

A Conceptual Framework for Architecture Alignment Guidelines

Project GRAAL WP1 Whitepaper

P. A. T. van Eck¹ (editor) H. Blanken¹ M. Fokkinga¹
P. W. G. Grefen¹ R. J. Wieringa¹

October 17, 2002

¹Department of Computer Science, University of Twente, P.O. Box 217, 9700 AE Enschede, the Netherlands, (blanken|vaneck|fokkinga|grefen|roelw)[@cs.utwente.nl](mailto:)

Contents

1	Introduction	3
2	Conceptual framework	5
2.1	Architecture alignment: A definition	5
2.2	Four-dimensional architecture descriptions	6
2.2.1	The lifecycle dimension	7
2.2.2	The service layer dimension	8
2.2.3	The aspect dimension	10
2.2.4	The refinement dimension	10
2.3	Design Charter and Environment	13
A	Summary of Models	17

Chapter 1

Introduction

Current research in software architecture concentrates on the design and evaluation of software architectures with respect to a set of desired quality attributes and properties of the development organization [1, 2, 4]. This research is mainly concerned with designing software systems embedded in a technical environment, such as embedded control software. Moreover, the development situation is assumed to be a greenfield situation and focus is on translating requirements into structures of custom-made components.

In business environments, however, it is almost never possible to assume a greenfield development situation. In business environments, existing components are configured and deployed by embedding them into existing hardware and software infrastructures. These components are often large-grained standard components that are acquired from outside sources, or are legacy components. The resulting architecture must be aligned with business strategies and processes, including the development organization, and even with the external business environment.

Alignment between information systems and business has been an issue since the 1970s. In the last ten years, most of the work in this area has taken the approach of Henderson and Venkatraman [3]. This approach provides clarity at the strategic level but does not tell us how to partition software into applications and how to allocate these to the available implementation platform. The close intertwinement of external and internal business processes and application software in modern service organizations, however, requires a seamless integration of these systems with the business at the operational level.

In this report, we intend to study the design and evaluation of application architectures in alignment with their wider business context. This not only includes quality attributes, but also includes the architecture of business processes, legacy software, legal requirements, and business strategy. In addition, compared to most current research, our study takes a larger set of stakeholders into account, which not only include software designers and programmers, but also users, project management, and the maintenance organisation. Our study focuses on information systems for the financial service business, including banks, insurance companies and the Dutch tax department.

The goal of the work reported in this whitepaper is to develop a framework

for experimental research on architecture alignment. In this research, practical guidelines for alignment of application architectures and business architectures are to be assembled and validated. These guidelines address a number of issues with respect to alignment, which include the following:

- Which design principles relate business processes to software architectures?
- What is the relationship between the surrounding business environment and software architecture?
- Can we find patterns that relate domain properties to properties of the software architecture?
- How is impact analysis done, and can we find general guidelines for this?
- What documentation is currently produced, and can we find guidelines to manage this?

The results of our experimental research should allow businesses to manage the alignment of their software architectures to their business architectures at an operational level. We envision our results to comprise a catalog of real-world application architecture in the financial sector, related to aspects of their business context, as well as a set of guidelines for finding these architectures and instruments for analyzing the impact of architectural decisions on the business context. For the kind of research described above, a framework is needed that provides concepts for the representation of application architectures, their business context, and guidelines. This report presents the conceptual framework that will be used in future experimental research.

We expect to update this report with a comparison with frameworks that we encounter in our research.

Chapter 2

Conceptual framework

2.1 Architecture alignment: A definition

In our study, we are primarily interested in the alignment of software applications with the environment in which they operate. We use the term *software application* to refer to any automated system that supports business functions and that directly interacts with people who work in the business, or actors outside the business such as clients or the clients' software applications. Examples range from relatively small scale applications such as a sophisticated data analysis application used by a handful of staff employees, to enterprise-wide information systems such as ERP, SCM or CRM systems. Software that only exists to support other software, such as operating systems, middleware and database management systems, is outside the scope of our definition. We assume that in a modern business, every software application is somehow connected to at least one other system; isolated systems do not exist.

