Right Sizing Reference Architectures;

How to provide specific guidance with limited information.

Prof. Gerrit Muller
Buskerud University College and Embedded Systems Institute
Gerrit.muller@embeddedsystems.nl

Abstract. The growing complexity and size of systems and the organizations that create these systems trigger the need for instruments that facilitate the creation of these systems. We shortly analyze these trends and the (potential) role of reference architectures as facilitating means. After a short discussion about the content of Reference Architectures and their relation with system architectures, system design and actual systems, we zoom in on the question what the appropriate level of detail is for Reference Architectures. We discuss an evolutionary approach to create and maintain Reference Architectures

Introduction

Reference Architectures are one of the means to cope with the increasing size and complexity of the development of systems. Especially when a multitude of context constraints are imposed, such as multi-site, multi-vendor, and multi-organization, *Reference Architectures* form a powerful instrument. The trend towards more integration and more dynamics further increases the value of *Reference Architectures*.

Reference Architectures capture proven architectural patterns and use these in an architectural blueprint. The blueprint provides guidance to multiple organizations, and multiple sites how to create new products or product families, or how to extend existing products. One of the values of *Reference Architectures* is to provide a common architectural vision, and common lexicon and taxonomy, to the heterogeneous set of involved stakeholders.

Reference Architectures also support the increased integration of systems, where the main challenge is to achieve interoperability between different and evolving systems. *Reference Architectures* explicitly articulate domain and realization concepts, and model functions and qualities above system level. Together with explicit decisions about compatibility, upgrades, and interchangeability a framework is created to integrate different systems in useful ways.

Reference Architecture Content

Reference Architectures go well beyond the system boundaries, because specification and design decisions have their rationale in the system context. Figure 1 shows a decomposition of a Reference Architecture into *technical architecture*, *customer context*, and *business architecture*. The *technical architecture* captures design pattern and *technology* considerations. The interface between the *technical architectures* and the *contexts* are the *requirements*, which provide a *black box view* on the systems. A *Reference Architecture* captures the essence of the usage domain, with information about for instance *customers*, *users*, or the *enterprise* context. The *business architecture* captures the underlying business

principles, such as *business models* and product *life cycle* considerations.

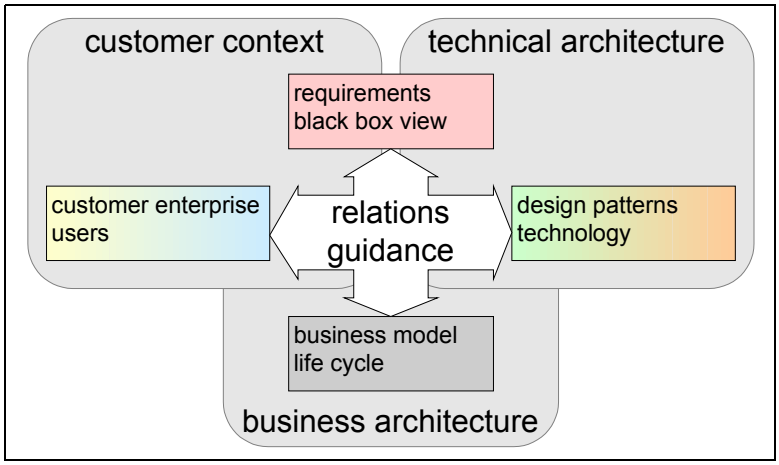

Figure 1, Reference Architectures address the system itself as well as the contexts of the system.

The information to be captured in a Reference Architecture we mentioned so far is interdependent. The relevant relations between all pieces also have to be captured to provide practical guidance in using the different kinds of information in Reference Architectures.

Distance between Reference Architectures and actual systems

Reference Architectures capture domain specific knowledge that is generic for multiple systems. This attempt to make the knowledge usable across multiple systems makes the knowledge less directly deployable. The insight of the re-usable knowledge has to be transformed in more concrete guidelines, these guidelines are used to design systems and the designed systems have to be build and tested before we actually have a real (physical) system. Figure 2 shows this process of concretization.

