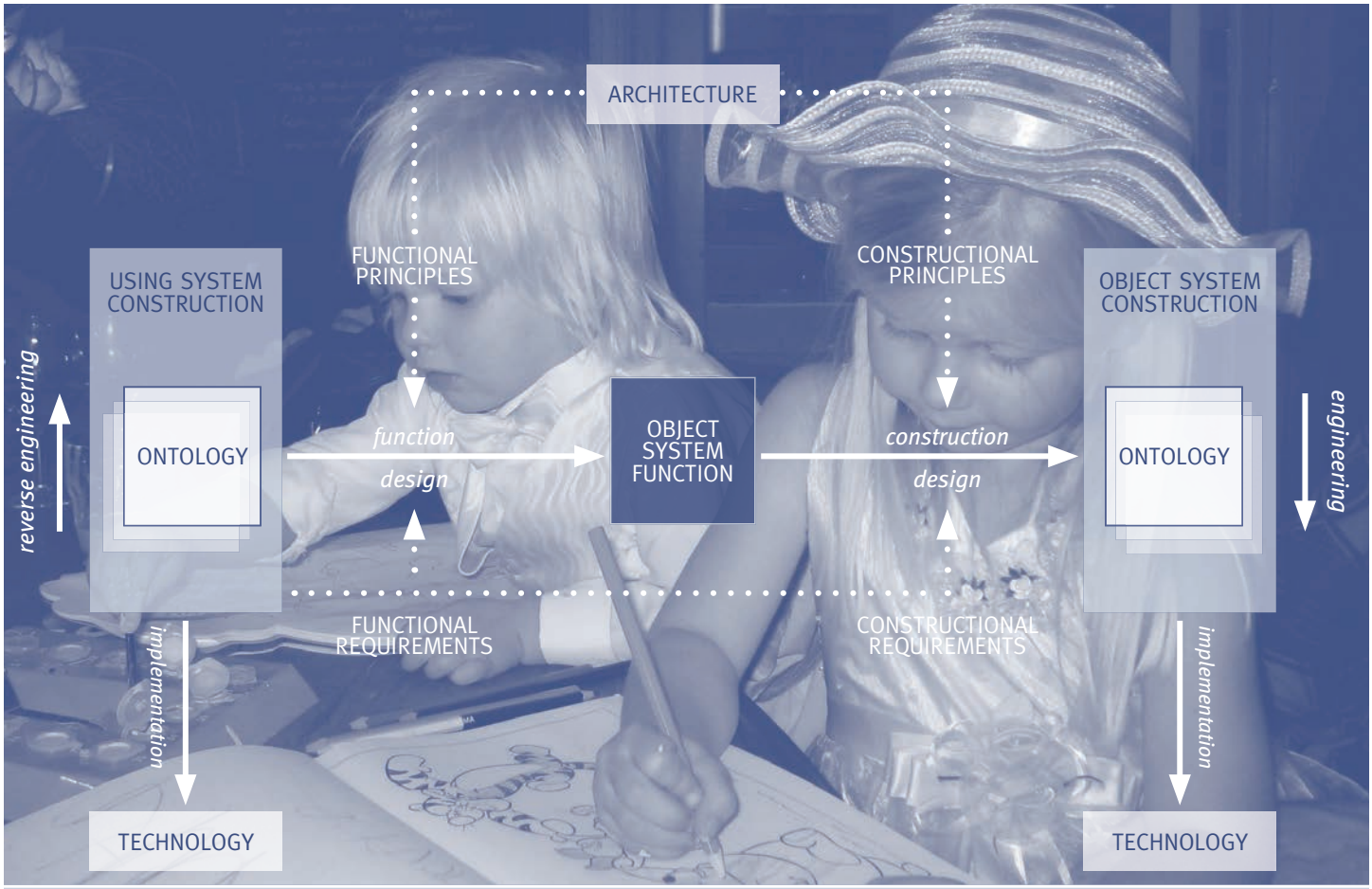


Architecture

Building strategy into design

Jan L.G. Dietz

ACADEMIC SERVICE

Naf
Nederlands Architectuur Forum
voor de digitale wereld

Architecture

Architecture

Building strategy into design

Jan L.G. Dietz

ACADEMIC SERVICE

For more information about this edition and other Sdu editions, please contact:
Sdu Customer Service
P.O.Box 20014
2500 EA The Hague
The Netherlands
tel.: +31 (0) 70 378 98 80
www.sdu.nl/service

© 2008 Jan L.G. Dietz
Academic Service is an imprint of Sdu Uitgevers bv

Lay-out: Redactiebureau Ron Heijer, Markelo
Cover Design: ZEDline, Amsterdam
Print: Drukkerij Wilco, Amersfoort

ISBN: 978 90 12 58086 1
NUR 982

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the publisher's prior consent.

