distilling a Reference Architecture: Approach and Techniques

7.1 The Challenge

Defining a reference architecture is a complex task, as the architect has to deal with a multitude of aspects, e.g.,

- Reference architectures need to be aligned with aspects of business, organization and processes [59, 60].
- The architecture design process requires revisiting, abstracting from, and finding relations between the customer and system domains that are described using different languages and goals [1].
- Reference architecture design faces all these challenges, amplified by the fact that the architect must critically assess a range of product and product family architectures. The reference architecture needs to contain the essential core of architectural decisions to assure that the business goals of the company will be met.

In the following paragraphs some of the difficulties of distilling a reference architecture will be revisited in a view of its relation to the business strategy and in a view of abstraction and generalization steps the architect has to make to design a reference architecture.

Finding out key values and business drivers.
Reference architectures establish standards and rules for a number of architectural decisions, but not for *all* of them. Only those designs, practices and guidelines that are critical in ensuring that the key business goals are achieved should be captured in a reference architecture. The business drivers are driving the scoping of the architecture.

Knowing when to stop modeling.
Related to scoping – a reference architecture is not a complete product architecture. So, the question is when to stop decomposing, defining interfaces, and describing the details. The architects typically want to be complete, but what is complete now may limit the development space for future specific solutions. They have to make this balance in covering future architecting with enough guidelines, and standards and allowing innovation to thrive. For a reference architect this is even more difficult than for a product architect as he needs to stop when far less is certain. Only the key architectural decisions should be captured in the reference architecture.

Dealing with an abundance of information.
Scoping and architecting guided by key values forces the architect to reason in a top-down manner. In practice, the architect also needs to take a bottom-up approach. Many companies have to deal with their legacy: systems have been designed, manufactured, installed, used and serviced in past decades.

Design decisions of the past put constraints on the design freedom of the architect. Some of those historical decisions are key to the success of the company and need to be respected. Many others are not that important and can be reconsidered. The architect has to deal with all these past design decisions and has to review the consequence of changing past decisions. Bottom-up reasoning is needed to generalize, from the existing solutions; the reference architect has to extract the *core* architectural choices made in the last years, or even decades. Only these should be captured in the reference architecture.

Here, as an example, one can think of a company designing and manufacturing printers in product lines for 30 years. The printer as a product has not changed its primary function - printing ink on a sheet, but the quality and performance have been improved following technological advancements. Typically, high-tech systems are driven by the time-to-market, and have a certain amount of technical debt accumulated, and some of the knowledge implicitly exists in experts' heads. The system architect has to manage this tremendous amount of information efficiently, to be able to reason about the system on reference architecture level in reasonable (cost efficient) timeframe.

The current architecture should not be taken as by default basis for the reference architecture.
Current architectures are a starting point for creating a reference architecture. The architect has to critically assess product architectures, and product family architectures, and check if they are still in line with the current business drivers and customer applications. If necessary, also the architect needs to correct and re-scope the existing architectures before generalizing them into a reference architecture that aims to reflect and support the company's mission in the long-term.

Incomplete system architecture documentation before designing a reference architecture.
Table 1 shows how reference, platform and product architectures differ in goals and contents. Platform architectures focus on multiple products, and reference architecture possibly on multiple platforms. Therefore, as part of distilling a reference architecture, it is essential to understand the existing platform and product architectures. If they are not described, the architect has to extract the essentials of the platform and product architectures as well.

Finding the optimal system decomposition.
Functional decompositions are not enough as a criterion for system decomposition. In deciding which system decomposition is better (if all deliver the required functionality), other criteria play a role. Among them are non-functional requirements for the customer and for business values (see section 7.6- Value-Driven System Decomposition). Since optimizing one value often goes at the expense of another value, architects have to make choices and perform trade-off analyses. They need to find the optimal decomposition in view of all customer and business values that were selected to drive the definition of the reference architecture.

Reference architecture in an organization.
To enable and encourage reference architecture adoption, the organization should establish reference architecture related processes. These processes enforce checking reference architecture compliance in the new product design. Also, in the process of introducing a reference architecture, there has to be a support and change management in place to ensure that reference architecture does not stay yet another document or a nice but not implemented idea.

7.2 The Approach

It is clear from the above-mentioned challenges that it is hard, or even unrealistic to describe the way of working in systems architecting/systems design as a simple series of discrete steps to be followed sequentially, leading to the desired result in an optimal way. In part, it is because systems engineering tasks can be decomposed for a dozen of roles: from the role of the requirements owner to the role of, for instance, a process engineer; roles that are very different [61] Next to that, according to Muller [62], the responsibilities of the architect have a less-tangible nature, which complicates the process.

In this section, the reference architecture distilling approach is described for the first phase of the life cycle of a reference architecture: the reference architecting, see Figure 1. *Reference architecting* is about capturing or extracting crucial architectural information for which three starting points are proposed, as shown in Figure 21. Customer values and business values are considered to be the driving forces for the reference architecture (for a range of products): they allow to work out the needs into functional and non-functional requirements and into (abstract) system descriptions. This is *top-down reasoning*. The third starting point is given by the existing portfolio of products. Generalizing the established knowledge about existing products, their development, and their use in the field forms another basis for creating functional and (abstract) system descriptions. This is called *bottom-up reasoning*.

The top-down and bottom-up reasoning, from the three different starting points (Figure 21), will follow three paths that deliver different sets of information at various abstraction levels. In section 6 the 6-layers were described for structuring the information when starting from the customer values. This structure can, for a large part, also be used when starting with from business values or from the realized products. The approaches from these three angles may result in conflicting information (e.g., three different system decomposition descriptions), which requires resolution. In principle, one should aim for convergence, which is essentially the validation and alignment activity shown in Figure 1. In our experience this confrontation happens at all levels but is really prominent at the system decomposition level: the three paths each provide a system decomposition based on the information obtained and reasoning so far (Figure 21).

To summarize the approach:

- Analyze *top-down*:
    - from the customer values to the <u>customer-perspective system decomposition;</u>
    - from the business perspective to the <u>business-ideal system decomposition;</u>
- Analyze *bottom-up*:
    - reconstruct using reverse engineering / reverse architecting from the existing systems to the <u>reconstructed (actual) system decomposition;</u>
- *Confront and converge* the 3 perspectives at system decomposition (SD) level to analyze the differences in structure and rationales (at various abstraction levels).

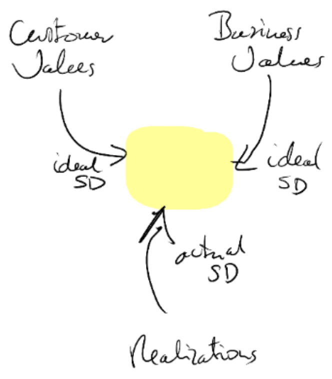


Figure 21 - Confrontation of Perspectives (Business Values, Customer Values, Realizations)

Some remarks about reference architecting in practice

As the figure above indicates, these activities occur in parallel! There is no ordering required, per se. Depending on the availability of experts, researchers, and the reference architects, the work is progressing; sometimes this is perceived as chaotic. This means that it is required to work iteratively and with jumps between the indicated paths, the levels in the 6-layer structure, and the levels of detail. Depending on the available persons and expertise, the team picks a topic, and - as we experienced when working with R&D personnel - starts from the realizations and existing products to immediately get good rapport and fast progress.

Depending on the availability, the quality and the amount of information, especially architectural documentation, one may have to recreate or reconstruct this documentation first before next steps can be taken. In practice, often detailed design documents have to be used to abstract and generalize the design concepts, mechanisms and architectural intentions. This is not an easy task.

When doing the confrontation, it is important to keep the story lines consistent. This is challenging because the acceptance of the content of the reference architectural views by the architects and system designers is not obtained automatically. However, the reference architecture is a core, a center for connecting people across disciplines, departments and sites. It is important to solve misunderstandings and conflicts when they appear, for which the concreteness and logic of the reference architecture will be of great help.

Note that gaps discovered in the confrontation can identify elements for the architectural or component roadmaps, and as such are input to R&D programming, product portfolio decision making, program management, etc.

There is a large variety in stakeholders and people that interact with the reference architecture. The right representation for different stakeholders can vary significantly, and choosing wisely is the responsibility of the reference architect:

- Senior management: connecting product properties to customer values and the impact on business performance;
- Architects: completeness of functional and system decompositions; clarity of interfaces and ownership; reasoning across many system views; coherence in product roadmaps;
- Component owners: mainly system decomposition and realization layers; dependency views;
- Operational and department management: clear assignment of ownerships to people and to units in the company’s organization.

In the remainder of this chapter, per layer, the techniques will be described to create the elements of the reference architecture.

7.3 Business and Customer Values

In the top-down approach to reference-architecting, two main groups of values are distinguished: *business* and *customer* values. The business decides on which customer and markets to focus, and which customer values to prioritize. These are represented in the Customer Values layer. The business also provides a set of requirements and constraints independent from the customer, which are addressed in the group of Business Values.

Customer values

Typically, the marketing department singles out typical markets, use cases, the most important goals and the way to how a product can solve the client's problems. The reference architect needs to understand what these values are. If the marketing department created value maps, then these are an important source of knowledge for the architect. Another useful source of knowledge are product architects and program managers who have worked with marketing to select what the next product will offer to satisfy the customer needs.

In reference architecting, using simple diagrams is preferred to relate a customer end-goal to system properties, as the one shown in Figure 5 on page 23. Most probably, value maps and other models that the marketing department created are not in this form, and the architect will have to translate them, or simplify them to design such a diagram.

When creating diagrams, to structure the values, one looks separately at the markets, applications, market segments and categories of uses and organizations. As already mentioned, the "5 why's" technique [48] can be used to structure them. Once the relationship between the values has been identified, and priorities have been assigned, the architect looks for the commonalities in a single market and creates a value model for the market. Finally, the architect looks across markets, and identifies differences and commonalities.

The number of the values in the model should be small enough for the stakeholders to be able to communicate and reason about them. At the other hand, it should be sufficiently complete to cover all relevant market aspects to identify their consequences for system design decisions.

Business values

Business Values (requirements and constraints) that drive the reference architecture reflect the long-term strategy, and decisions such as: where do we, as a business, want to be in 5-10 years from now and how do we want to go there? Structuring business values is part of the future work. For now, we name examples of business values:

- strategic values, such as the decision to focus on certain markets, market segments or application domains (for these, the customer values are collected and represented in the Customer Value layer);
- economic values and drivers, such as "reduction of the cost of good shipped", or "reuse to achieve efficiency" or "commonality among products to enhance the company image" or "commonality to reduce service and maintenance costs";
- organizational requirements and constraints, such as or competences, backgrounds and knowledge present in the company, or geographical dispersion of the company (factors the architect should take into consideration according to Conway's law);
- legal and regulatory constraints, such as for current and future policies on privacy and security, and regulatory standards applicable in target markets.

The reference architect needs to identify the main business values, requirements and constraints and their impact on the functional and system decomposition. In the design of the reference architecture, the architect has to make strategic choices: how to decompose the system (and its functionality) to assure that customer values will be realized while realizing the strategic values of the company too.

On the Gaudi-website, a collection of presentations is available from Gerrit Muller addressing “architecting for business value” [21] [63] . Also methods such as Design for Six-Sigma (e.g., [64]) provide tools and techniques to link specifications and design decisions to customer and business values. In the reference architecture distilling methodology described in this whitepaper, we have applied the principles from these to link reference architectures to customer and business values.

7.4 Workflows

As discussed in section 6.4, the workflow description is a model, consisting of the customer workflow model and the related system workflow models. The steps and questions for creating workflow models discussed in this section are not sequential; they are points of attention to keep in mind when designing this model.

What is the purpose of the model?

The goal of modeling workflows is to relate a selected subset of customer values to the properties of (abstract) systems and subsystems of the reference architecture. This goal can be refined into more specific modelling choices:

- which subset of customer values to choose?
- how accurate and precise should the values be?
  (usually, in a reference architecture one wants to compare global values without caring too much about their precision)

There is no single correct answer to these questions, the architect has to make trade-offs. When choosing the subset of values to represent in the model, the trade-off to be made is between completeness and model complexity and maintainability. On one end, there is the option to design one large, complex model that represents all the customer values that the reference architecture should address. On the other end, there is the option to design multiple simpler models, each focusing on one or two values. The first complex model may become unreadable and too complex for the stakeholders to understand it and maintain it. On the other hand, smaller models need to be updated and kept consistent.

What are the typical customer workflows that are related to the set of values?

It could be that detailed models of customer workflows already exist, but that these were made for a different purpose. For example, there may be models used for the purpose of simulation, of for optimizing timing performance of one specific workflow. It can be tempting to take over these models and reuse them without changing anything. But as their purpose is different, they are not necessarily the right representation of the reality for the property the architect wants to learn about by using the model. Or they are too detailed and contain information that is not needed, which makes the model unnecessarily large and complex.

In the customer workflow, only one or a small number of steps are performed by the system in scope. The steps that are not performed by the system need not be decomposed in detail; they should provide only the relevant parameters for the overall value quantification.

The questions to answer in this step are:

- what are the typical customer workflows we want to represent in the model?
- are they fixed or they may be changed, based on the result of modeling?
- are they the same for different market segments, can we generalize them for the value we are assessing?

It is important to note that the workflow models do not have to be complete for creating a reference architecture. It is an architectural and business choice which workflow information to capture and use as the main drivers for designing the reference architecture. Many details may be added later, when designing a product/platform architecture for a specific market segment and a specific application.

How does the system contribute to value creation in customer workflows?

The system workflow is an extension of the steps in the customer workflow. It captures, in more detail, which workflow steps are performed by the system, or by the system and another actor (system or user).

Before creating more detailed workflow models of the workflow steps in which the system plays a role, first check in which workflow steps the system's properties influence customer values the most. For these workflow steps decide

- which are the system workflow steps that contribute most to the customer value?
- which functions of the system contribute most to the value creation?

The system workflow steps and functions have to be identified that contribute most, and the workflows for these should be modelled in more detail. It can happen that the workflow models of the major contributors contains more detailed steps than this of other workflow steps. The customer values drive the level of details, and the granularity of steps in the system workflow model.