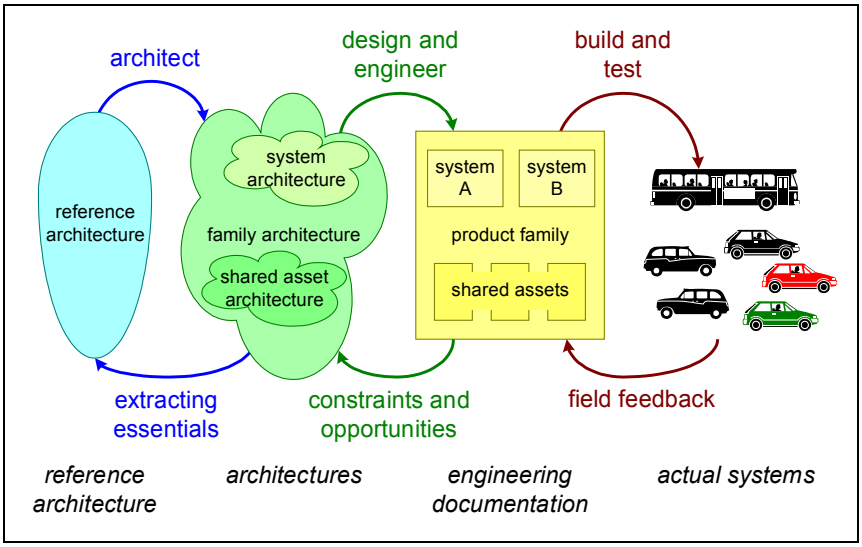

Figure 2, Reference Architectures capture the essence of actual family, system and platform architectures. Several concretization and instantiation steps are required to turn a Reference Architecture in actual systems.

Figure 2 also shows that knowledge emerges in the other direction. We learn from using actual systems. This feedback translates into architectural constraints and opportunities. Existing architectures are mined for the essential domain knowledge that can be generalized across products.

Note that Figure 2 shows system architectures, family architectures and shared asset (component) architectures all at the same distance of actual products. Family architectures contain concrete rules, conventions and guidelines to design product members. Figure 3 shows that these existing architectures are mined for the essence: what are common problems and what are proven solutions, see [Cloutier 2006]? When is a solution appropriate?

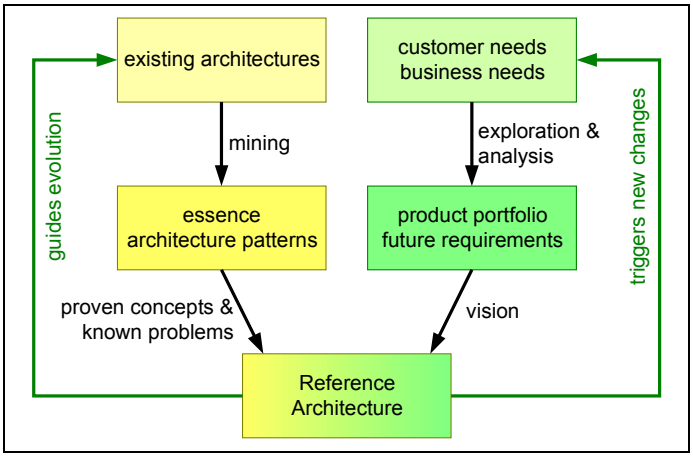

Figure 3, Reference Architectures are based on mining proven solutions in existing architectures. The vision on future needs is used to further shape the reference architecture.

Level of detail and content in current system architectures

The main challenge in creating Reference Architectures is to provide information that actually is useful for design across systems. If we look at one single individual system, then the number of details in contemporary systems is in the order of tens of millions[1]. Figure 4 shows that we can look at such a system at several levels of abstraction. Typical detailed system descriptions are at mono-disciplinary design level: millions lines of SW code, millions of connections and parts at E-CAD or M-CAD level. If we look at much higher abstraction levels than we can describe the system with its system functionality and system characteristics. Between system level and mono-disciplinary design we have to make a multi-disciplinary design step, where we make multi-disciplinary choices and allocate functions and properties to more detailed designs, until we reach the level of mono-disciplinary engineering.

We will pose several observations about abstractions in practice based on working in and with tens of companies and providing education to tens of companies

Observation 1: In nearly all companies explicit systems architecture documentation is either at rather detailed level or at very high level. High level information is disconnected from the detailed level of information.

[1] Of course this depends strongly on the type of systems, airplane designs may contain a few orders of detail more than for example high volume printers.