While every effort has been made to ensure the reliability of the information presented in this publication, Sdu Uitgevers neither guarantees the accuracy of the data contained herein nor accepts responsibility for errors or omissions or their consequences.

Contents

Prologue

This book is the final report of a working group of the NAF (Netherlands Architecture Forum). The original name of the working group was GAF (Generic Architecture Framework) but it was very soon named after a crucial idea that we started to develop: the Extensible Architecture Framework (xAF). The objective of the working group, as stated in the foundational proposal, was to develop a framework for information architecture that should be generic in the following aspects:

- The notions in the framework constitute a complete set, i.e., they are necessary and sufficient for the successful execution of architectural projects.
- The framework is broadly applicable in the whole area of business strategy, business processes, ICT, organization, etc.
- The definition of every notion allows for further specialization in any sub area.

The xAF idea was that there is a root architecture framework, on the basis of which other frameworks can be defined as extensions of this root framework, while being extensible themselves in a similar way. In doing so, one would build up a lattice of Extensible Architecture Frameworks (xAFs). Theoretically, it is possible in this way to reformulate any existing architecture framework as a particular xAF in this lattice. The idea of the xAF has been presented and discussed successfully at the LAC (the Dutch national architecture congress) in 2003. Many new members joined the working group since then.

Fairly soon it became clear that a useful architecture framework could only be developed if two notions would be conceived in an appropriate way and would be defined precisely. These notions are "architecture" and "architecture framework". At the same time, we had to conclude that both notions were hardly defined at all and used in very diverse ways. I do not exaggerate when I say that the working group suffered from a veritable tower of Babel. Despite the definitions of architecture by respectable institutions, like IEEE and The Open Group, we were convinced that architecture is not or should not be what these institutions proposed, namely a kind of global design or blueprint. This would not be very innovative. Gradually the members of the working group agreed that architecture is not the global design or blueprint or fundamental organization of a system.

A similar story can be told regarding the notion of architecture framework. Whereas we only needed to conceive a framework as a basic conceptual structure for ordering the needed architectural notions, we observed that the term "architecture framework" mostly referred to former system development methodologies. As it was the case for the notion of architecture, the members of the working group gradually agreed that architectural frameworks have nothing to do with its current meaning, i.e., with system development methodologies.

Therefore, within a year we felt the urgent need for fundamental research into the notions of architecture and architecture framework. Obviously, performing this kind of research is not a task for a NAF working group. Still we wanted to achieve the, apparently ambitious, objectives as stated in the foundational proposal. The way of working that we gradually adopted was that I conducted the needed research at the TU Delft as part of the CIAO![1] program and that the group took the role of sounding board for the research results. As a more or less logical result of this way of working for several years, I became the author of this book instead of its editor.

All persons listed as members of the working group on one of the previous pages have been active for some time between 2003 and 2007. As one may expect from a group of 25 people (in total), over such a long period, and with such challenging objectives, there is a large variance in the size and the impact of the contributions by the members. Nevertheless, I would like to express my gratitude to all of them for what they have contributed because I know this was done from an equal base of enthusiasm and devotion.

As one may expect by now, this book is neither a report on well-established ideas, nor one on well-accepted practices for devising and applying architectures. However, we are all convinced that it contains valuable and precious food for thought and further exploration. Moreover, there seems to be no other way to master the complexity we are faced with.

Delft, September 2008
Jan L.G. Dietz
Chairman

[1] CIAO stands for Cooperation & Interoperability – Architecture & Ontology. For more information, the reader is referred to the www.ciao.tudelft.nl website

Final report of the NAF working group
Extensible Architecture Framework (xAF)

Frank Baldinger
Jeroen van Beele
Frits Broekema
Wiel Bruls
André van Dalen
Jan Dietz
Henk Gingnagel
Hans Goedvolk
Guido van der Harst
Bart-Jan Hommes
Jan Hoogervorst
Michel Janssen
Olaf Kruidhof
Henk Lof
Paul Mallens
Wouter Mellink
Karin Middeljans
Marleen Olde Hartmann
Martin Op 't Land
Erik Proper
Roelof Rabbers
Louis Stevens
Leon van der Torre
Wim Verbeek
Rob Vreke

Executive Summary

The prosperity of modern society is largely determined by the societal performance of enterprises, of all kinds, including commercial, non-profit, and governmental companies and institutions, as well as all kinds of alliances between them, such as virtual enterprises and supply chains. This societal performance depends on two key factors. One is the strategy chosen by the enterprise, as well as the continuous adaptation of this strategy to upcoming threats and challenges. The other one is the implementation of this strategy in a comprehensive, coherent, consistent, and efficient way.