Decomposing the models

Regardless of whether workflows are modelled in one or in multiple models, and even though they are modelled for a selected, small sub-set of values, and only for typical workflows – they will grow and need to be decomposed into smaller models.

One thing to keep in mind is the purpose of the workflow model and what the architect wants to capture, as this will guide how to structure and decompose the model itself. For example, a workflow can capture control, data, resources and the decomposition will follow accordingly control, or data, or usage of resources decomposition.

One best-practice is to keep the model simple and limit the set of steps defined in one model. Here the architect has to answer the question: what are the decomposition criteria for the model itself? The workflow can, for example, be decomposed according to the systems as they are seen by the user, but they can also be decomposed according to, for instance, data flows. The architect doing the workflow modelling and decomposition needs to make choices that fit the applications and the way the stakeholders typically view and talk about the way or working in these.

One pattern in the sort of high-tech sequences that manipulate a workpiece, is to align it with the data and the level of details about the workpiece controlled in the sequence. For example, in a TEM microscope, the control and the sequence requires different data, different accuracy levels and speeds for the grid, grid square, cluster of foil holes, foil hole and then finally the actual region of interest for which the image is acquired. This is shown in Figure 22, where adjustment of the field of view is done in stages, with preparation of the optics and the alignments for every level of this adjustment.

Following the data decomposition, the sequence can be seen as modelling steps to prepare the optics and positioning to come to desired coordinates and then focus on the level of the data "below" in this hierarchy. The modelling pattern of workflow layers can thus be generalized as shown in Figure 23.

In the example, this modelling pattern was chosen to distinguish the steps for the stage and the optics system, and to identify the accuracy of the parameters needed to gather for the model.

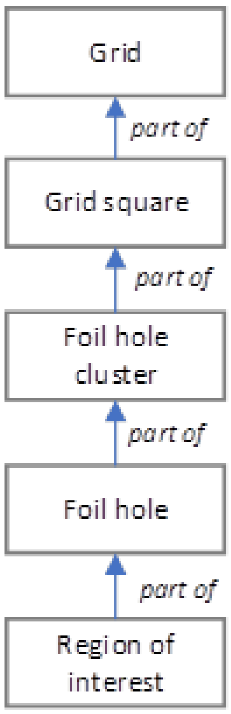


Figure 22 - Entity diagram representing the Grid consisting of Grid squares, with the foil with holes on it, mapped in clusters, and in one such hole there is a region of interest to acquire image of.

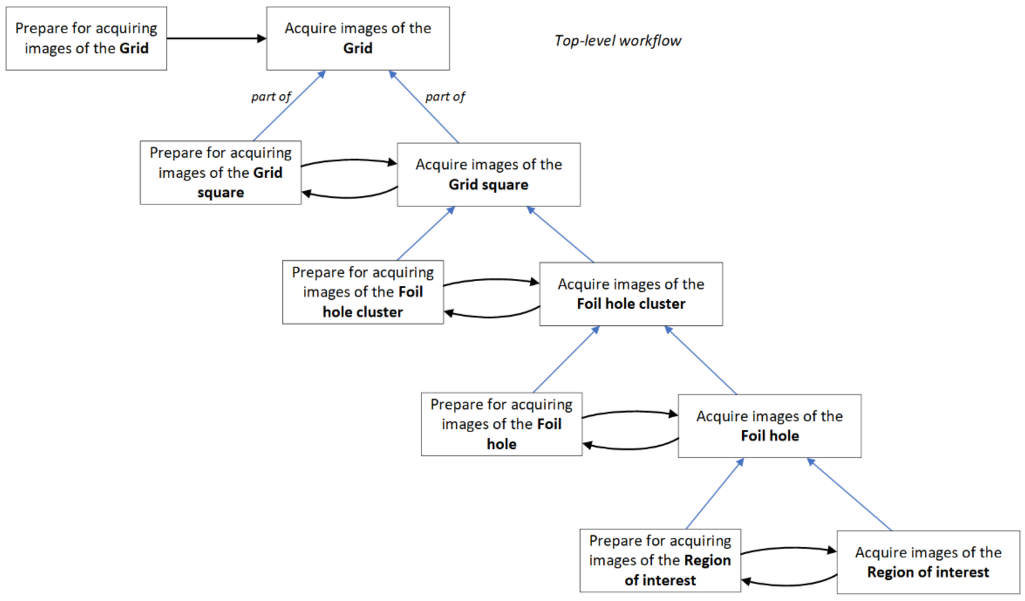

Prepare for acquiring images of the Grid
Acquire images of the Grid
Top-level workflow
part of
part of
Prepare for acquiring images of the Grid square
Acquire images of the Grid square
Prepare for acquiring images of the Foil hole cluster
Acquire images of the Foil hole cluster
Prepare for acquiring images of the Foil hole
Acquire images of the Foil hole
Prepare for acquiring images of the Region of interest
Acquire images of the Region of interest

Figure 23 - Decomposition of the workflow according to the levels of accuracy and speed of the controller, and optics and stage sub-system Functional Decomposition

7.5 Functional Decomposition

Modeling techniques to identify and decompose functions are well-known in the field of systems engineering [65, 66]. A *function* is, broadly speaking, “a bridge between human intention and physical behavior of artifacts” [67]. Finding a hard definition of *function* that is generally accepted is difficult [66]. The identification of functions requires abstraction and systematic conceptual thinking. The results of functional modeling are often presented as diagrams. The format of the diagrams is often coupled to the computer-based tools used for functional modeling, e.g., the format of SysML diagrams [56, 65].

Functional modeling can be done without computer-based tools (just paper and pen). Choosing a simple format will fulfill the purpose, and is easier to learn, explainable to non-technical stakeholders, and can be used to highlight only important differences. A good format is IDEF0 [51, 52, 53] In IDEF0, a pure functional model is created, without deployment, or mapping to implementations:

- A function is defined as a *transformation* (process, activity) of inputs to outputs. It is represented as a rectangular *function block* in a diagram, see Figure 24. A *verb*, or *verb phrase* is used to express the ‘human recognizable’ meaning of the transformation function itself. This is of course a crucial point: the verb phrase must be explained in more detail (not in the diagram) and may be part of the standard vocabulary.
- IDEF0 identifies the inputs and outputs of the transformation, and indication of the type (material, energy, signals, information) with *nouns*. They are depicted as arrows pointing towards (for inputs) and pointing out of (for outputs) the *function block* and labeled with a meaningful noun (again: these may be part of the standard vocabulary).

Note that there are three 'roles' of inputs:

- *operand* (pay load): the main input that has to be transformed, and typically the reason the function exists;
- *support*: needed inputs that help the execution of the function, such as power, clock signals, etc.;
- *influence/disturbance*: unintentional input that changes the execution of the function, usually in an adverse manner.

Arrows can have splits or joints, which indicate a separation into parts or a combination of parts in the inputs or outputs.

- *Control inputs* (data or signals) will influence the function in some way; they are depicted as similar arrows but pointing into the function block from the top. These are intentional and used for a purpose, as opposed to the disturbance-type of inputs.

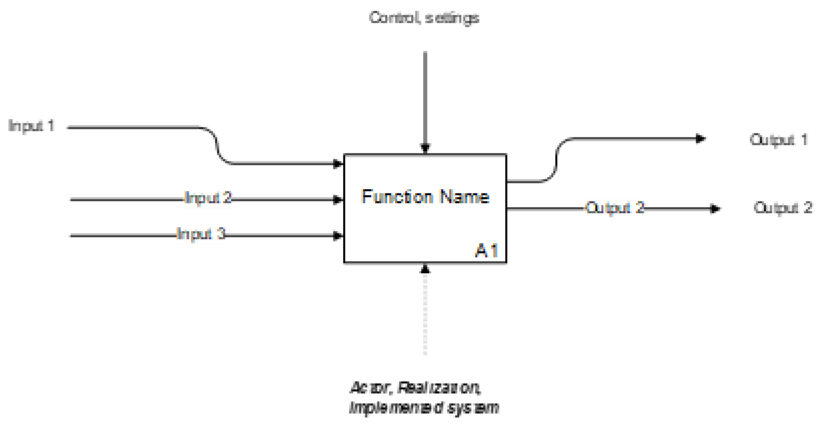


Figure 24 - Explanation of the concepts of IDEF0

- A different kind of element is the entity that performs the function: the *actor*, the *realization*, or the *implemented system*. This is, strictly speaking, not part of the functional reasoning, but it often helps in the functional modeling process. This mapping is indicated by a dashed arrow (showing that it does not belong to the functional world). It points into the function block from the bottom, see Figure 24.

Decomposition is splitting up the main function into sub-functions. This requires a creative step! To find the sub-functions one can get suggestions from identifying which type of actions take place, e.g., measure, data analysis, calibrate, etc. Another way is to look at an existing sequence of actions: e.g., first create, then configure, then transform, then copy, then destroy, etc. Note that the functional decomposition captures the transformation, but not the timing or order of actions, i.e., the workflow. It could be that in different scenario, the order is different, but that is captured in the dynamic behavior of the system (state machines or workflows). May be there are other kinds of 'logic' in the system that provide suggestions for sub-functions. Finally, an existing product can give suggestions for sub-functions by identifying the physical system parts, or their difference in location (see Figure 25).

This last method may cause confusion, as it may introduce concepts from the realization world into the functional world. The mentioned approaches to identify functions can lead to different outcomes. Therefore, the architect has to consolidate the different outcomes to find the right level of abstraction.

Decomposing functions further (deeper) generates a hierarchy, a tree of functional decompositions (from functions of the context of the system, to the system functions, to sub-systems functions, to sub-sub-system functions, and so on). In Figure 25 the function blocks are identified by a label (e.g., A1). At the deeper level an extra digit (e.g., A11) is added to make clear what the 'parent' function is.

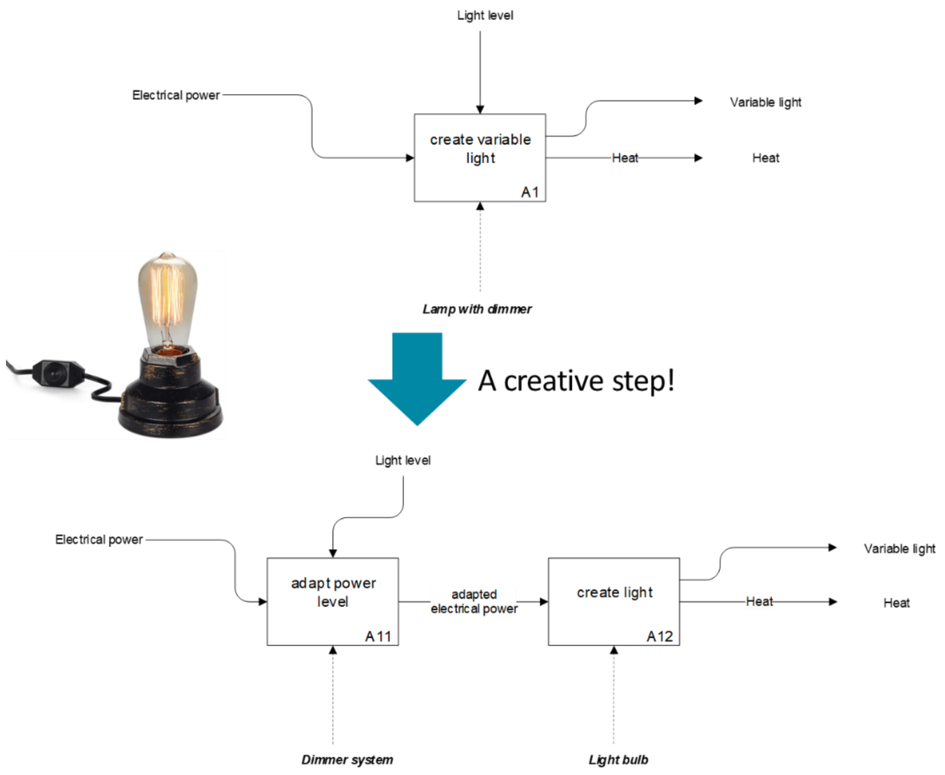


Figure 25 - Example of a decomposition step for the function of a lamp-with-dimmer system in the picture. At the top the main function is shown, which is internally split into two, see the bottom diagram. Note that an internal arrow is added. This decomposition also means that the lamp-with-dimmer system consists at the realization level of 1) a dimmer system and 2) a light bulb.

Of course, alternative decompositions can be made for Figure 25. For example, the sub-functions 'create light' and subsequently 'filter light' will deliver the same functionality. In this case the 'filter light' function should be controlled by the 'light level. The realizations are also different: 'Light bulb' and 'Filter system'.

An actual industrial example is shown in Figure 11 on page 30.

The described method is well-suited for learning functional modeling due to its strict focus on functions, and its simple notation. For full-scale industrial practice, computer-based tooling is necessary to cope with the scale and complexity of such systems. With a tool, often a modeling method and notation is coupled, which may differ significantly in understandability, simplicity and ease-of-use from the above shown method.

There are a few best practices to keep in mind:

- Keep the diagrams consistent, especially when shifting levels of detail (are the inputs and outputs at the deeper level at the boundary the same?)
- Is the diagram sufficiently complete (for the purpose)? It is easy to model too 'deep', or too 'shallow'. Note that one can focus on important functions only.

- Keep the right coverage in mind. Are all products in the portfolio covered?
- Find the hidden assumptions by what-if reasoning: is the product still functioning in a space rocket? Or in the desert?
- Check the naming of blocks and arrows for understandability and stick to nouns and verbs.
- Try to separate the concepts of the realization world from the functional world. This is a challenge, but it is sometimes unavoidable (e.g., ‘power’ is abstract, but sometimes it is functionally relevant to be more concrete: ‘electrical power’, ‘low voltage’, ’3V supply voltage’)
- Limit the number of functions in one diagram to 5-10. This can be done by combining functions, although it may lead to an extra level in the decomposition. Also limit the number of arrows by joining them.
- During functional modeling, all kinds of questions and remarks pop up (sometimes not relating to functionality, but to non-functionals or realizations). It is important to detect and capture these, as they will help in a next iteration to improve the model, or to obtain new insights.
- Capture assumptions in text and add them to the diagram.