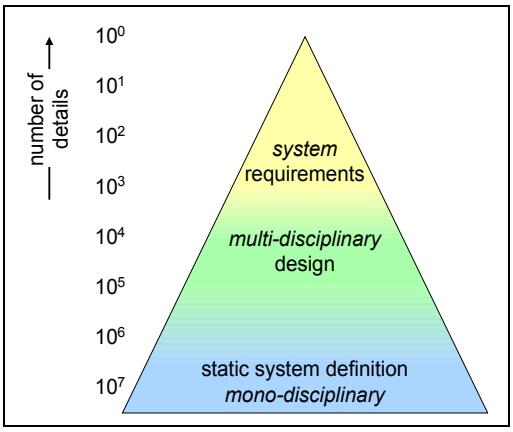

Figure 4, the number of details in the design of current systems.

Level of detail and content in Reference Architectures

If we broaden the scope from a single system to a family of systems or even a portfolio of systems, then we increase the number of details of all these systems with one or two orders of magnitude. Figure 5 shows this increase. Figure 5, however, also adds the number of details of the different contexts to the figure. The context is added to the top, with the number of details increasing in the other direction. This visualization shows that the system specification is an abstraction in two dimensions: we abstract from technical details and we abstract from context details. For example in the usage context we might describe an operator, but in the field we might have millions of operators, ranging from Chinese or Korean individuals to Latin American or Dutch individuals, ranging from young to old, ranging from highly educated to lower class, ranging from skilled to handicapped, et cetera. We have to abstract in both dimensions in order to cope with this the tremendous complexity of the contexts and realizations.

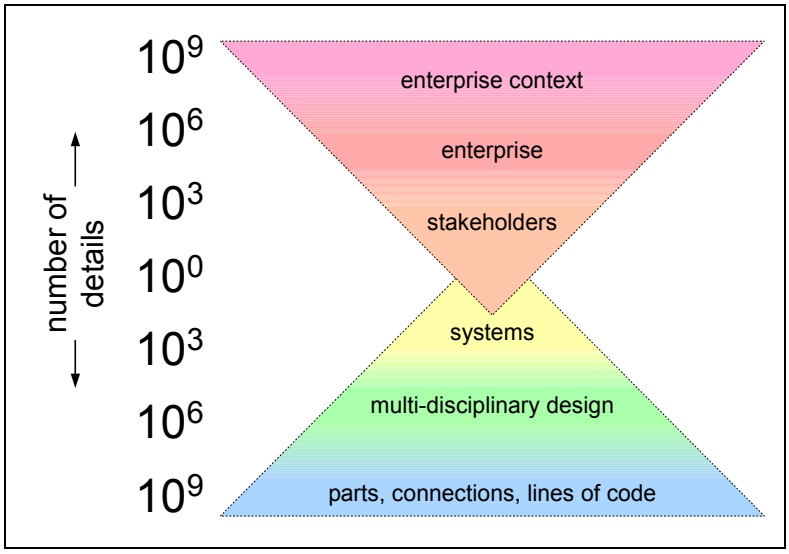

Figure 5, the number of details is a family or portfolio of systems plus the number of details in the contexts of these systems.

Observation 2: In many companies, especially the larger companies design and engineering departments have very little insight in customer context and business architecture.

Requirements specifications serve as interface effectively isolating technical people from customer and business.
A lot of the art of architecting is related to making abstraction choices, when is an abstraction appropriate, when will an abstraction back fire? Observation 1 and 2 show that the current state-of-practice in architecting is that it is still very difficult to describe systems and their contexts at multiple levels of abstraction.
Reference Architectures lift the ambition level even higher: capturing the essence of the system architectures and their contexts at a level of detail that provides guidance without drowning in the details. In terms of Figure 5 the Reference Architecture mostly belongs in the middle, capturing at sufficient level details of system and contexts. However, every architecture is as good as it deals with the most critical or important details. Few of the mono-disciplinary details and usage details may show up in the reference architecture. The crucial question for Reference Architectures is how many details and how much abstraction is appropriate?
Figure 6 shows two examples of Reference Architectures at different levels of details. At the left hand side a compact Reference Architecture is shown with about thousand details, distributed over seven views or models. The right side shows a version with about one million details distributed over 29 more elaborated views or models. In both figures the views and models address issues from technological nature (e.g. decomposition, concurrency and synchronization) up to customer context (e.g. key drivers and application process flow).