Unfortunately, the vast majority of strategic initiatives fail, meaning that enterprises are unable to gain success from their strategy. High failure rates are reported from various domains: total quality management, business process reengineering, six sigma, lean production, e-business, customer relationship management, as well as from mergers and acquisitions. Whereas often unforeseen or uncontrollable events are presented as the causes of failure, research has shown that strategic failure is mostly the avoidable result of inadequate strategy implementation. It is rarely the inevitable consequence of a poor strategy.

The key reason for strategic failures is the lack of coherence and consistency among the various components of an enterprise. At the same time, the need to operate as an integrated whole is becoming increasingly important. Globalization, the removal of trade barriers, deregulation, etc., have led to networks of cooperating enterprises on a large scale, enabled by the virtually unlimited possibilities of modern information and communication technology. Future enterprises will therefore have to operate in an ever more dynamic and global environment. They need to be more agile, more adaptive, and more transparent. In addition, they will be held accountable more publicly for every effect they produce.

Said problems are often addressed with black box thinking based knowledge, i.e., knowledge concerning the function and the behavior of enterprises. Such knowledge is sufficient and perfectly adequate for managing an enterprise within the current range of control. However, it is fundamentally inadequate for changing an enterprise, which is necessary to meet performance goals that are outside the current range of control. In order to bring about those changes in a systematic and controlled way, white-box based knowledge is needed, i.e., knowledge concerning the construction and the operation of enterprises. Developing and applying such knowledge requires no less than a paradigm shift in our think-

ing about enterprises, since the traditional organizational sciences are dominantly oriented towards organizational behavior, based on black-box thinking.

Fortunately, a new discipline, Enterprise Engineering, is emerging. The mission of Enterprise Engineering is to combine (relevant parts from) the traditional organizational sciences and the information systems sciences, and to develop emerging theories and associated methodologies for the analysis, design, engineering, and implementation of future enterprises. Two fundamental notions have already proven to be indispensable for accomplishing this mission: Enterprise Ontology and Enterprise Architecture.

Enterprise Ontology is defined as the understanding of an enterprise's construction and operation in a fully implementation-independent way. The ontological model of an enterprise offers a reduction of complexity of well over 90%(!) compared to current ways of describing business processes, organizational structures, etc. It is only by applying this notion of Enterprise Ontology, and in particular the DEMO methodology [17, 43], that substantial strategic changes of enterprises can be made intellectually manageable.

Enterprise Architecture is defined as the normative restriction of design freedom. Practically, it is a coherent and consistent set of principles that guide the design, engineering, and implementation of an enterprise. Any strategic initiative of an enterprise can only be made operational through transforming it into principles that guide the design, engineering, and implementation of the 'new' enterprise. Only by applying this notion of Enterprise Architecture, consistency between the high-level policies (mission, strategies) and the operational business rules [31, 19] can be achieved.

Both fundamental notions are extensively discussed and elaborated in this book. Contrary to Enterprise Ontology, the practical application of Enterprise Architecture is still in its infancy. One can imagine that it is quite a job to translate strategic statements in operational design principles and business rules. It is one thing to say that "our enterprise strives to be the best of its sort" but it is quite something else to have all operations consistent with it. However, what is sure by now and by itself already of great value, is that one can only intellectually manage this enormous task by applying the notion of architecture as presented in this book. There is no other way. At the same time, there is also no escape: future enterprises will be required to be designed 'under architecture'. It is not unlikely that this requirement will once become a quality standard.

1 Introduction

Somewhere in the eighties, the terms "architect" and "architecture" became 'hot' in the practice of building, implementing, and using ICT[1] applications in enterprises[2]. The proliferation of both terms went quickly. By the beginning of the nineties, designers of e.g., information systems, infrastructural networks, and business processes called themselves architects. Likewise, the global design of a system was commonly called its architecture. To worsen the case, many system development methodologies were renamed "architecture framework", mostly after having been restructured more or less along the lines of Zachman's framework [54]. Another evidence of the 'hotness' of architecture is the establishment of the NAF[3] in 2002. Its articles of association include that the goal of the NAF is to propagate 'working under architecture', remarkably without clarification of what this would mean. Lastly, in 2008 the LAC[4] celebrates its 10th anniversary.