There is always a choice of what to model. For a functional decomposition hierarchy as part of a reference architecture one can choose to model all system functions in the entire product portfolio, or only the most relevant for a certain audience. It depends on the purpose of the model: e.g., when one wants to understand the influence of the heat distributed in a product, one should include disturbance-type input arrows for those functions that are affected (and they in turn show different output). In case one only wants to explain the main idea, one can hide the irrelevant functions and inputs/outputs.

To conclude, functional decomposition is a crucial element of the reference architecture, as it connects the customer world to the technical implementations. In our experience functional modeling is in general a very powerful approach in product development. Unfortunately, it is often ignored in the high-tech industrial practice, so much can be improved in this respect. Note that a rich set of documents is available for further reading on the topic of functional modelling (e.g., [68, 69, 70, 71]).

7.6 Value-Driven System Decomposition

As discussed in section 6.7, System Decomposition is about decomposing the system into sub-systems, to define their interfaces and to allocate the functions and the flows between them to these sub-systems and interfaces. In a reference architecture, it is not the target to do a complete decomposition of the system. Instead, the reference architecture should only define those sub-systems and interfaces that are needed to assure that the architecture of all products based on the reference architecture will be well aligned with customer and business values.

Criteria for System Decomposition evaluation

There is no system decomposition that is "the best for everyone”. A decomposition can be designed by mapping one-to-one functional de composition, but that may be in discord with the way how teams are organized or geographically distributed. The system decomposition is therefore a result of a trade-off analysis, in which different aspects are considered and optimized for. The outcome of this analysis will depend on many factors in the context in which the architecture is being defined.

Inspired by the work of Penzenstadler [71], and following the BAPO structure, categories of factors to consider when designing the system decomposition can be identified. Within each group, there are different factors that are specific for an organization, for example:

- **B**usiness and customer value drivers:
    - strategic (e.g., “keep the core competence in the company”, “outsource what is not core”, or “put it in open source”);
    - economic (e.g., CoGS – Cost of Goods Shipped and costs of R&D);
    - legislations (e.g., GDPR in European Union, or MDR – Medical Device Regulations);
    - strategies for dealing with existing realization (legacy) and the system diversity in the installed base;
- **A**rchitecture and technical strategy:
    - functional and quality design aspects
        - alignment with the functional decomposition;
        - services and features offered to users;
        - quality criteria (all the, so called, “-ilities”), for example those defined in ISO 25010 [72] and others, e.g., functionality, reliability, usability, efficiency, maintainability, portability, safety, security, serviceability, etc.;
        - performance related aspects, like communication requirements, such as the need for the communication speed and bandwidths;
        - data requirements, like the amount and type of data to be stored, the type of data access foreseen in the product;
    - technical roadmaps and strategy:
        - emerging technologies and their impact on architectural decisions;
        - expected obsolescence of key technologies;
        - application of commercial of the shelve technologies or develop proprietary technologies;
        - technical strategies such as: SW architecture, centralized control vs distributed units/processors of control;
- **P**rocesses, alignment of the architecture with processes such as:
    - logistical processes, e.g., manufacturing, shipment and installation processes;
    - (continuous) integration processes;
    - concurrent engineering processes;
    - verification and validation processes;
- **O**rganizational, alignment with organizational structures and constraints, such as
    - the structure of the company in view of Conway’s law (e.g., departmental structures and responsibilities, geographical distribution of teams);
    - resources and competencies (e.g., educational background and experience).

In the following section examples of different trade-offs will be discussed, starting from functional decompositions and analyzing the factors mentioned above.

Different System Decomposition Scenario's

Figure 26 shows a simple example of creating a system decomposition:

- on the left-hand side, an arbitrary functional decomposition is shown:
  function F:(I_1, I_2, C_1) → O_1 is decomposed into three subfunctions F1, F2 and F3;
- as shown on the right-hand side, the architect has decided to follow this functional decomposition in the system decomposition by creating three subsystem S1, S2 and S3, where S1 implements F1, S2 implements F2 and S3 implements F3;
- the incoming outgoing flows of the functions are mapped one-on-one on the interfaces of the three subsystems.

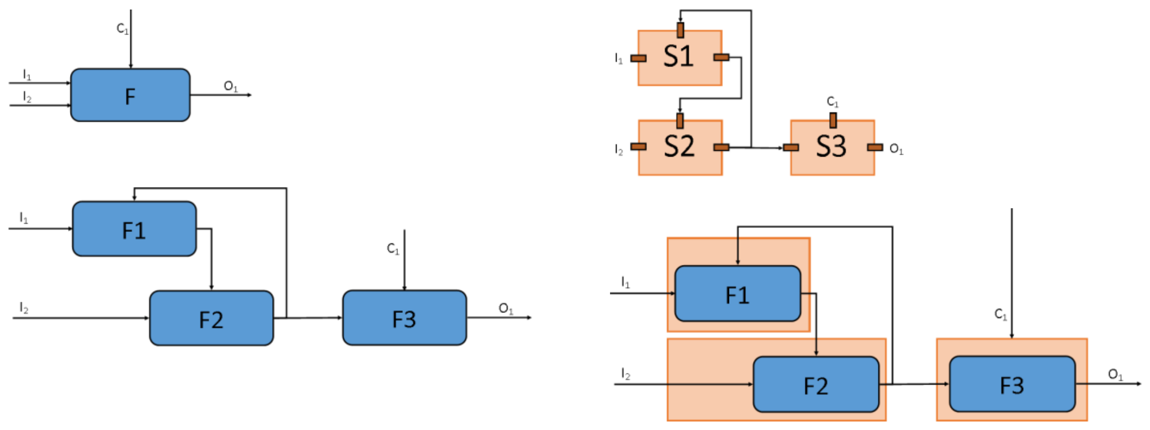


Figure 26 - One-to-one mapping of functions and systems

By making the mapping this way, the architect has created a "correct" decomposition of function F and system S according to the rules discussed in section 6.7:

- each element of the lowest-level function decomposition has been allocated to exactly one subsystem;
- each flow between functions has been allocated to exactly one connection between subsystem interfaces;
- all flows map on interfaces between subsystems with equivalent subsystem connections (let's assume this for the moment, as interface definitions have not been given);
- all interface connections are between compatible subsystem interfaces (let's assume this for the moment, as interface definitions have not been given).

For the architect, the main challenge is, however, not to find *a* decomposition (that is simple as shown in the example above), but to find the optimal decomposition according to a set of criteria (as discussed before). The decomposition shown in the example is not the only "correct" system decomposition for function F. In Figure 27, four alternative system decompositions SD1, SD2, SD3 and SD4 are shown.[9]

[9] This discussion of four decompositions is by no means meant to be complete; there are many more ways to create a "correct" decomposition, the example is there to demonstrate how a different sets of criteria might result in different decomposition).

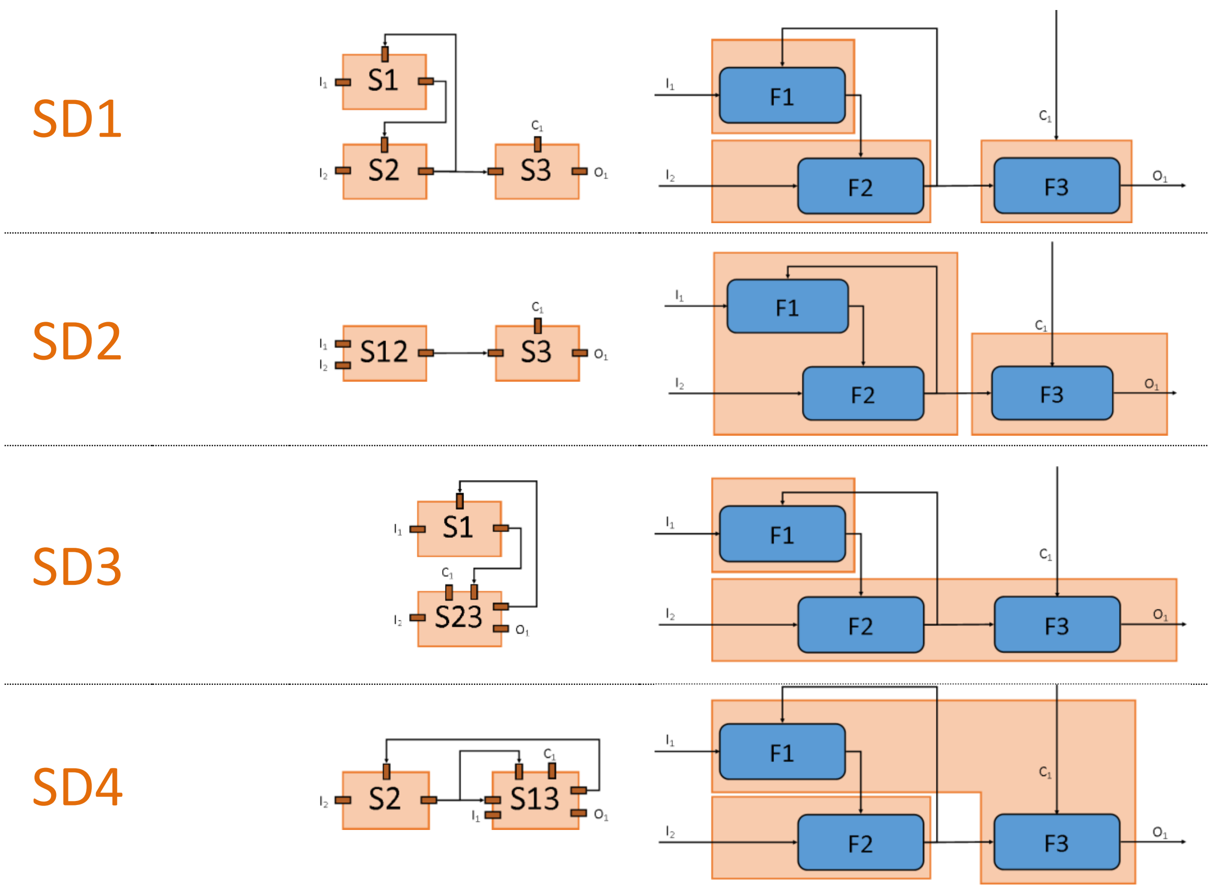


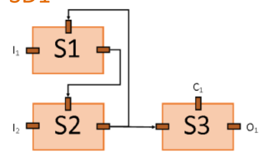
Figure 27 - Some correct system decompositions. SD1 follows the functional decompositions, whereas SD2-SD4 could be better according to some additional criteria

Although some of these decompositions may look complicated or non-intuitive (in particular SD3 and SD4), they are all "correct" and they all capture function F completely. What makes one decomposition better than the other?

- one type of criteria comes from good design practices such as cohesion, coupling, dependencies, etc.;
- another type of criteria comes from the BAPO-drivers discussed before: what are the important value/cost-drivers and what is the impact of choosing a system decomposition on these?

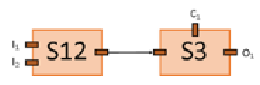
To take some examples:

SD1



Follows the functional decomposition without any further grouping. It is based on the assumptions that the functions F1, F2 and F3 are largely independent and that the interfaces between them are stable and can be standardized.

SD2



This decomposition follows the basic functional decomposition to a large extent and seems rather logical. The control-dependency between F1 and F2 is nicely captured in one subsystem.
This decomposition could be the preferred one if there is a tight coupling between F1 and F2 (e.g., if changes to the implementation of F1, imply related changes to the implementation of F2 in almost all cases). By putting these

functions into one subsystem S12, more freedom is given to the developers (of the functions F1 and F2) to optimize the interfaces between these two functions to optimize their overall performance.

There can be many other reasons to combine functions in one subsystem, e.g.,

- for BOM cost reasons if F1 and F2 share resources, or to avoid extra infrastructure (one PCB for F1 and F2 is probably cheaper than two separate PCBs + busses/cables to connect them);
- for serviceability, to reduce the total number of spare parts in the system (likewise for manufacturability);
- for stability reasons, if the realizations of F1 and F2 are tightly connected and require complex calibrations or alignment;

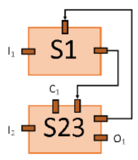
In addition, in SD2 the system nicely consists of 2 subsystems with just one interface that has to be standardized in the reference architecture. Many decisions can be delegated to subsystem development teams which gives a lot of freedom that could deliver fast innovations in S12.

There are also many good reasons for functions to be separated and to prefer SD1, e.g.,

- if the expertise for implementing F1 and F2 has been organized on two different locations (or in two departments);
- if (large parts of the) implementations for F1 can be acquired of the shelve where developing F2 is a core competence of the company
- if the development life cycles of F1 and F2 are different, e.g., new versions of F1 are available once per month, while new versions of F2 are released only once every 5 years
- for serviceability, if F1 is a “consumable” part that has to be replaced every month (note: this could be conflicting with the serviceability aspect mentioned before, to minimize the number of spare parts).

Many more reasons can be identified why SD1 or SD2 is the better system decomposition. Many of this will not be strictly technical. Most of them will depend on business drivers like the structure of the organization, the innovation cycle of system functions/parts, serviceability and manufacturability aspects, core IP versus of-the-shelve technology, etc.

SD3



For SD1 and SD2, it is not so difficult to see that they are two reasonable ways to decompose the system:

- SD1 follows the functional decomposition 1-on1;
- SD2 follows the functional decomposition in a logical way and combines two functions that have a (potentially strong) control dependency, and it delivers a simple system structure.

Both are logical, even without knowing the details of the business drivers for the choice between SD1 and SD2.