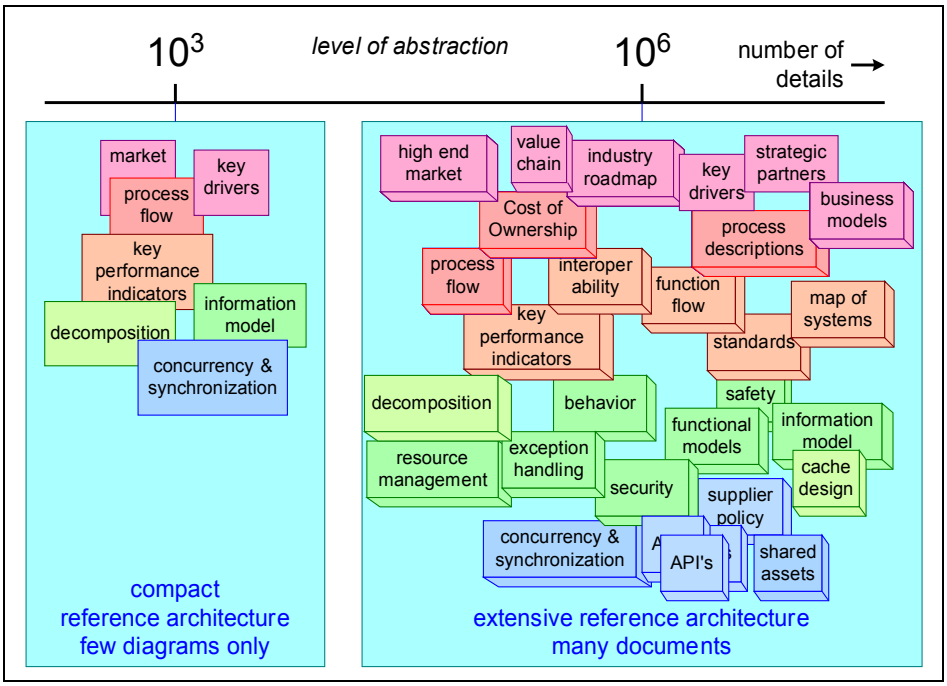

Figure 6, Reference Architectures at two different levels of abstraction; left hand side 7 models/views with limited detail or right hand side 29 more extensive documented views/models.

The version with thousand details provides high level insight and overview of the domain and the systems. Because the size is rather limited such Reference Architecture is easily maintained and kept up-to-date. Unfortunately, the high level nature of the information means that there is quite a distance from this overview to the realization concerns of

designers. The guidance value for them is quite limited; the value is mostly to provide background information and a more global positioning.
The Reference Architecture with one million details captures much more information about the system and its contexts. As a consequence it will be more work to create and maintain. If we make a very rough estimate using software productivity metrics (few thousand lines of code per man-year, assuming that effort 1 line of code ~= effort 1 detail), then we estimate that such Reference Architecture requires hundreds of man-years to create. Maintenance cost to keep up-to-date will be proportional to the creation effort, many man-years per year. Also bad is that reading and using such more extensive description also takes more time. The benefit of this extensively elaborated Reference Architecture is that the distance to actual realization is much smaller. This level of abstraction actually might provide direct guidance to product designers.

Recommendations for right sizing Reference Architectures

Budgeting the amount of effort

Based on experience we have seen that we can provide figures of merit for the ratio between system engineers (or architects) and specialized designers. A healthy ratio is that one out of twelve engineers is system oriented rather than specialized. These system engineers should spend most of their time on creating and maintaining the architecture and in using the architecture to design, engineer, build, integrate and test the actual systems. Roughly 80% of the time of a system engineer should be used for this main task. The remaining 20% of the time is divided over several other activities, such as technology forecasts, feasibility studies, process improvement, and strategic discussions. Part of this 20%, about a quarter, can be used to create and maintain a reference architecture.
Let's assume that we have an organization of thousand engineers. Then about 80 of them should be system oriented. In average they spend 5% (one quarter of 20%) of their time on creating a reference architecture. This is about 4 man years of work per year.

Evolutionary approach

The available capacity is needed for mining, processing the results, and creating the reference architecture description. Also capacity is required for exploring the customer and business needs, processing and capturing these needs, and merging this information in the reference architecture description. The recommended capacity is insufficient to create an entire reference architecture that covers the full breadth of system and context considerations. We recommend using the available effort in short focused time-boxes[2]. The subject of the time-box should be well-connected to hot issues in ongoing architecting efforts. In this way the reference architecture is created incrementally. However, when the growing reference architecture indeed connects well with ongoing work, then the system engineers will get lots of feedback. The feedback should be processed; this triggers an evolution of the reference architecture.