To illustrate the astonishing proliferation of the use of "architect" and "architecture", imagine someone who vanished 25 years ago from our society and has lived in the bushes of Papua New Guinea since then. How surprised would he or she be when returning in the western world? Most probably, he or she would think that our profession has passed through an enormous development, and that it would be very hard to catch up with it. To reassure him or her right away: don't worry, almost nothing has changed; it is really for the greater part a terminological matter. But how could this happen? Have we really become out of our mind and should we as soon as possible return to the vocabulary of the eighties? Or, have new ideas evolved in the meantime, ideas that may be worthwhile to articulate

---

1 ICT stands for Information and Communication Technology; it is the nowadays quite common successor of IT, including explicitly Communication Technology, like the modern Internet.

2 Throughout the book, the term "enterprise" denotes any organized human undertaking, like a company, a governmental agency, and a non-profit institution. An enterprise may be rather permanently, like a railway company, or temporarily, like the building of a railway section.

3 NAF stands for Nederlands Architectuur Forum; it is a Dutch association of ICT service providers, of consumers of these services, and of research and educational institutes, that seeks to propagate the notion of architecture, and to improve the professionalism of the architect.

4 LAC stands for Landelijk Architectuur Congres; it has become the largest Dutch congress on applying ICT in organizations.

and apply? Fortunately, a couple of new ideas have emerged. Now that the architectural dust cloud of the past decade comes down, some ideas are revealed that seem to provide effective answers to the key challenges that 'architects' currently face. Before discussing them, however, let us briefly look at the prevalent current understanding of the notion of architecture. As an example, we take the definition known as IEEE 1471 [39]. It reads as follows:

> Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

Let us start by making a critical note: this definition will not win the beauty award, for two reasons. One reason is that it is ambiguous. It states that architecture is 'the fundamental organization of a system' and, at the same time, it states that architecture consists of 'the principles guiding its design and evolution'. Therefore, what is it going to be; do we have to toss a coin? The other reason is that the formulation, as cited above, scarcely deserves the name definition; it is merely a description. In order to call something a definition, it must be embedded in a theoretical framework, and it must be formulated precisely, preferably in a formalized language. Having made this critical note, let us try to identify the relevant issues that this definition, like most other ones that go around, seems to address. There are two of them, apparently.

One issue is 'the fundamental organization of a system'. Indeed, there is a great need to grasp what is fundamental or essential about the organization or structure of a system[5]. The systems 'architects' deal with have become so complex that we need effective help in our attempts to understand them. Alternatively, as Dijkstra already stated at the dawning of the era of software engineering [21]:

> Because we struggle with the small sizes of our heads as long as we exist, we need intellectual techniques that help us in mastering the complexity we are faced with.

He considered in particular the next intellectual techniques indispensable: *separation* of concerns, effective use of *abstraction*, and devising appropriate *concepts*. Dijkstra successfully separated the concern for correctness from the concern for efficiency in programming and thereby made programming intellectually manageable.

---

[5] Although the definition by IEEE was intended for the field software engineering, we take the notion of system broader, considering the definition to hold for every system kind.

In order to accommodate this apparent need for understanding the organization of complex systems, we will introduce the notion of *system ontology*. The first part of this term comes from the Greek word "οντοσ", which means being and essence. As will be shown, applying the appropriate notion of system ontology for a certain kind of system, leads to an enormous reduction of complexity.

The second part of IEEE 1471 reads: 'the principles guiding its design and evolution'. It expresses the need for guidance in the process of designing a system; the design freedom is apparently too large. Indeed, this is generally the case, regardless the kind of system, so whether one thinks of a car or a house or an enterprise. For software systems in particular, the design freedom is immensely large. Because the need for understanding the organization of a complex system is going to be addressed by the notion of system ontology, we will reserve the word "architecture" exclusively for addressing the problem of restricting design freedom. Therefore, this will be our concept of *architecture*. Operationally, architecture is a set of design principles that together achieve the restriction of design freedom that is considered necessary. In addition, we will show the benefits of deriving these principles from the strategic statements that are in force in an enterprise, such that they become consistent with the enterprise's strategy.