But this is not always the case: SD3 is a far less obvious choice. It does not follow the functional decomposition as SD1 does, nor does it deliver a nice-and-simple system structure as SD2 does. Still there could be reasons to prefer SD3, e.g.:

- if F1 is an optional function of the system where F2 and F3 constitute the core functionality;
- if implementations of F1 are being developed in Asia while the implementations of F2 and F3 are done in Europe;
- if the link between F2 and F3 is a time critical high-speed link, while the connections to F1 are not critical;

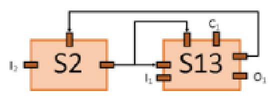
	• if cabling to implement the interface between F2 and F3 would be very expensive and complex, which could be avoided by putting these functions on a single PCB; • if F1 is a consumable, while F2 and F3 are not; • if the I_1 input interface requires dedicated and sensitive technology that requires special shielding.
SD4 	After discussing SD1, SD2 and SD3, it is probably obvious that there could also be good reasons to select SD4, which look complex and strange. When looking at it without any knowledge about the market- and business drivers, the only conclusion would be that it does not look good and that it is overly complex. That is a valuable and important judgement. But at the same time, the decomposition is “correct” since it covers all functions and lows correctly. Rejecting it because of its complexity and its lack of beauty is probably the right intuition, but what if: • cost pressure forces you to reduce the BOM and to combine subsystems and the implementation of F2 has to be done by an external company or by a team at a remote location; • S2 is a consumable subsystem, or a subsystem with a very high innovation rate; • S2 is an existing subsystem and the functions F1 and F3 are new functions to be released for a large installed base; • … It would still be fair to say that SD4 looks ugly and complex. But is that enough to discard it without any consideration?

Table 4 - Discussion of four “correct” system decompositions of function F

The main message from the discussion of these System Decompositions is that market- and business-value drives the selection of the best System Decomposition. The 6-layered approach introduced in this document therefore focusses on collecting information about customer/market-value, the workflows used in the market, the functions used in these workflows and on the company’s business drivers (see also section 4, regarding concerns driving the reference architectures) to collect the value-based criteria for creating and evaluating System Decompositions.

Making the right choice [area of future research]

From a wide range of possible system decompositions, the reference architect must make the right choice. At the same time the architect needs to address the question posed in the following paragraph: when to stop detailing the reference architecture?

This is part of the trade-off analysis tasks of the architect. For this task, several methods and processes are available, e.g., ATAM [73]. This process evaluates the impact of architectural choices on selected customer and business values that have been made explicit with use cases/scenarios.

In such a process, means are needed to relate architectural choices to values and to perform the required trade-off analysis in a structured way. This topic is in an ongoing field of research.

When to stop decomposing?

The evaluation of System Decompositions based on market- and business-driven criteria is applicable for all the types of architecting described in section 2: for reference architecting, for platform architecting and for product and system (or even subsystem/component) architecting. Having various layers of architectures, that represent various layers of abstraction, a key question is: where to start and where to stop? Especially for a reference architecture as discussed in this document, it is crucial to stop at the right level of detail.

For the value-based system decomposition, the criterion should be: is adding extra details to the specification needed to address a strategic value, or can further details be deferred to a lower-level architecture? If too many details are addressed in the reference architecture, this will cause three types of problems:

- choices may be suboptimal for specific system types, e.g., the cost of an entry-level system may become too high if technology choices defined in the reference architecture force the use of expensive components;
- the design freedom of system designers can be limited too much, thereby reducing their room for innovation;
- the effort needed to manage and maintain the reference architecture will become larger than needed.

Making deliberate value-based choices are therefore needed if:

- an extra layer of decomposition is added;
- extra design aspects (like interfaces) are being standardized.

The system decomposition does not have to be a complete design; it has to describe the strategic choices only.

Finding the right criteria for stopping/continuing further decomposition is a topic of research that will be addressed in the PaloAlto2-project (2021), in cooperation with Thermo Fisher Scientific.

7.7 Mapping Realizations to a System Decomposition

In this section, an approach will be outlined to reconstruct a System Decomposition-layer based on existing system realizations. The method iteratively creates two System Decomposition views (showing structure and interfaces) and relates them to functions and realizations. The method uses a *language* of System Decomposition and Realization layers, as described in the previous sections.

As mentioned in section 7.2, the bottom-up approach differs from the top-down one in its scope and outcomes. In particularly, it focuses on existing system components. Those may include legacy subsystems created when the company had a different organizational structure, used different technologies, or focused on different customer and business value drivers. For instance, a 230 Vac power interface could have been chosen from earlier requirements linked to mains electricity, whereas today a 3-phase 400 Vac interface would have been much more appropriate. In the bottom-up approach, one does not question why such an interface was created in the past or whether it should have been DC or 3 phase power. One captures such interfaces on the level of details that are important for future work of architects and designers. This approach focuses less on business values than the top-down method described above.

Identifying a structure of Blocks

To construct a system decomposition structure, one iteratively performs the following steps through discussions with experts and architects:

1. *Choose an abstract system and the functions it performs*.
   Abstract systems are shown at the bottom of the functions in the functional decomposition (e.g., see Figure 11), the functional flow graph of a 2D Imaging System for abstract systems "Camera system" and "Image Correction System").

2. *Identify components that perform the function in different realizations*.
   For instance, sensors of different brands can perform the same function "Detect electrons". In this step, specific system configurations should be included in the analysis.

3. *Formulate an SD Block that abstracts from the realizations identified in step 2*.
   The mentioned sensors can be grouped into an SD Block called "Sensor".

4. *Relate the abstract systems from Step 1 to SD Blocks from Step 3*.
   For example, the identified "Sensor" Block is a part of the "Camera System".

5. *Identify if different SD blocks can be grouped into larger SD Blocks*.
   For instance, "Sensor Electronics", "Sensor", and "Retraction" can be grouped together into a Block called "Camera", which is a part of Camera System. Consider discussions with module owners. The output of several iterations a tree-like structure of SD blocks (see subsection *Structural view: Structures of blocks* on page 34, in section 6.7). Such a system decomposition tree can have the chosen subsystem at the top level, sensors and actuators at the bottom, and intermediate groupings (e.g., assemblies) in the middle. The total amount of levels can be limited to maximum five or six to facilitate understanding.

6. *List realizations for each Block* [optional]
   (see subsection 6.8: *Layer 5: Realizations*).

The outlined steps heavily rely on creativity and group discussions. To facilitate this process, developers can adopt the following guidelines:

- Having discussions is essential to identify names and structures of System Decomposition Blocks. Therefore, involving multiple stakeholders (e.g., architects and designers of modules, components, etc.) is highly desirable;
- To stay recognizable, the names and structures in the System Decomposition should reflect company-specific terms and concepts.
- Block names should be unique, non-contradicting, and they should suggest that different realizations or conceptual solutions are possible (e.g., a "Cooling" sub-system that can be realized as Water Cooling or Passive Cooling).

- Formulating Blocks in such a way that they reflect individual Functions can be beneficial to support further modularity and future re-use of components in different systems.
- Tracing which functions the SD Block performs, helps to identify gaps in the decompositions.

Identifying connections between Blocks

To identify the interfaces in the System Decomposition (like shown in Figure 16, on page 36), the architect:

- takes the system decomposition (the tree-like structure of Blocks) as a starting point;
- redraws the structure by showing how larger Blocks incorporate inner blocks;
- identifies relevant interfaces by examining functions that the subsystem realizes;
- discusses with stakeholders if these interfaces shall indeed be captured and captures the rationale/reason why this Block or Interface exists (in text).;
- constructs a list of Ports on the boundary of the largest Block by:
    - drawing rectangles (▭) on the blocks' boundaries and by
    - adding a text note about their meaning;
- elaborates, if desired, how external connection can be traced within Blocks;
- Iterates, if needed.

Guidelines:

- For visual readability, incoming connections (towards block's themselves) can be grouped on the left side of the block; outgoing interfaces – on the right side.
- Many interfaces may have a direction of data/material flow that the designer can show as an arrow.
- Interfaces can be color-coded for readability:
    - external-facing interfaces can be shown in a specific color;
    - deprecated interfaces can be highlighted as orange (as in Figure 20);
- Interfaces can have a text note describing its properties. The note can be informal, if stakeholders agree to have a free-format description. Alternatively, the format can be standardized to a desired level of details, for instance:
    1. list of inputs (e.g., mechanical, electrical/thermal…);
    2. specifics of interactions, e.g., synchronous or asynchronous;
    3. content, e.g., protocols, signals, mechanical specs, and electrical ranges.
- Interfaces can be systematically identified by:
    - methodically moving from the outside to inner Blocks (e.g., blue boxes);
    - cross-checking links between all constituent Blocks (e.g., blue boxes);
    - looking at the realized interfaces (of the corresponding green boxes in the system decomposition).

Patterns

Developing and applying patterns for splitting System Decomposition elements helps to improve the consistency of the model and it supports communications between professionals. Such patterns and their usefulness for the Reference architecture level shall be discussed and agreed.

One pattern can be to separate mechanical and control components. A further split of controls is possible, e.g., to consider allocation of control functions in connection to a remote update strategy. For this, developers can choose to structurally separate hardware (mechanical or computational HW), firmware, and software, as follows:

- PCB as a 'bare board';
- non-programmable electronics: PCB + passive and non-programmable active components (e.g., transistor);
- programmable electronics: processors, FPGA added to non-programmable electronics;
- firmware (FW, can be distributed) has no business logic, but it
  - can be seen as one item because of timing constraints (e.g., relies on control loops with low-level timing);
  - implies autonomous behavior for keeping uptime, e.g., for safety needs;
  - provides a first level of abstraction by providing an (functional) interface, hiding hardware design aspects (e.g., it does not use values like voltages, but functional interfaces for set points and other settings);
- software (SW) – in general, containing business logic and other functionality (using the "abstract" interfaces provided by the firmware).

Another pattern can be used to highlight 'breaking changes' across generations, which is essential for re-usability of realizations. The figure below identifies Ports based on incompatible Generations (note that the Ports shows more than one interface).

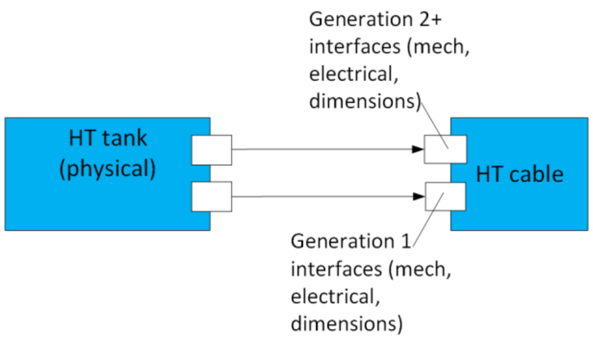

Generation 2+ interfaces (mech, electrical, dimensions)
HT tank (physical)
HT cable
Generation 1 interfaces (mech, electrical, dimensions)

Figure 28 - A pattern identifying breaking interface changes

Benefits of bottom-up construction of structures of Blocks

Comparing to top-down methods, the bottom-up method provides the following benefits**:**

- clear and effective communication with owners of (sub)systems, modules, and interfaces;
- building shared understanding of the realized system and the reasoning behind their architecture;
- clarifying current areas of responsibilities in the organization.

The iterative nature of the methodology and the continuous discussions involved in it give an opportunity to find the right scope for the System Decomposition (e.g., how many products and product lines should be covered).

The methodology also enables capturing variability and standardizing in-company configurations. As such, discussing modular items and interfaces in different configurations helps to identify variation points across products and product lines.

The identified existing system decomposition provides inputs to future-proof architecting. For instance, identified system variations can highlight the need for standardization. Identified rationale(s) for having those system variants can serve as business drivers and risks for assessing the current systems architecture (by using methods such as ATAM [73]). This supports the analysis of variations, specifically whether they originate from customer value drivers or business drivers. Once discrepancies have been identified, the next discussions can focus on how to handle undesired variations.

A scenario illustrating the bottom-up method

In this scenario, the bottom-up technique to construct SD views will be illustrated. For this a hypothetical company has been chosen that develops *astrophotography systems.* Such a night sky pictures using cameras with long exposures coupled to a telescope (e.g., see Figure 29).

The example illustrates topics relevant for complex high tech systems. First, the system realizations can vary significantly. A realization can be a professional product, e.g., 400 EUR computer-controlled telescope[10], but also a do-it-yourself project[11]. In a company dealing with a variety of systems architects strive to maximize reuse of components and interfaces across the systems. Second, such a system includes control loop typical for complex solutions. Specifically, to compensate for earth rotation during the photo exposure, the telescope continuously adjusts its rotation based on information from guiding camera.

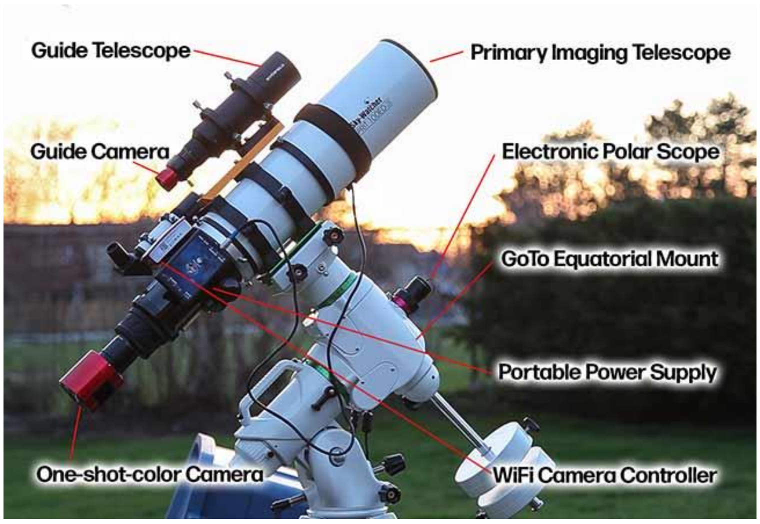


Figure 29 - A typical setup for deep sky astrophotography (figure from: https://astrobackyard.com/tracking-vs-guiding/)

[10] Nexstar 130SL computerized telescope, https://www.celestron.com/products/nexstar-130slt-computerized-telescope

[11] https://petapixel.com/2018/05/15/how-i-built-a-star-tracker-for-dslrs/, https://magpi.raspberrypi.org/articles/astrophotography-autoguider-project-showcase

The example starts at the moment that architects of a fictional company YourStars aim to construct a system decomposition that covers several product lines. As a preparation step, they identified their top-level function as ‘Create Night Sky Photo’ and functions that contribute to it (Table 5).

Function	Abstract system
Collect the guiding reference	Guiding Telescope + Camera
Gather and focus the light	Primary Imaging Telescope
Taking photos	Camera (e.g., DSLR)
Identify corrections	Guiding system[12] (SW + HW)
Implement corrections	Robotized Mounting

Table 5 - Functions and Abstract Systems in YourStars’ telescopes (example)

They architects visualized the functional decomposition of their top-level function ‘Create Night Sky Photo’ as shown in Figure 30. Afterwards, they focused on the function F2 ‘Follow sky target’ and created corresponding decomposition as shown in Figure 31.