Figure 2.1 depicts a system consisting of a number of software applications and their environment, which itself consists of an information technology infrastructure and actors that use the software applications. Each software application uses services provided by their supporting IT infrastructure. In turn, each of the software applications provides services to actors in its environment to support business functions they perform. In Figure 2.1, these actors are depicted in a human-like way. However, these actors can also be software applications of other businesses. As the double arrows between software applications indicate, software applications can also provide services to one another.

In any business, changing market characteristics call for new software applications to support new services provided to the market. Similar to existing software applications, new applications operate in an environment as depicted in Figure 2.1. Any new software application, therefore, has to be fitted in this environment. The process of fitting a software application in its environment is what we call (application) architecture alignment. To be more precise, we use the following definition:

Application architecture alignment is the process that determines the optimal fit between a software application and the existing software and business environment.

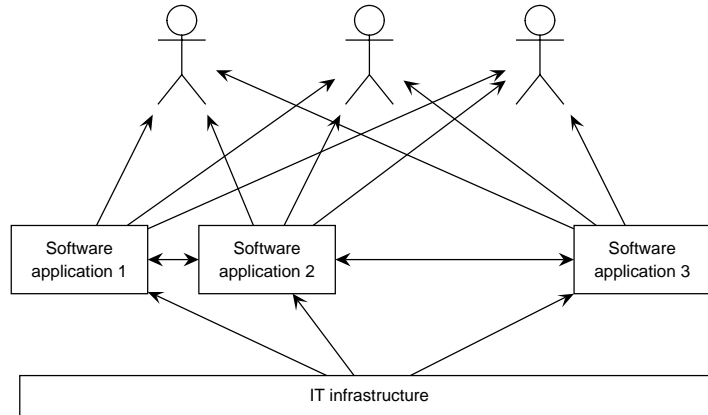


Figure 2.1: Software applications and their environment.

Optimal fit can be obtained automatically if a new software application can be realised completely out of custom-made components. In this case, optimal fit is defined by a set of traditional functional and non-functional requirements. Assuming that the set of requirements is complete and precise, carefully carrying out a traditional software development model such as the waterfall model will result in a new application that fits optimally in its environment.

However, in a modern business context, more often than not, this approach is not feasible for economical reasons: cost and time constraints prohibit a software application that is completely custom built. A new software application will be composed of coarse-grained standard components, the characteristics of which cannot be hidden inside the new software application. In this case, pre-existing components, or sometimes even whole software applications, have to be literally fitted inside the existing environment.

Compared to traditional software development, architecture alignment puts much more emphasis on environment modelling. Software requirements are implied by the environmental models. In addition, the waterfall model is replaced by a model in which ensembles of pre-existing components [7] evolve concurrently with the environmental models until an optimal fit is found.

2.2 Four-dimensional architecture descriptions

We distinguish four dimensions on which a system can be described: the lifecycle (process or time) dimension, the aspect dimension, the service layer dimension, and the refinement dimension. These dimensions, which are described in more detail in the remainder of this section, are motivated as follows.

An architecture description describes a system, on which a static and dynamic view can be distinguished. The dynamic view describes the process in which the system is planned, designed, realised and deployed. This process manifests itself in the lifecycle phases of the system. The lifecycle phases constitute the lifecycle dimension. The lifecycle dimension is elaborated in Section 2.2.1.

The static view describes the system as it is at a specific moment in time.

2.2. FOUR-DIMENSIONAL ARCHITECTURE DESCRIPTIONS

This view describes both external aspects and internal aspects of the system. The external aspects constitute the second dimension of architecture descriptions. As described in Section 2.2.3, we distinguish five external aspects in this dimension: functions, dictionary, quality, behaviour and communication. The only internal distinguished in our framework is the internal composition structure of the system. As explained in Section 2.2.2, we place the components of a system at different service layers. These service layers constitute the third dimension.