[2] The technique of time-boxing allocates a fixed amount of time a priori for an activity. As a consequence this activity will not be completely finished at the end of the time-box. Time-boxing is combined with rapid iterations: repeat a set of activities several times to improve the quality of the results. Core to the combination of time-boxing and rapid iteration is that insights obtained in one activity benefit another activity. For example, understanding customers provides insight in requirements, but also insight in requirements help to focus the understanding of the customer. Time-boxes can be as short as 15 minutes or as long as several days or even weeks.

The evolutionary approach is similar to evolutionary and incremental approaches such as EVO [Gilb 2005] and SCRUM. Some additional guidance is required to achieve the right size:

Time-boxes are used to limit the amount of time spend on one subject. One of the major pitfalls is, also at this level, analysis paralysis.

Feedback is obtained by early exposure and use of the results. The analysis paralysis effect may also cause delay of exposure and use. Late feedback might be killing: if we discover that we have been addressing the wrong issues after spending significant amount of critical capacity, then we lose the credit to work on these longer term assets. Feedback is also necessary to find the appropriate level of abstraction: if the results are too high-level to provide guidance, then we will have to improve our previous work.

Reflection and planning is required to prevent too much ad hoc work, where we might follow from crises to crises, instead of bringing some stability by capturing past know how. Reflection is used to created overview: what is the overall situation, what have we done, what are the most important or critical issues to address. The insight gained in reflection is used to make and communicate a rough plan.

Framework(s) can be used as checklist for content. We don't recommend to follow these frameworks rigorous, since most frameworks have been created for different purposes. Potential frameworks range from DoDAF [DoD 2003] or Zachman [Zachman 1987] to CAFCR [Muller 2004]. Core to any framework is the multi-dimensional nature of architectures and the need for multiple complementary representations to capture content. Note that reference architectures inherently have at least the same set of dimensions as system architectures, but the broader scope might even add dimensions.

Summary and Conclusion

The increasing size of complexity of the systems as well as of the organizations that create the systems trigger the need for guidance across organizations, sites, vendors et cetera. Reference architectures capture the essence of past system architectures and add a future direction based on customer and business needs. A major challenge is to keep the reference architecture as compact as possible, while still providing practical guidance. This is already a challenge for system and family architecture descriptions, but this is even more challenging for Reference Architectures.

We recommend an evolutionary approach to Reference Architectures, with time-boxes to avoid analysis paralysis, early feedback to ensure usability, reflection and planning to avoid ad hoc firefighting and the use of frameworks as checklists to ensure proper coverage.

Acknowledgements

Discussions in the System Architecting Forum provided a lot of input for this paper, see [Muller 2007].

References

[Cloutier 2006] Robert Cloutier, *Applicability of Patterns to Architecting Complex Systems*, Doctoral Dissertation, Stevens Institute of Technology, Hoboken, NJ, 2006.

[DoD 2003] DoD Architecture Framework, Volume 1: Definitions and Guidelines, Version 1 US Dept. of Defence, 2003.

[Gilb 2005] Thomas Gilb, *Competitive Engineering*, Elsevier, 2005

[Muller 2004] Gerrit Muller, *CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity*, Ph.D. thesis, http://www.gaudisite.nl/ThesisBook.pdf, 2004.

[Muller 2007] Edited by: Gerrit Muller and Eirik Hole, *Reference Architectures; Why, What and How*, White Paper Resulting from Architecture Forum Meeting March 12-13, 2007 (Hoboken NJ, USA), http://www.architectingforum.org/whitepapers/SAF_WhitePaper_2007_4.pdf, 2007.

[Zachman 1987] John Zachman. *The Zachman framework for enterprise architecture*. http://www.zifa.com/, 1987.

Author Biography

Gerrit Muller.

Gerrit Muller received his Master's degree in physics from the University of Amsterdam in 1979. He worked from 1980 until 1997 at Philips Medical Systems as a system architect, followed by two years at ASML as a manager of systems engineering, returning to Philips (Research) in 1999. Since 2003 he has worked as a senior research fellow at the Embedded Systems Institute in Eindhoven, focusing on developing system architecture methods and the education of new system architects, receiving his doctorate in 2004. In January 2008 he became a full professor of systems engineering at Buskerud University College in Kongsberg, Norway.
All information (System Architecture articles, course material, curriculum vitae) can be found at:
Gaudí systems architecting http://www.gaudisite.nl/