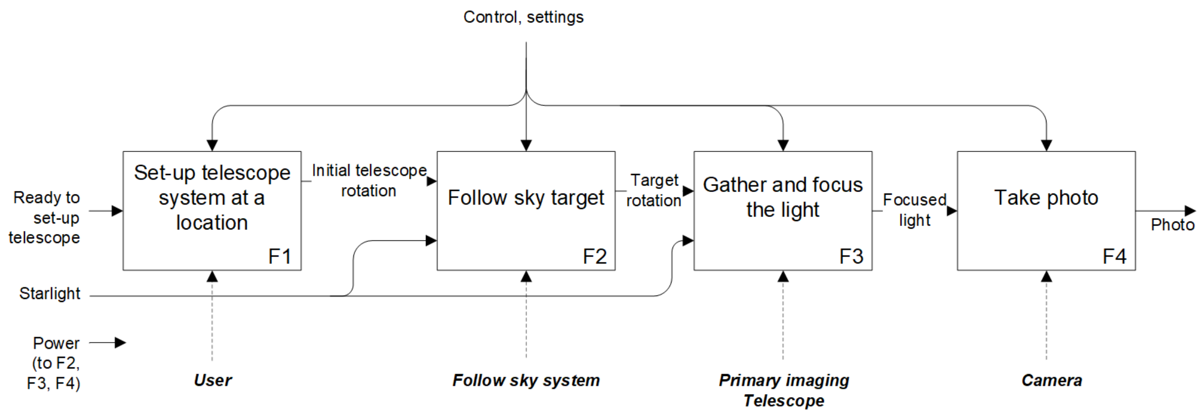


Figure 30 - Functional decomposition of ‘Create Night Sky Photo’ function

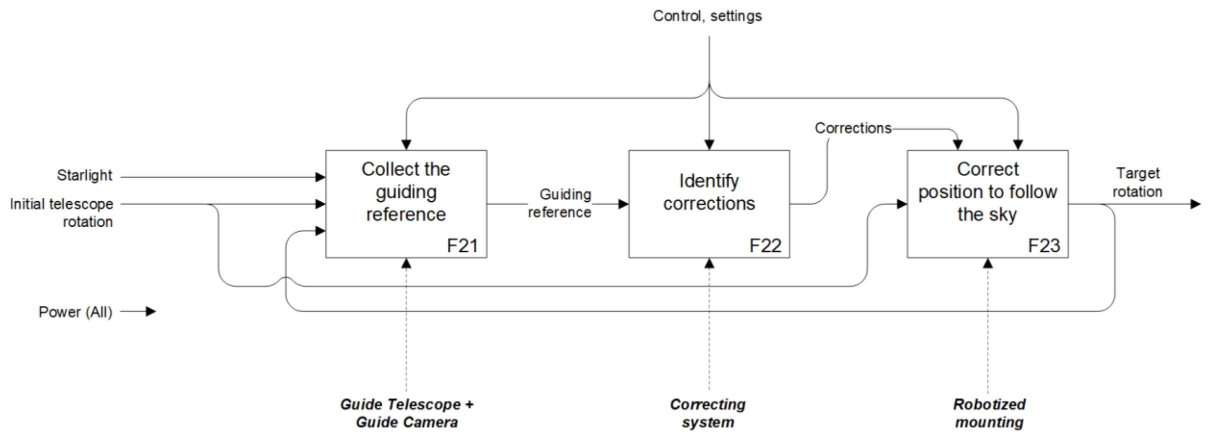


Figure 31 - Functional decomposition of ‘Follow sky target’ function (F2)

[12] To match the Earth rotation, small adjustments shall be made to the telescope’s orientation. Specialized software that follows a single star using a secondary camera and communicates small adjustments (pulses) to the mount (https://astrobackyard.com/tracking-vs-guiding/)

The architects followed the steps described in *Identifying a structure of Blocks* on page 59 as follows:

1. *Choose an abstract system and the functions it performs*.
   The architects agreed to focus on ensuring re-use of software and hardware components that perform the function 'Identify corrections'.

2. *Identify components that perform the function in different realizations*.
   The architects identified implementations performing the selected function across products by checking their legacy, current, and potential future realizations. They identified that their telescope-guiding software is running on iPads, Raspberry Pi's, and laptops in different hardware and software configurations. Configuration 1 is an iPad-based that needs a USB adapter. Configuration 2 is stand-alone Raspberry Pi running a specially compiled SW locally. Configuration 3 is a PC with a cloud-based SW for extended services.

3. *Formulate an SD Block that abstracts from the realizations identified in step 2*.
   The architects formulated SD blocks. For this, they chose to highlight the reuse of HW (hardware), FW (firmware), and SW (software). They also identified that a Water-Proof Case used for some systems . Also, the same Screen was used for a Pi and a connected PC configurations. Both Water-Proof Case and Screen became SD blocks.

4. *Relate the abstract systems from Step 1 to SD Blocks from Step 3*.
   The architects discussed how SD blocks form together the 'Correcting system'. During discussions, they found the same core software being reused. They decided not to split Screen into Screen-HW and Screen-FW parts, as this stand-alone component was supplied by a third party.

5. *Identify if different SD blocks can be grouped into larger SD Blocks*.
   Through discussions, the architects decided to group HW, FW, and SW into a new SD block 'Processing system'. This provided an opportunity to discuss interfaces to Processing system at once.

6. *List realizations for each Block* [optional]
   The architects related realizations to each SD block. Figure 32 shows the result of this optional step. The color-coding of Pi2 block indicates a reached agreement phase out its support. The yellow color of 'AWS1 SW' highlights that it is under development.

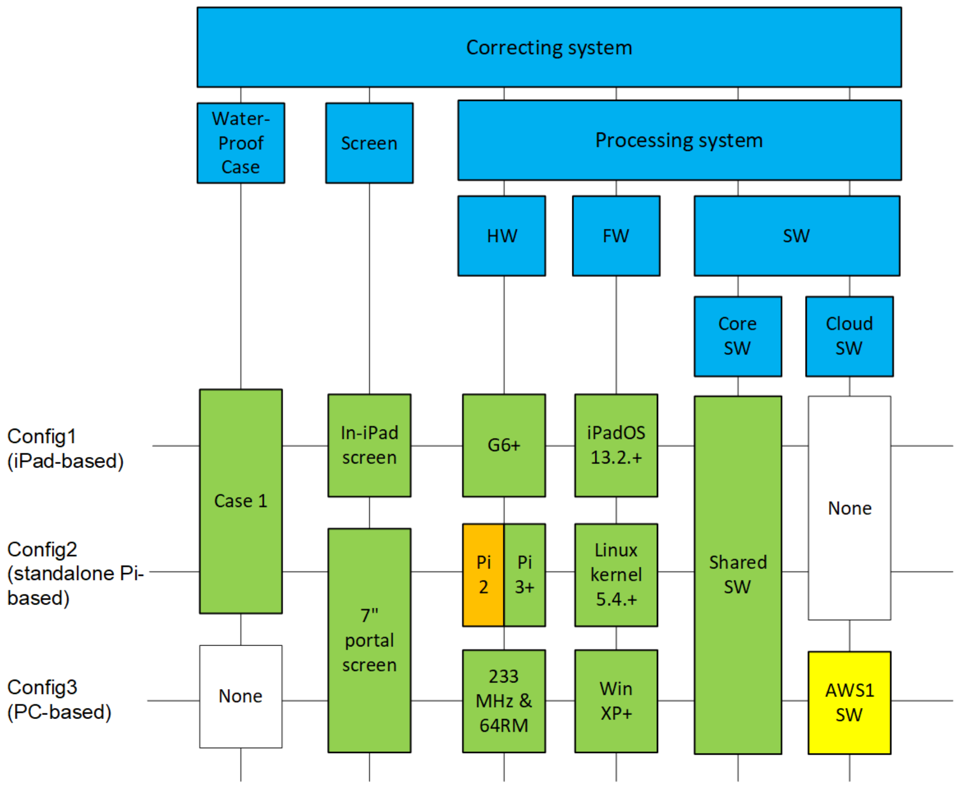


Figure 32 - System decomposition of 'Correcting system' ('F2. Follow sky target' function)

Afterwards, the architects focused on highlighting important connections between Blocks that should be preserved to support reuse of components. They created a structure of interconnections as shown in Figure 33. To note, while some Ports are more specified (e.g., 220V), other can be elaborated in separate documents, such as interface descriptions.

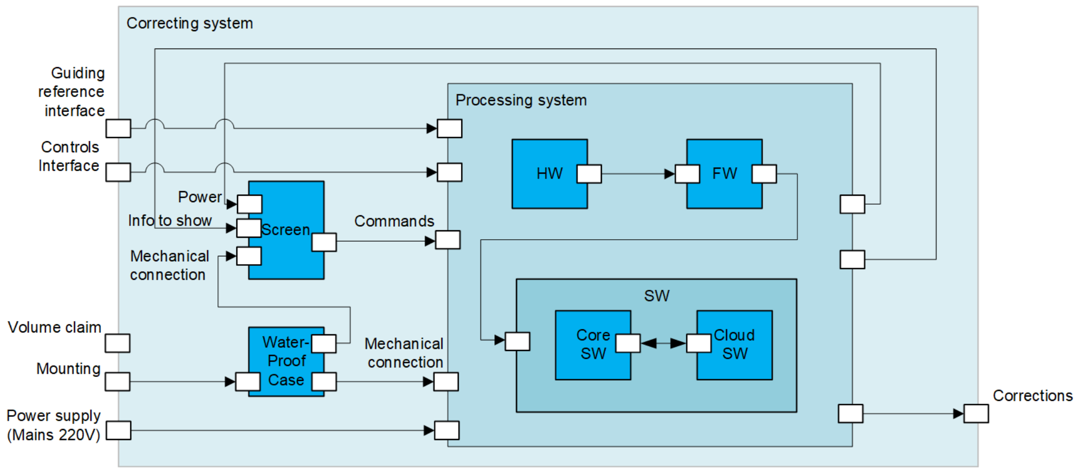


Figure 33 - Indicated port of 'Correcting system' system decomposition

In summary, the conducted steps led to a Reference Architecture description of a part of the system. Other functions can be described in the same manner.

The description obtained by using the method can be related to business and customer values. For example, it can inform discussions whether software reuse for all configurations is desired to address business needs and customer requests. By relating the obtained structures and company's business drivers it would be possible to judge suitability of the reference architecture to company needs.

8 Embedding and Using a Reference Architecture

Only when the reference architecture is embedded in the organization, one can actually say it is 'used'. This requires general acceptance[13] of the reference architecture as a source of guidance and constraints (by the target audience, e.g. platform and solution architects), as well as it being involved in the development processes on a practical level.

The embedding can be evaluated by observing the role the artifacts of the reference architecture, such as functional decomposition diagrams, play in architectural and design activities.

8.1 Processes

From a company-wide perspective, a reference architecture can contribute (improve steps in already existing processes or introduce new steps) to many of the processes running in the organization. To mention a few:

- *Defining product architectures*: the reference architecture shapes and restricts new product ideas in a managed way. The rationale of each restriction (in the form of a rule, guideline, or pattern) has to be clear to enable explicit discussions and decisions about how to make new product ideas fit in the reference architecture and with the underlying strategies.

  Furthermore, the reference architecture frames one's view of the system and gives handles (explicit system qualities, system functions, structures and behavior) to do the trade-off analysis for each new product (feature). In case the reference architecture contains quantitative models of system qualities, it enables making an initial quantitative analysis of the impact of system design choices.

  In the ideal case, in which the system has been "sufficiently completely" modeled, this will facilitate making a full mapping of customer requirements to system (component) configurations.

- *Improving system development efficiency with reuse*, e.g., with a "platform strategy": the analysis of the system decomposition and the variability in existing realizations gives insight in where to *standardize* on components (and processes), and so identify viable platform components. This enables optimizing reuse of both software and hardware.

  The reasons to standardize need to be captured explicitly. Deriving a well-founded platform architecture contributes to company strategies such as increasing of innovation speed and development efficiency.

- *Contribute to system design and system integration* processes through guidelines:
  - The guidelines on interfacing are meant to support the challenge of interface management. A standardized way of describing, and reasoning about interfaces will

[13] This is often not a done deal. In practice, personal and cultural factors influence the acceptance of new ways of working or new processes. It requires a typical change management process to enable a mindset change and general acceptance.

ease the discussions significantly. In particular, guidance in deploying new functions while respecting existing interfaces, is very important.

- A better understanding of the impact on customer values (or system-internal properties and qualities) provided by the reference architecture, helps by establishing a basis for *budgeting* (the distribution of the "pain", by setting requirements to components).

- There is also a coupling to quality and risk management analyses, such as FMEA, and its dedicated artifacts (e.g., boundary diagrams). The reference architecture provides the structure to systematically execute the FMEA analysis.

- Enabling a consistent role of software in a system architecture is often a difficult issue in system architecting and design, because the terminology, thinking and perspectives differ in the two disciplines. The clear definition and role of "software" (e.g., are we talking about concepts, high-level design, java code, embedded code, or data?) helps to decompose functionality, to partition the system, and to map the functions to the system partitions.

- *Securing knowledge sharing*: having an accessible and understandable reference architecture enables better communication, as it provides a shared basis: vocabulary, meanings, and ways of reasoning about *the* system, its use in the customer's context, its contribution to customer value and strategic choices to optimize business value.

- *Establishing a roadmap and portfolio strategy*: the reference architecture has to be well-aligned with the company's roadmap and it has to support the its long-term strategy. The drivers that shape the reference architecture have to be made explicit and clear: e.g., serving multiple *markets*, based on one technology platform; being prepared for creating 'specials' (non-standard products) to pursue adjacent markets opportunities; being future-proof for technology and business changes; using one common technology platform strategy; and enabling better support of deployed field systems including legacy configurations.