The fourth dimension is the refinement dimension, which refers to the amount of detail the description of the system provides. Unlike the other dimensions, this dimension is strictly a description dimension: it is not derived from the static or dynamic view on the system that is described. We call the three dimensions derived from these view system dimensions. The system dimension are depicted in Figure 2.2.

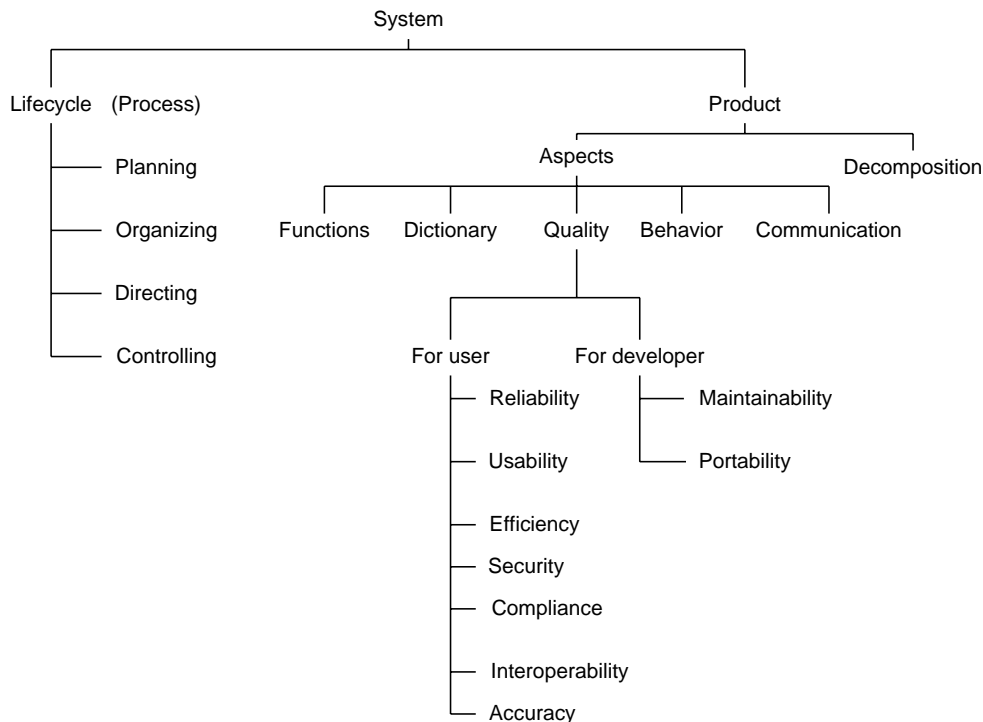


Figure 2.2: Overview of process and product dimensions.

2.2.1 The lifecycle dimension

The four dimensions describe systems as existing artefacts, i.e. as products, or results of processes in the lifecycle (planning, developing, implementing and decommissioning). In our framework, this process is present in a declarative way in the lifecycle dimension. The lifecycle dimension describes in what phases a system can be in the process, but not how this process is planned, organised, directed and controlled. See for instance [5, 6] for extensive treatment of how