- *Configuration management processes* to manage the system variants (product configurations) the company will actively support (i.e., by manufacturing, shipping and installing, and by servicing these in the field). These choices can be reflected in the reference architecture by either (i) putting restrictions in the architecture of by (ii) adding variation points and diversity mechanisms to the architecture.

  In such processes, aspects to be considered are, for instance, (i) the total number of variants an organization wants to support in view of their added value and the cost incurred by each variant (in product development, verification/validation and release, manufacturing, service and support costs, the effort of verification, etc.); (ii) the impact of component or technology obsolescence on systems being serviced in the field and those being manufactured; (iii) compatibility of new functions with existing system variants, etc.

- *Defining and deploying the way of working* in the organization, on multiple sites. The diagrams of the reference architecture provide a handle to organize the way of working: e.g., the functional and system decomposition hierarchy can be used to suggest the distribution of responsibility (sometimes called 'ownership'), which will be a part of the governance in the company. A shared view on the reference architecture helps to create synergies across multiple sites and departments, as it gives a rationale for decisions that might impact different sites differently.
- *Supporting the establishment of component roadmaps*. As already mentioned, a reference architecture provides support for product design and for creating a portfolio strategy. This can be applied to the *creation* of roadmaps for system components too. The reference architecture also provides a structure to interrelate component roadmaps and to relate component roadmaps to product and platform roadmaps.

In general, for all the mentioned processes, it is essential to have good communication among people. Having a reference architecture contributes to this, by providing a shared vocabulary (e.g., naming of system parts, functions, customer values, processes and workflows, etc.) that gives clarity on terms and their meaning. This does not only help in collaborating within the company, but also in collaboration across companies (e.g., 3rd party suppliers, tool vendors, outsourcing).

Note that the use of a reference architecture can be addressed explicitly in existing formal processes (defined in the company's quality system), e.g.:

- in the company's product development process, tasks can be defined for a change control board (CCB) to review new proposals;
- tasks can be defined to show compliance of the new proposal to the reference architecture at specific milestones in the development process;
- the obligation can be defined to explicitly assign ownership of/responsibility for functions, components, and interfaces to a specific employee.

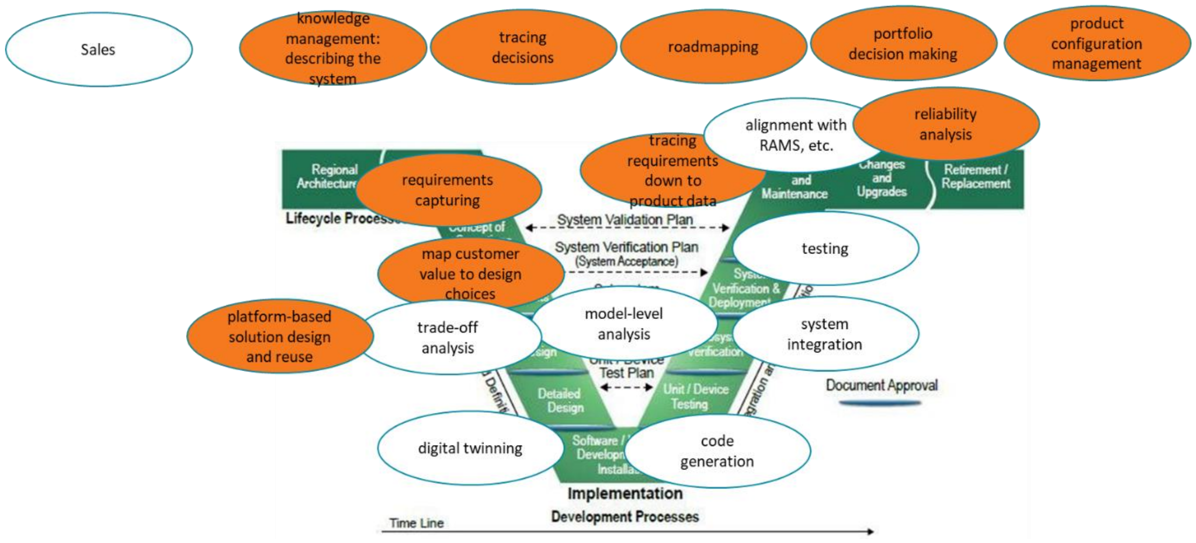


Figure 34 - A visualization of the product development process (expressed here using the V-model) showing in the ellipses typical activities. The orange ellipses indicate activities that are strongly supported by the reference architecture.

8.2 Roles

With so many processes, also many stakeholders are involved who can be influenced by the reference architecture, or who influence the reference architecture as it evolves. In Figure 34, a simplified overview is given of product development using the V-model, showing a number of development and related processes (in ellipses). The orange ellipses have a direct relation with the reference architecture. The number of orange ellipses clearly indicates the importance of a reference architecture in product development!

When looking at the development processes, the following users are relevant (see also, Table 1 on page 9 in section 1: Introduction]): subsystem architects, product architects, product family architects, portfolio architects. Next to these, product owners, team leads, lead component designers and engineers are also direct users of the reference architecture,

But there are many more potential users of the reference architecture, outside the development community, e.g.:

- employees from the Sales and Marketing departments, and other customer-facing specialists;
- strategists determining the long-term direction and goals;
- newcomers who need to learn about the systems during the onboarding process;
- managers organizing responsibilities and governance;
- employees from communication departments (e.g. PR department).

It is important to distinguish the activities of these users from the core users in product development, as they have different, but related goals. The interactions between the development of product architecture and product design, the development of family architecture(s), and the development of a platform (architecture) are complex and continuous.

Focusing on *product architects*, *designers and engineers*, the following activities are related to the reference architecture*:*

- they elaborate their (sub)systems based on the structures and interfaces defined in the reference architecture, and they are guided and constrained in their choices of types of interfaces, values exchanged via these interfaces, and other design aspects;
- they create product architectures by mixing and matching platform components and by reusing existing (non-platform) components;
- they show compliance to the reference architecture (thereby relying on the reference architecture to enable the product to meet customer and business targets), at defined milestones in the development processes defined in the company's quality manual.

8.3 Use Cases

After introducing a methodology for distilling and consolidating a reference architecture, discussing techniques for doing this and discussing the role of a reference architecture in the processes of a company, some examples will be given:

- use case 1 briefly describes how a reference architecture can be used to analyse the impact of "component improvements" on customer value;

- use case 2 briefly describes how the impact of the introduction of a new product feature can be analysed.

Both use cases are based on cases that were addressed in the cooperation of ESI (TNO) with Thermo Fisher Scientific in the PaloAlto-project.

Use Case 1: Analysing Component Improvements

A concrete example is the model developed to support the decision process of a system architect. In this case, the system architect used this model to check his gut feelings and to create arguments and underpinning facts for the discussion with other architects and the management on what type of stage to use.

The analysis uses the information available in the reference architecture descriptions and models (including the relations between the layers). It starts at the customer level, describing the concrete customer value. The role of the reference architecture is to define the customer value explicitly, and to attribute the customer value to the customer's workflow properties. The reference architecture also describes the mapping to the steps in the system workflow, the elements of the functional decomposition and those of the system decomposition. Finally, mappings to the realizations are needed to obtain values to calculate the relevant parameters. This enables relating a customer value to a product configuration (a realization).

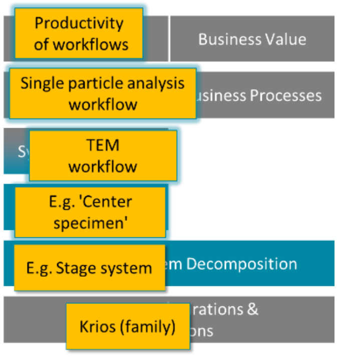


Figure 35 - The stack of models in the 6-layer structure, used for analyzing the productivity/time-to-result all the way down to the stage type choices at the realization level (for the Krios product two types of stages can be chosen).

In this example, the system components that to choose between are two stage variants with different timing properties. The timing properties of the microscope stages are relevant for the time-to-result value of the customer workflow, but their impact is not easy to determine directly. There is not a one-to-one relation between the speed of the stage and the time-to-result, which is chosen to be the best indicator for the customer value *productivity*. This means the architect has to determine the impact of a technical choice by taking the stage property values and propagate them up to the level of the workflow layers. There the workflow step durations and time-to-result can be calculated. A similar reasoning holds for the cost calculation of the workflow: what does the execution of one single particle analysis procedure cost? A set of models and their connections have been created in the DAARIUS tool [58] as shown in Figure 36. Switching between the type of stage in the models shows the impact on the time-to-result in this particular situation as shown in Figure 37.

Let's go through the layers in Figure 36:

- The customer value as just described will be part of the top layer, it is named time-to-result and in this case it is measured in hours.
- In order to understand what contributes to the time-to-result, one needs to know the measurement procedure in more detail. This is modeled as a workflow, as described in sections 6.4 and 7.4, and position the model in layer 2. This workflow modeling starts by capturing the tasks preceding and following the TEM measurement, e.g., the surrounding tasks, such as preparing a specimen before the measurement. Note that this analysis often leads to new insights and surprises, for instance that these activities take much more time than the actual measurement and that therefore improving the time-to-result may require improvements at different locations than expected!
- Next, the TEM workflow, the measurement procedure in the system, has to be modeled. What determines the time-to-result of this workflow? One starts by capturing which steps are in the workflow, positioned in layer 3. While doing so, one gets a feeling which steps are more complex than others, and contribute more to the total time. The more complex steps are decomposed into smaller steps, for which it is convenient to draw these steps in a new diagram, thus making a tree-hierarchy of workflow steps.
- It is important to make early estimates of the duration of each workflow step and to capture a rough value. Given these rough values, one is able to calculate the total time for a group of workflow steps, of repeated steps, and eventually for the entire hierarchy. It is possible to use simple formulas to calculate the aggregated time-to-result. To be more realistic, the models take the system uptime of the TEM, and

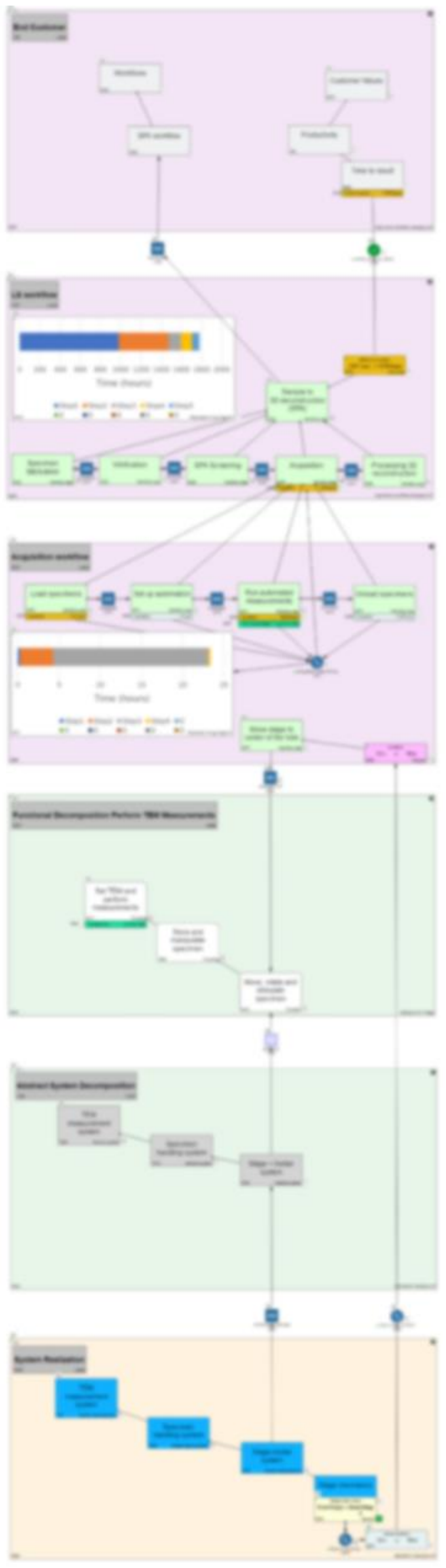


Figure 36 - The layered structure used to analyze the impact of a choice between two types of microscope stages on the time-to-result of a specific customer workflow. The reference architecture supports mapping from layer to layer, independent of the particular product.

preparation yield factors into account too. Here is a well-known pitfall: it is very important not to “over-model”, and not to go into details of steps, or different scenarios that do not affect the final value significantly. Note that the inputs to the calculations are fixed, rough estimates.

- Next, the connection to the technical implementation choices has to be made (to get rid of the fixed estimates and get specific values). The architect can use the structures of the reference architecture for that. A system workflow step is actually the performance of a system function. By using the mapping of a workflow step to a function in the functional decomposition (in layer 4), the architect knows which system function is involved.
- The architect also knows that, that function is executed by a particular system component. See the grey box (in layer 5), which is an abstract representation of the systems and that is also part of the reference architecture. This particular system component can be implemented in various forms: the real existing, concrete, tangible building blocks of the system (the blue boxes, see layer 6).
- Suppose one has 2 variants of a positioning stage: type1 and type2, with different speed and accuracy characteristics. To see the effect of having stage type 1 or 2, the speed values are connected to the calculation of the time-to-result values for each of these stage types: specifically to the associated workflow step duration.
- In the example (see Figure 37) , a time-to-result of 1,807 hours was calculated when selecting stage type 1. When the stage type is changed to type 2 one gets 1,817 hours. The value is not scaling linearly with the stage speed value! Other realization choices that are relevant to the time-to-result (e.g., camera types with different speeds, electron source types with different beam intensities) could be added to be able to compare and understand the scale of impact. In doing so, one could verify for all product configurations what the impact is on time-to-result and cost.

Figure 37 - The top two layers showing the impact of a choice between two types of microscope stages on the time-to-result. Note that at the bottom layer the choice between the stages is made, and its impact is calculated at the workflow layer.[14]

[14] Note: as Figure 36 and Figure 37 contain confidential information from TFS, they have been blurred intentionally.

This analysis can be extended by adding other customer values (e.g., product cost, running costs, image quality) and other relevant workflows, or business values (e.g., service costs, manufacturing effort) to do quantitative trade-off analysis. The DAARIUS tool can support both types of analysis.