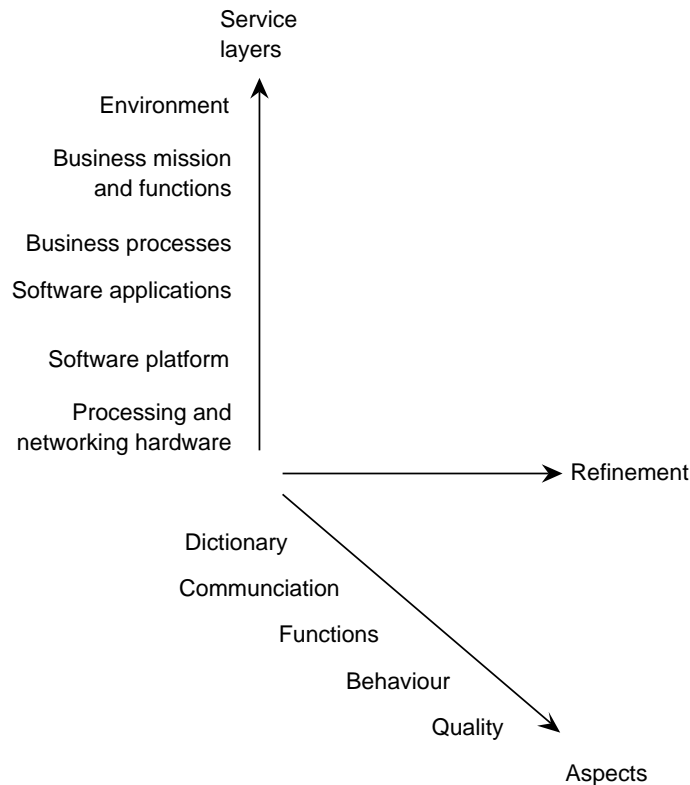


Figure 2.3: Three dimensions of architecture descriptions.

to organise the process of developing architectures.

In the study of architecture alignment, the place of a system on the lifecycle dimension can be considered constant. For instance, in an application of our framework, a recently deployed system is evaluated, or a planned system is analysed. The place of the system in the lifecycle dimension is thus stable and clear. In the description of our conceptual framework, this dimension is therefore kept implicit. The three remaining dimensions are depicted in the form of three axes in Figure 2.3.

2.2.2 The service layer dimension

At the service layer dimension, we distinguish six service layers. The relationship between the layers is that lower layers provide services to higher layers. Services at one layer may be used by several entities at the higher layers. The layers and the interfaces between the layers are depicted in Figure 2.4. The systems depicted by solid lines are the systems under consideration. The systems depicted by dashed lines indicate that at each layer, systems also interact with other systems at the same layer. The layers and interfaces are described below.

- The *environment* consists of the value chain in which the business operates. This includes clients and client groups, suppliers, competitors,

2.2. FOUR-DIMENSIONAL ARCHITECTURE DESCRIPTIONS

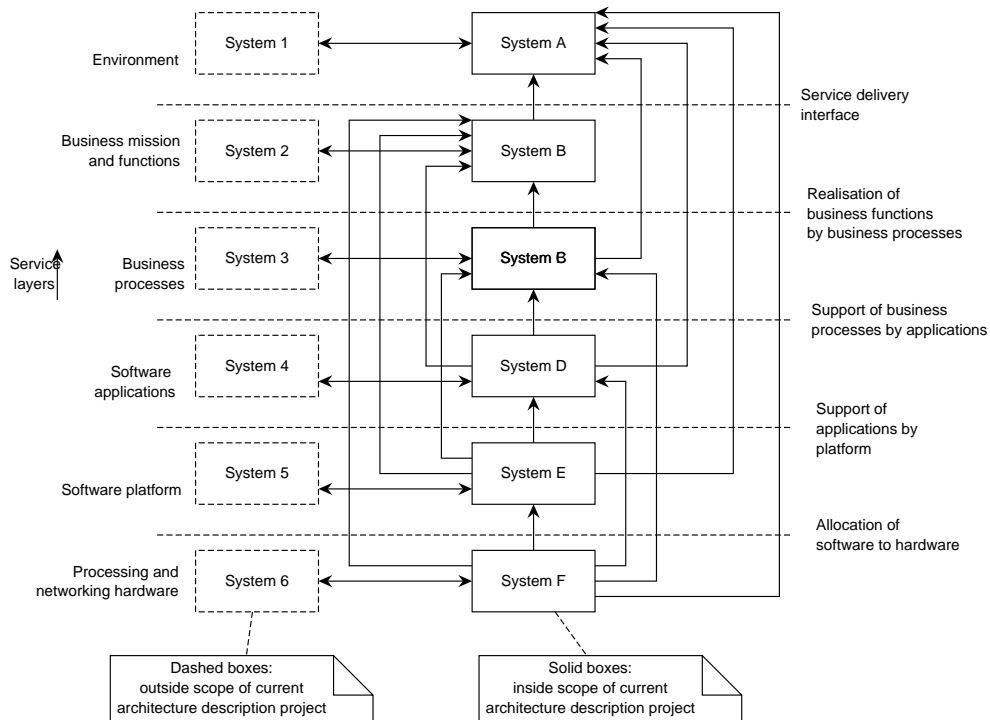


Figure 2.4: Systems in a layered view.

government bodies, distribution- and communication channels.

- The business is an organization of people and machines with a common purpose to deliver a product or service to a market. The common purpose is stated in a shared *business mission* statement. The delivery of products or services is summarized in a list of external functions.
- *Business* processes are
 - operational processes that respond to external and temporal events to deliver products and services,
 - supporting processes, and
 - strategic and tactical management processes.
- *Software applications* support or fully perform parts of the operational and other business processes.
- The *software platform* is the collection of standard general-purpose software needed to run the application software. It is also called “implementation platform”. It ranges from operating systems, middleware, network software to database management software.
- The *processing and network hardware* consists of the physical resources that run the software platform and the application software. “Physical” means “having a size and weight”. The network consists of boxes that

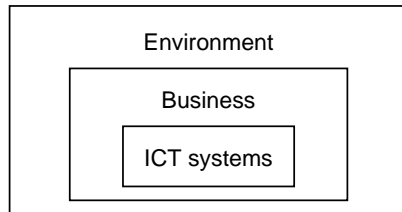


Figure 2.5: Organizations as isolated Chinese boxes.

contain metal, plastic and silicon, glass screens, copper wires and other physical entities.

In Figure 2.4, each line between two layers generally represents a many-many mapping between the two adjacent layers.

- The service delivery interface maps events in the environment to services to be delivered by the business in response to those events.
- Functions are realized by business process. One function may need several processes and one process may support several functions.
- Each application supports several business processes and each process can be supported by several applications.
- Each application uses several software platform entities and each platform entity supports several applications.
- Each software system (application or platform entity) runs on one or more network nodes and each such node runs several software systems.

This distinguishes layering from decomposition (Figure 2.5). In a decomposition, the subsystem provides services to only the composed system of which it is a part and not to other systems. The interface of the composed system hides the subsystem from the view of other systems in the environment. The layered view makes it possible to see that entities at any layer (say application software) can be used by many entities at the next higher layer (e.g. many business processes). By breaking the bounds of the organization, we may also see that our application software can be used by business processes in other firms.

2.2.3 The aspect dimension

For each system, a number of aspects of the system can be distinguished. In our framework, we use the five aspects distinguished in many analysis methods (as identified in for instance [8, 9]): function, behaviour, communication, information and quality. The aspect dimension is related to architectural views as present in many other architecture frameworks: an architecture view, such as for instance the communication view, and the information view, consist of an integrated description of the same aspect of a system at different service layers.