This example shows the practice of so-called 'value-based architecting' in which the customer's values and the company's business values are leading the architecture development process. The reference architecture clearly supports this type of architecting.

Use Case 2: Analysing a new Product Feature

As a second example of the use of the reference architecture, the process developing a new microscope feature will be discussed. This product feature is usually developed by a product architect or a module architect, and has to comply to the reference architecture. (Exceptions do exist, of course, but they will at least lead to in-depth discussions about the justification of deviating from the reference.)

Below a simplified view is given of the specifics of the development process to clarify the essence. Therefore all iterations, validations, experiments, modeling activities, etc. have been omitted[15]. In essence, the architectural analysis requires the following steps related to the reference architecture:

1. Define the customer's workflow for the new feature; this is done by the product architect or the module architect as an element of the feature proposal (which is part of the company standard product/feature development process).
2. Define the associated system's workflow that is part of the new customer's workflow;
3. Analyze which functions are needed and check if they are already present in the reference architecture; this should be done together with the reference architect. Any changes can be proposed to be part of the reference architecture, and will have to be approved in a reference architecture change control board (RA-CCB) setting.
4. Follow the mapping to the system decomposition and check if new abstract components are required for the implementation of these new functions.
5. Check if existing interfaces suffice and define new ones if needed; this should be done together with the reference architect, in order to ensure completeness and consistency. The needed changes can be proposed to be part of the reference architecture, and will have to be approved in a reference architecture change control board (RA-CCB) setting.
6. Identify whether implementations and existing (platform) components are suitable for reuse in systems that offer the new functions.
7. Determine impact of the new feature on all system properties and qualities.

[15] The next update of this whitepaper will contain more about the usage of and processes connected to the reference architecture.

8. Verify if the complete solution complies to the reference architecture; this should be done together with the reference architect, as a final check before the implementation development phase, and as a way to obtain valuable proposals for the reference architecture.
9. The proposed changes to the reference architecture must be approved by the reference architecture change control board (RA-CCB).
10. Determine ownership of new functions, subsystems and realized components for the new feature; this is a governance topic and should be addressed by the appropriate management board.
11. Define further detailed design and implementation activities as defined in the standard product development process.

9 Observations, Conclusion and Future Work

9.1 Industrial Deployment of the Methodology

This document reflects the status in February 2021, at the end of the PaloAlto-project. In this project, ESI (TNO) and Thermo Fisher Scientific developed the methodology and applied it to distil the reference architecture of their Transmission Electron Microscopy (TEM) systems. During the project, the focus has primarily been on distilling the reference architecture of their *current* high-end TEM portfolio, with some connections to their mid-range portfolio. Continued development, deployment and validation of the methodology is needed to address the commonalities and differences for the *full* TEM portfolio (including entry-level and mid-range systems) and to address *future* customer and business values and their impact on the reference architecture for a next generation of systems.

During the PaloAlto-project, close cooperation of ESI (TNO), developing the methodology, and Thermo Fisher Scientific applying it for their systems has proven a successful way of validating and embedding the reference architecting methodology in a high-tech industry. To support the embedding, ESI (TNO) has cooperated in use cases to apply the method and ESI (TNO) has developed methods and materials for training architects to develop the diagrams and models that constitute a reference architecture. In particular, training methods and materials have been developed for (i) workflow modelling in the Customer and System Workflow layers; (ii) functional decomposition for the Functional Decomposition Layer; (iii) mapping realizations to system decomposition for the Realization and System Decomposition layers. These training methods and materials have been validated by training groups of systems architects of Thermo Fisher Scientific. Although these materials were based on confidential use cases from Thermo Fisher Scientific, a set of materials is available at ESI (TNO) from which generic training materials can be made to train systems architects of other companies.

An important lesson during the deployment process at Thermo Fisher Scientific is the vital importance of "ownership". During the process, there has been constant management focus to assure that ownerships for all models and materials were assigned. This applied for the artefacts at all layers that were worked out in detail: the Customer and System workflow layers (including models to calculate key performance indicators for customer values such as *productivity*); the Functional Decompositions layer; the System Decomposition layer and the Realization layer for the high-end TEM systems, resulting in the delivery of a complete set of diagrams and models. During the deployment, an essential aspect was to assure *systems thinking* of all owners of elements of the reference architecture: the ownership for the reference architecture was distributed and delegated to a group of architects. Together they have to assure its quality and consistency, this requires (i) an adequate level of system overview and systems thinking for everyone involved; (ii) the assignment of the role of "the reference architect", taking the ownership of the full package.

Furthermore, several processes of R&D at Thermo Fisher Scientific have been updated to assure that the reference architecture will be maintained and updated and to assure that the reference architecture will deliver its expected added value. Examples of such process changes are: (i) a Change Control Board has been established for the TEM reference architecture; (ii) regular reporting about the status of the reference architecture (improvement) and deployment has been added to the regular senior management reviews of the TEM R&D department; (iii) the process to create customer-specific system configurations (at Thermo Fisher Scientific better known as Non-Standard Requests) has been extended with obligations to analyse whether the specific configurations is compliant with the TEM reference architecture and to report on any non-compliance in the related decision processes.

Although a customer value analysis was performed at the start of the project (for the Customer Value layer), this analysis was done with a limited scope and depth. A set of customer values has been identified and for a number of these, models have been created to link system decomposition and realization choices to their impact on these customer values (as sketched in *Use Case 1: Analysing Component Improvements* on page 71). These models have proven to be very powerful in facilitating discussions between architects and between the architects and their stakeholders. After training the architects of Thermo Fisher Scientific, they were quickly able to continue the creation of such (workflow based) models to analyze and communicate (with management) the impact of system and workflow changes on customer values such as productivity.

During the project all diagrams and models have been created with tools such as Microsoft Visio and the DAARIUS-tool of ESI (TNO) [58]. This worked well for the PaloAlto-project, in which a relatively small group of architects used and created the models. As the group grew, the need for a software tool to support the process emerged. This need is based on the sheer scale and complexity of such a set of interrelated diagrams and models: managing the interrelations and their complexity, version and change control, cooperative working, etc. In a next step of scaling up the deployment and embedding of the methodology, a next step in tooling is expected to be taken by the adoption of a tool for architectural modelling or for model-based systems engineering.

9.2 Observations, Evaluation and Conclusions

Based on the applied research and industrial embedding done in the PaloAlto-project, the following conclusions were formulated:

- The 6-layer methodology for distilling a reference architecture has proven to be both feasible and valuable in the high-tech industrial context of Thermo Fisher Scientific. Because of its generic nature, it is expected to be feasible and valuable in similar high-tech industrial contexts too. Application at other industrial companies is needed to validate this expectation. First discussions and presentations with other industrial partners in the network of ESI (TNO) has confirmed a broad interest.[16]

- During the project, we noticed that finding customer values and mapping these on workflows works well; these values are mostly strongly related to workflows and to system functions (and properties of these functions like performance). Systems architects are used to working from requirements and functions to logical and physical allocations in ways that are similar to the approach we have chosen (for instance the RFLP: **R**equirements, **F**unctional, **L**ogical, **P**hysical-approach common to today's MBSE tools and methods).

  Starting from business values and using these to evaluate alternative system decompositions, as discussion in sections 6.7 and 7.6, is a process that is less natural to most architects. The

[16] In the Canvas-project of ESI (TNO) and Canon Production Printing, the methodology will be applied and validated.

initial attempts to do this had only limited success. We concluded that the aspect of business value-driven system decomposition requires a special focus in future applied research.

- We observed that, once the process of creating diagrams and models got started and accepted, architects have the drive to capture as much system knowledge and design decisions as possible. This has resulted in a set of diagrams and models that is rather detailed. Making a clear separation between the product/system architecture and the overarching reference architecture was a challenge during the project. To make the separation, the criterion used was that the reference architecture should only capture those stable structures, interfaces and other decisions that are of strategic relevance for the full product range[17]. Although this criterion gives guidance, it requires good judgement to decide which (useful, relevant and valuable) design and architecture knowledge not to capture in the reference architecture.

- In the PaloAlto-project, the reference architecture has been distilled for an *existing* product line. As many design choices and their rationales were already known for the existing system realizations, there was a strong drive to capture "as much as possible". Consequently, the bottom-up process of mapping realizations to the system design was perceived as very valuable. When defining the reference architecture for a new product line or for a next generation of products, this is a less likely pitfall since many detailed design decisions are still to be taken in such cases.

  When starting from an existing product line, shaping a reference architecture avoiding this pitfall is not easy, as many the added value of capturing and structuring architectural knowledge is of major value for the company, as it fills lacunae in the existing architectural frameworks for the daily work of the architects and of their management. We concluded that, in such cases, having a reference architecture and its rationales defined for the *existing* products is a prerequisite for defining a reference architecture for *future* system (in view of *future* customer, market and business values). The captured reference architecture for current systems forms an essential steppingstone.

- To boot-strap the process, no "power" MBSE tools are needed. A good whiteboard, a set of markers and a drawing tool (such as Microsoft Visio) are enough to start the process and to create the initial sets of drawings and models.

  The DAARIUS tool has proven very powerful to support this process by its capabilities to create quantitative models in an intuitive and explorative way (without enforcing a very strict and formal modelling formalism). These capabilities enabled researchers from ESI (TNO) and architects from Thermo Fisher Scientific to create the quantitative models that relate

[17] Once architecture/design decisions inside individual system variants start to become dominant in the architectural diagrams, this is a warning signal that you might have gone too deep.

realization and design choices to customer values. The capability of the tool to support multiple architects to work on the models in parallel also proved valuable.

After creating an extensive set of diagrams and models, the need for a software tool to support the process was evident. State-of-the-Art architecting tools and MBSE-tools provide a range of functions that are needed for scaling-up the deployment. A first observation is that the focus of these tools is not on some of the vital elements of the method: (i) customer and business value modelling; (ii) quantitative analysis of the impact of system decomposition and realization choices on customer and business values. For the creation of such models, the DAARIUS tool has [58] some unique capabilities that are (at first sight) absent in most MBSE-tools.

- Finally, we concluded that deployment of a reference architecting method requires focus from senior R&D management. Management needs to ask for tangible results and embed the reference architecture approach in their R&D processes. Assigning clear ownership and driving for the commitment of the architects in combination with adequate support in the form of training, tools and time to work on these new reference architecting tasks are a crucial for successful and sustainable deployment.

9.3 Future Work

The methodology described in this document is already usable in its current form. Yet, it can clearly benefit from future work.

First, it's highly desirable to ***validate*** the methodology in addition contexts to identify (i) limitations and (ii) opportunities for generalizing. Without doubts, applying it at another company and on another type of system will provide new insights.

Second, the method can be further extended to account for:

- New views and non-functional aspects. Consider, for example, questions like:
    - which fundamental principles are important to capture in reference architectures?
    - how can the reference architecture structure be related to traceable structures for individual product design/instance?
    - how to best capture tensions between product qualities?
    - how the reference architecture can assist with working out system requirements?

- Deep integration of business values into the reference architecture and into architectural reasoning. It will help to address questions like
    - what would be the impact of organizational structures on the architecture (and vice versa)?
    - how to link business mission, vision, strategy to the reference architecture in a traceable way?
    - how to ensure systematic connection of in-company processes with the desired way of using a reference architecture in a company?
    - how to link reference architectures to platforms?

- how can a company create a structure of relevant budgets in its reference architecture to ensure that business-critical aspects are adequately addressed?

- Maintenance of a reference architecture during its active use and updating it to reflect changing market conditions, new customer values and evolving business strategies. Relevant questions include:
    - how to cope with limitations in innovation due to constraints in the reference architecture?
    - which options (e.g., to allow for a later choice between two technologies technology) shall be incorporated into the reference architecture and in product architectures?”

- Scaling up of the application of the methodology to larger groups of architects with powerful tool support to manage the consistency and complexity of the set of diagrams and models that constitute the reference architecture. At the end of the PaloAlto-project, most models were created with tools like Microsoft Visio, and some models were created in the DAARIUS tool of ESI [58]. A tool that integrates and relates all diagrams and models was lacking. State-of-the-Art architecture modelling tools and Model-Based Systems Engineering tools offer functionality for creating and managing (a subset of) the models.

- Criteria for deciding where to stop adding architectural “details” to the reference architecture. The reference architecture should address strategic architectural decisions for a range of product (lines). When it becomes too detailed, it will unnecessarily limit future developments in product (line) specific systems architectures.

9.4 The PaloAlto2-project

Further validation and several of the extensions listed above will be addressed in the PaloAlto2-project that ESI and Thermo Fisher Scientific will do in 2021. The project will focus on defining a reference architecture for the next generation of transmission electron microscopes. It will start from the reference architecture of today’s systems, which was “distilled” in the PaloAlto project.

In the PaloAlto2-project, we plan to elaborate the methods for the *Business Value* layer and deepen our insights into how business values drive system decomposition. We will take the business need for professional management of configuration diversity (configuration management) as one of the relevant business drivers to shape the reference architecture of the next generation of systems.

Furthermore, we will deepen the methods for the *Customer Value* layer, by analyzing customer values for the next generation of products in a range of application domains and market segments.