2.2. FOUR-DIMENSIONAL ARCHITECTURE DESCRIPTIONS

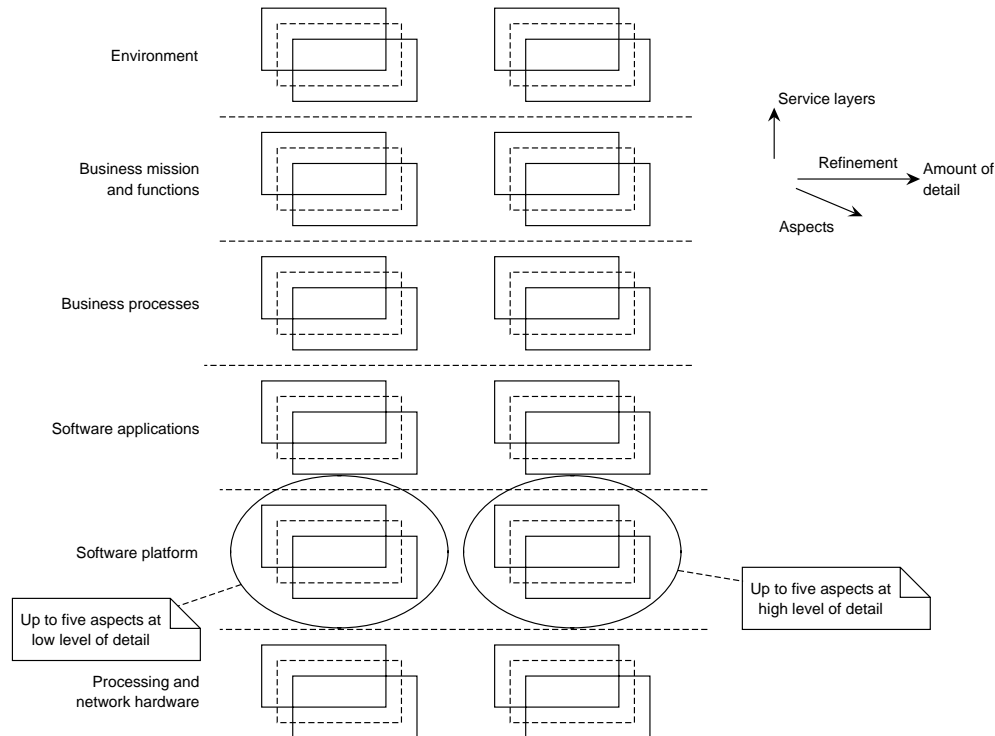


Figure 2.6: Refinement. Per layer and refinement level, any number of aspects may be present, including no aspect at all.

2.2.4 The refinement dimension

The refinement dimension reflects our methodological approach to architecture description, which is both top-down and middle-out (Figure 2.6). It is top-down in the sense that we start with modeling the environment and proceed to lower layers. It is also top-down in that we start with descriptions that keep a high level of abstraction and proceed to descriptions that have a lower level of abstraction (which is a high level of detail). The trick to make it a middle-out process is to see that refinement is independent from layering. We can describe any layer at any level of detail. When we describe lower service layers, we generally require more detail at this layer and at all higher layers. So our initial environment description is quite abstract, but when we describe the application software, we need a more detailed description of the business environment and of the business processes.

When we add detail, we must choose the *aspect* about which we want to add detail. For example, when we add detail about the environment, we can add technical details (e.g. which communication technology is used), or legal details (e.g. which laws are applicable), or financial detail (where do cash flows come from and go to) etc.

The choice of aspect that we choose to detail depends upon the kind of

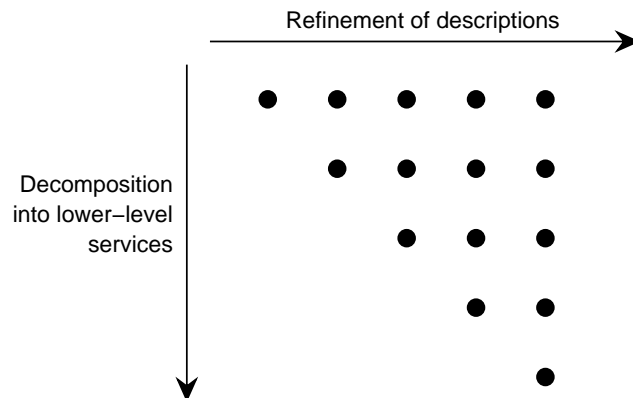


Figure 2.7: Description of lower layers requires adding more detail to descriptions of higher layers.

lower-layer service that we wish to describe. For example, if we describe the services of application software, we will need other kinds of detail about the business environment and business processes than we need when we describe the kind of furniture to be put in the building, or when we describe the social processes in the business. To describe application software, we need a model of the entities about which the software must store information and a model of the processes that will provide the software with data about these entities. When we describe the desired furniture, we need information about the image that the business wants to project to its customers, and about the ergonomic properties required by the people who perform the business processes.

Figure 2.7 illustrates this. We start at the upper left corner with a simple description of the environment. When we describe the business mission and function, we may want to add more detail to the environment description, such as the communication channels through which the business communicates with other parties. Descending to the business processes, we may want to add yet more detail to the environment model, for example by adding a list of relevant external events to which the business should respond. Etcetera.

In the appendix, we summarize the models we think are relevant for each layer and level of refinement. One description we mention at the outset: The dictionary. From our very first environment model, we will start a dictionary of important terms, that will be extended as we add more detail. The dictionary supports the primary function of the models: Communication between people.

2.3 Design Charter and Environment

A second elaboration of the simple top-down approach is that we do not start from scratch. In a greenfield approach, the environment would be given and the business is to be designed. As a matter of fact, the environment and many parts of the business are given and some parts of the business may be redesigned.

2.3. DESIGN CHARTER AND ENVIRONMENT

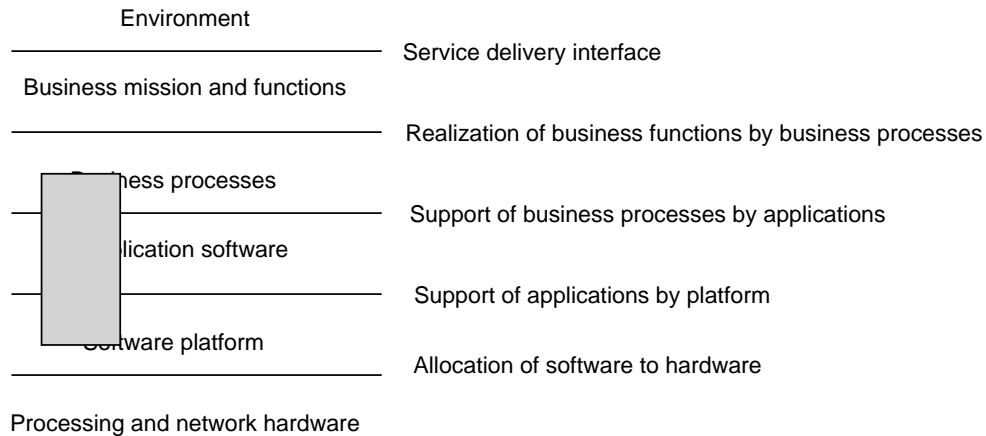


Figure 2.8: Design environment and charter for change.

For example, we may have a charter for changing only the grey rectangle in figure 2.8 but must leave the rest of the business unchanged. These unchanging entities and processes are the *environment* of our design activity. It is important to see that the design environment is not the same as the business environment. The design environment exists at every layer of our hierarchy.

The subject of our design decisions is bound by our *charter* to change part of the world. We may have a charter to change business processes but leave application software unchanged, or to change application software but leave business processes unchanged, to change the software platform and leave everything else unchanged, etc. Figure 2.8 gives simplified picture, for a charter for change may contain entities scattered over many layers and the design environment may be scattered over the very same layers.