Finally, we will address the issue of tool support for our methodology by mapping the diagrams and models we made on the diagrams and models that can be made with state-of-the-art architecture modelling tools. We will do so by creating proofs of concepts: (i) by transferring diagrams and models that were created in the PaloAlto-project and (ii) by using such a tool for the diagrams and models that will be newly created in the PaloAlto2-project. This activity will leverage the learnings from the study on Model-based Systems Engineering (MBSE) that ESI and its industrial and academic partners have started in 2020, which will continue in 2021 in parallel to the PaloAlto2-project (Thermo Fisher Scientific is a participating ESI-partner in this project too).[18]

[18] As this study is running, the amount of publicly available information is limited. First results were shared in the webinar on MBSE that was organized by ESI on October 6, 2021 [81]. Additional results will be shared in the next webinar on MBSE that ESI and SERC will organize on April 16, 2021 *MBSE* Applied as part of ESI's International Digital Enablement Week (April 13-20, 2021) https://esi.nl/events/2021/idew-2021

10 Bibliography

[1] *G. J. Muller, "CAFCR: A Multi-view Method for Embedded Systems Architecting. Balancing Genericity and Specificity," 2004.*

[2] *ISO/IEC JTC 1/SC 7, "ISO/IEC/IEEE 42010:2011 - Systems and software engineering — Architecture description," ISO/IEC/IEEE, 2011.*

[3] *K. Ulrich, "The role of product architecture in the manufacturing firm,"* Research Policy, *vol. 24, pp. 419-440, 1995.*

[4] *L. Laperrière and G. Reinhart, CIRP Encyclopedia of Production Engineering, L. Laperrière and G. Reinhart, Eds., Springer, Berlin, Heidelberg, 2014.*

[5] *R. Sanchez, "Using modularity to manage the interactions of technical and industrial design,"* Design management journal, *vol. 2, pp. 8-19, 2002.*

[6] *C. Y. Baldwin and C. J. Woodard, "The Architecture of Platforms: A Unified View," Boston, 2008.*

[7] *Wikipedia, "Car platform," Wikipedia, 2 November 2020. [Online]. Available: https://en.wikipedia.org/wiki/Car_platform. [Accessed 4 November 2020].*

[8] *R. Sanchez, "Creating modular platforms for strategic flexibility,"* Design Management Review, *vol. 15, no. 1, pp. 58-67, 2004.*

[9] *G. Muller, "Gaudí Systems Architecting," 6 September 2020. [Online]. Available: https://www.gaudisite.nl/HowtoManageablePlatformArchitectureSlides.pdf. [Accessed 4 November 2020].*

[10] *R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole and M. Bone, "The Concept of Reference Architectures,"* Systems Engineering, *vol. 13, no. 1, pp. 14-27, 2010.*

[11] *Office of the Assistant Secretary of Defense Networks and Information Integration (OASD/NII), "Reference architecture description," the Office of the DoD CIO , June 2010. [Online]. Available: https://bigdatawg.nist.gov/_uploadfiles/M0238_v1_7215638225.pdf. [Accessed 11 November 2020].*

[12] *A. Vasenev and T. Hendriks, "Structured problem exploration approach for the pre-concept stage of system development," in* 14th Annual Conference System of Systems Engineering (SoSE), *Anchorage, AK, USA, 2019.*

[13] *A. Vasenev, "Structuring of Methods to Estimate Benefits of Partial Networking," in* Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2018), *Funchal, Madeira, Portugal, 2018.*

[14] *U. Eklund and J. Bosch, "Architecture for embedded open software ecosystems,"* Journal of Systems and Software, *vol. 92, no. 1, pp. 128-142, 2014.*

[15] *J. Bach, S. Otten and E. Sax, "A taxonomy and systematic approach for automotive system architectures-from functional chains to functional networks," in* International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2017), *Porto Portugal, 2017.*

[16] *P. Pelliccione, E. Knauss, R. Heldal, M. Ågren, P. Mallozzi, A. Alminger and D. Borgentun, "Automotive Architecture Framework: The Experience of Volvo Cars,"* Journal of Systems Architecture, *vol. 77, 3 2017.*

[17] *ESI (TNO), "Reference architectures for product families," 2021. [Online]. Available: https://esi.nl/research/output/methods/reference-architectures-for-product-families . [Accessed 12 February 2021].*

[18] *G. Muller, "A Reference Architecture Primer," 3 September 2020. [Online]. Available: https://www.gaudisite.nl/ReferenceArchitecturePrimerPaper.pdf. [Accessed 4 November 2020].*

[19] *G. Muller, "How Reference Architectures support the evolution of Product Families," January 2008. [Online]. Available: https://www.gaudisite.nl/CSER2008_Muller_EvolvabilityRA.pdf. [Accessed 4 November 2020].*

[20] *M. Treacy and F. Wiersema, The Discipline of Market Leaders, Addison-Wesley, 1995.*

[21] *G. Muller, Systems Architecting - A Business Perspective, Taylor & Francis Inc, 2011.*

[22] *H. Obbink, J. Müller, P. America and R. v. Ommeringen, "COPA: A component-oriented platform architecting method for families of software-intensive electronic products," 2000.*

[23] *G. Muller and P. van de Laar, "Researching Reference Architectures," in* Views on Evolvability of Embedded Systems*, P. Van de Laar and T. Punter, Eds., Dordrecht, Springer Netherlands, 2011, p. 107–119.*

[24] *M. E. Conway, "How do committees invent,"* Datamation*, pp. 28-31, 1968.*

[25] *J. M. Juran, "The Quality Function," in* Juran's Quality Control Handbook*, 4th Edition ed., McGraw-Hill, 1988.*

[26] *F. Liu, J. Tong, J. Mao, R. B. Bohn, J. V. Messina, M. L. Badger and D. M. Leaf, "NIST cloud computing reference architecture," 2011.*

[27] *M. Weyrich and C. Ebert, "Reference Architectures for the Internet of Things,"* IEEE Software*, vol. 33, pp. 112-116, 2016.*

[28] *C. Ott, A. Korthaus, T. Böhmann, M. Rosemann and H. Krcmar, "Foundations of a Reference Model for SOA Governance," in* Information Systems Evolution*, Berlin, 2011.*

[29] *M. Staron and D. Durisic, "AUTOSAR Standard," in* Automotive Software Architectures: An Introduction, *Cham, Springer International Publishing, 2017, p. 81–116.*

[30] *J. Tummers, A. Kassahun and B. Tekinerdogan, "Reference architecture design for farm management information systems: a multi-case study approach,"* Precision agriculture*, 2020.*

[31] *M. Irlbeck, D. Bytschkow, G. Hackenberg and V. Koutsoumpas, "Towards a Bottom-Up Development of Reference Architectures for Smart Energy Systems," in* 2nd International Workshop on Software Engineering Challenges for the Smart Grid (SE4SG)*, San Francisco, CA, USA, 2013.*

[32] *ISO/IEC/IEEE, "ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description," 2011.*

[33] *G. Magistrati, "Avionics System Reference Architecture (ASRA)," 2014.*

[34] *B. van der Sanden and A. Vasenev, "Architectural guidance in automotive for privacy and security: survey and classification," 2020.*

[35] *E. S. I. TNO, 2020.*

[36] *T. Bijlsma, B. v. der Sanden, Y. Li, R. Janssen and R. Tinsel, "Decision support methodology for evolutionary embedded system design," in* 2019 International Symposium on Systems Engineering (ISSE)*, 2019.*

[37] *ESI (TNO), University of South-Eastern Norway, Stevens Institute of Technology, "Systems Architecture Forum," [Online]. Available: http://www.architectingforum.org/index.shtml. [Accessed 9 November 2020].*

[38] *G. Muller and E. Hole, "Reference Architectures; Why, What and How," 2007.*

[39] *Secredas Consortium, "SECREDAS," [Online]. Available: https://secredas-project.eu/. [Accessed 5 November 2020].*

[40] *N. Marko, A. Vasenev and C. Strieks, "Collecting and Classifying Security and Privacy Design Patterns for Connected Vehicles: SECREDAS Approach," in* Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, virtual conference, 2020.*

[41] *J. Delange, J. McHale, J. Hudak, W. Nichols and M.-Y. Nam, "Evaluating and Mitigating the Impact of Complexity in Software Models," Pittsburgh, 2015.*

[42] *G. Muler, "Architectural Reasoning; CAFCR," 11 October 2020. [Online]. Available: https://www.gaudisite.nl/ArchitecturalReasoningBook.pdf. [Accessed 23 November 2020].*

[43] *DoD, "Reference architecture description," 6 2010.*

[44] *R. B. Woodruff, "Customer value: The next source for competitive advantage,"* Journal of the Academy of Marketing Science, *vol. 25, p. 139, 01 3 1997.*

[45] *J. Anderson, J. Narus and W. Rossum, "Customer Value Propositions in Business Markets,"* Harvard business review, *vol. 84, pp. 90-9, 149, 4 2006.*

[46] *M. Bertoni and A. Bertoni, "Models for value-driven engineering design," 2016.*

[47] *J. C. Anderson, D. C. Jain and P. K. Chintagunta, "Customer value assesment in business markets:a state-of-practice-study,"* Journal of Business-to-Business Marketing, *vol. 1, p. 3–29, 1992.*

[48] *WikipediA, "Five Whys," 15 September 2020. [Online]. Available: https://en.wikipedia.org/wiki/Five_whys. [Accessed 4 December 2020].*

[49] *F. van der Linden, J. Bosch, E. Kamsties, K. Känsälä and H. Obbink, "Software Product Family Evaluation," in* Software Product-Family Engineering, 5th International Workshop, PFE 2003, Revised Papers*, Sienna, November 4-6, 2003.*

[50] *W. Aalst, "The Application of Petri Nets to Workflow Management,"* Journal of Circuits, Systems, and Computers, *vol. 8, pp. 21-66, 2 1998.*

[51] *Materials Laborotary, "Integrated Computer-Aided Manufacturing (ICOM) Architecture Part II, Volume IV - Function Modelling (IDEF0)," 1981.*

[52] *IEEE Computer Society, "IEEE 1320.1-1998 - IEEE Standard for Functional Modeling Language - Syntax and Semantics for IDEF0," IEEE SA, 1998.*

[53] *National Institute of Standards and Technology, "Integratration Definition for Function Modelling (IDEF0)," 1993.*

[54] *ISO/IEC JTC 1/SC 7, "ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," ISO/IEC, 2011.*

[55] *D. Parnas, "On the criteria to be used in decomposing systems into modules,"* Communications of the ACM*, 1972.*

[56] *Object Management Group, "SysML - OMG System Modeling Language - About the OMG System Modeling Language Specification Version 1.6," [Online]. Available: https://www.omg.org/spec/SysML. [Accessed 22 December 2020].*

[57] *L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modelling Language, Addison-Wesly, 2013.*

[58] *ESI, "DAARIUS Methodology," ESI, [Online]. Available: https://esi.nl/research/output/methods/daarius-methodology. [Accessed 23 November 2020].*

[59] *F. Linden, "Linden, F.: Software product families in Europe: the Esaps and Cafe projects,"* Software, IEEE, *vol. 19, pp. 41-49, 8 2002.*

[60] *F. Linden, J. Bosch, E. Kamsties, K. Känsälä and H. Obbink, "Software Product Family Evaluation," in* Software Product Lines*, Berlin, 2004.*

[61] *S. Sheard, "he Value of Twelve Systems Engineering Roles," in* INCOSE International Symposium*, July, 1996.*

[62] *G. Muller, "The Role and Task of the System Architect," 21 June 2020. [Online]. Available: https://www.gaudisite.nl/RoleSystemArchitectPaper.pdf. [Accessed 4 February 2021].*

[63] *G. Muller, "Gaudí Systems Architecting - Architecting for Business Value," [Online]. Available: https://www.gaudisite.nl/ABV.html. [Accessed 5 October 2021].*

[64] *S. Chowdhury, Design for Six Sigma: The revolutionary process for achieving extraordinary profits, Prentice Hall,, 2002.*

[65] *J. Lamm and T. Weilkiens, "Funktionale Architekturen in SysML," in* Tags des Systems Engineering, *München, Germany, November 2010.*

[66] *M. Erden, H. Komoto, T. van Beek, V. d'Amelio, E. Echavarria and T. Tomiyama,* Artificial Intelligence for Engineering Design, Analysis and Manufacturing, *vol. 22, pp. 147-169, 2008.*

[67] *Y. Umeda and T. Tomiyama, "Functional Reasoning in Design,"* IEEE Expert, *March-April 1997.*

[68] *P. E. Vermaas, "The coexistence of engineering meanings of function: Four responses and their methodological implications,"* Artificial Intelligence for Engineering Design, Analysis and Manufacturing, *vol. 27, pp. 191-202, 2013.*

[69] *B. Eisenbart, K. Gericke and L. T. Blessing, "Taking a look at the utilisation of function models in interdisciplinary design: insights from ten engineering companies,"* Res Eng Design, *vol. 28, pp. 299-331, 2017.*

[70] *N. Chiriac, K. Höltta-Otto, D. Lysy and E. S. Suh, "Three Approaches to Complex System Decomposition," in* 13th International Dependency and Structure Modelling Conference, DSM'11, *Cambridge, MA, USA, September 14-15, 2011.*

[71] *B. Penzenstadler, "DeSyRe: Decomposition of Systems and theirRequirements— Transition from System to Subsystem usinga Criteria Catalogue and SystematicRequirements Refinement," 2011.*

[72] *ISO/IEC JTC 1/SC 7: Software and Systems Engineering, ISO/IEC 25010:2011 Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, ISO/IEC, 2011-03.*

[73] *R. Kazman, M. Klein and P. Clements, "ATAM: Method for Architecture Evaluation," August 2000. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf. [Accessed 4 January 2021].*

[74] *B. van der Sanden and A. Vasenev, "Architectural Guidance in Automotive for Privacy and Security: Survey and Classification," in* Proceedings of the 2020 IEEE 14th International Systems Conference (SysCon), *Virtual Conference, 2020.*

[75] *P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Almiger and D. Borgentun, "Automotive Architecture Framework: The experience of Volvo Cars,"* Journal of Systems Architecture, *vol. 77, pp. 83-100, June 2017.*

[76] *M. Verhoeven, "MBSE in a platform driven company," 6 October 2020. [Online]. Available: https://downloads.esi.nl/webinar/webinar-2-verhoeven-20201006-def.pdf. [Accessed 11 November 2020].*

[77] *P. America, H. Obbink, R. van Ommering and F. van der Linden, "CoPAM: A Component-Oriented Platform Architecting Method Family for Product family Engineering," in* SPLC 1, *Denver, Colorade, USA, 2000.*

[78] *G. Muller,* How to Create a Manageable Platform Architecture?.

[79] *Cambridge University Press, "Cambridge Dictionary - to distil," Cambridge University Press, 2021. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/distil. [Accessed 11 January 2021].*

[80] *J. Sobieszczanski-Sobieski, "OPTIMIZATION BY DECOMPOSITION: A STEPFROM HIERARCHIC TO NON-HIERARCHIC SYSTEMS," 1988..*

[81] ESI (TNO), "Webinar 2 - MBSE Adoption and Added Value," 6 October 2020. [Online]. Available: https://esi.nl/events/2020/webinar-2-2020. [Accessed 5 February 2021].