2.3. DESIGN CHARTER AND ENVIRONMENT

Bibliography

- [1] L. Bass, P. Clements, R. Kazman, and K. Bass. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, 1998.
- [2] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.
- [3] J. C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organisations. *IBM Systems Journal*, 32(1):4–16, 1993.
- [4] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Object Technology Series. Addison-Wesley, 2000.
- [5] W. van der Sanden and B. Sturm. *Informatie-architectuur. De infrastructurele benadering*. Panfox Holding, 1997. In Dutch.
- [6] R. Wagter, M. Berg, J. Luijpers, and M. Steenbergen. *DYA: Snelheid en samenhang in business- en ICT-architectuur*. Tutein Nolthenius, Den Bosch, The Netherlands, 2001. In Dutch.
- [7] K. C. Wallnau, S. A. Hissam, and R. C. Seacord. *Building Systems from Commercial Components*. SEI Series in Software Engineering. Addison-Wesley, 2002.
- [8] R. J. Wieringa. *Requirements engineering: frameworks for understanding*. John Wiley & Sons, 1995.
- [9] R. J. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, 1998.

Appendix A

Summary of Models

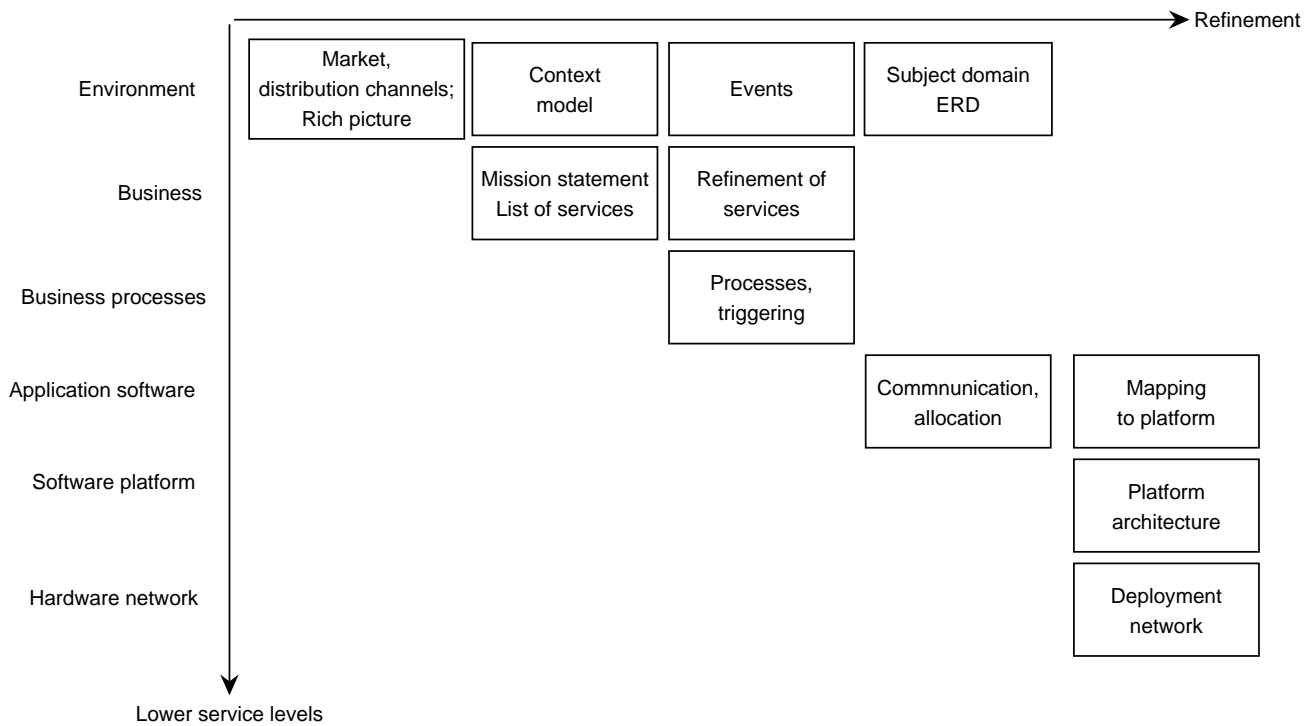


Figure A.1: Overview of models.