In our view, one of the key challenges 'architects' currently face is that the traditional organizational sciences increasingly fall short in helping enterprises to implement strategies effectively, and in a controlled way. Over 70% of the strategic initiatives appear to fail, meaning that enterprises are unable to derive success from their strategy [41], [34]. These high failure rates are reported from various domains: total quality management [42], business process reengineering [11, 48], six sigma [23], e-business [34], customer relationship management [36], and mergers and acquisitions [52]. Whereas all too often, unforeseen or uncontrollable events are presented, for convenience sake, as the causes of failure, research has shown that strategic failure is mostly the avoidable result of inadequate strategy implementation. Rarely is it the inevitable consequence of a poor strategy [34]. A plethora of literature indicates that the key reason for strategic failures is the lack of coherence and consistency, collectively also called congruence, among the various components of an enterprise [4, 41, 29, 35, 37, 30, 24, 42]. At the same time, the need to operate as an integrated whole, is becoming increasingly important. Globalization, the removal of trade barriers, deregulation, etc., have led to networks of cooperating enterprises on a large scale, enabled by the enormous potency of modern ICT. Future enterprises will therefore have to operate in an even more dynamic and global environment than the current ones. They need to be more agile, more adaptive, and more transparent. Moreover, they will be held more publicly accountable for every effect they produce.

Said problems are traditionally addressed with *black box* thinking based knowledge, i.e., knowledge concerning the function and the behavior of enterprises. Such knowledge is sufficient for managing an enterprise within the current range of control. However, it is totally inadequate for meeting performance goals that are outside that range, thus for changing an enterprise. In order to bring about changes in a systematic and controlled way, *white-box* based knowledge is needed, i.e., knowledge concerning the construction and the operation

of enterprises. Developing and applying such knowledge requires no less than a *paradigm shift*, since the traditional organizational sciences are not able to ensure that enterprises are coherently and consistently integrated wholes. The needed new point of view is that enterprises are purposefully designed, engineered, and implemented systems. The needed new skill is to (re) design, (re) engineer, and (re) implement an enterprise in a comprehensive, coherent and consistent way (such that it operates as an integrated whole), and to be able to do this whenever it is needed.

The current situation in the organizational sciences resembles very much the one that existed in the information system sciences around 1970. At that time, a revolution took place in the way people conceived information technology and its applications [18, 38]. Since then, people are aware of the distinction between the *form* and the *content* of information. This revolution marks the transition from the era of data systems engineering to the era of information systems engineering. The comparison we draw with the information sciences is not an arbitrary one. On the one hand, the key enabling technology for shaping future enterprises is the modern ICT. On the other hand, there is a growing insight within the information system sciences that the central notion for understanding profoundly the relationship between organization and ICT is the entering into and complying with commitments between social individuals [28, 51, 17]. These commitments are raised in communication, through the so-called intention of communicative acts. Examples of intentions are requesting, promising, stating, and accepting. Therefore, just like the content of communication was put on top of its form in the 1970's, the *intention* of communication is now put on top of its content. It explains and clarifies the organizational notions of collaboration and cooperation, as well as notions like authority and responsibility. This current revolution in the information systems sciences marks the transition from the era of information systems engineering to the era of enterprise engineering. At the same time, it enables information systems engineering to converge with the traditional organizational sciences, as illustrated in Figure 1.1.

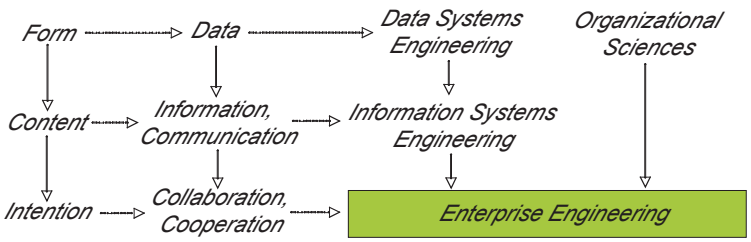


Figure 1.1 The roots of Enterprise Engineering

As said before, the basic premise of enterprise engineering is that an enterprise is a designed system. In order to ensure that the designing of a system is performed coherently and consistently, such that the resulting system is a truly integrated whole, the notions of ontology and architecture are crucial. The *ontology* of a system is theoretically defined as the understanding of its construction and operation in a fully implementation independent

way. Practically, it is the highest-level constructional model of a system, the implementation model being the lowest one. Compared to its implementation model, the ontological model of an enterprise offers a reduction of complexity of well over 90% [18]. *Architecture* is theoretically defined as the normative restriction of design freedom. Practically, it is a coherent and consistent set of principles that guide the design of a system. Any strategic initiative of an enterprise can only be made operational through applying the notion of architecture, namely, by expressing it in principles that guide the designing of the 'new' enterprise.

According to one of the founding fathers of the General System Theory, a core problem facing modern science is developing a theory about 'organized complexity' [7]. Based on the level of complexity, Weinberg identifies three areas [50], as shown in Figure 1.2. The first area regards (relative) low complexity, identified by Weinberg as "organized simplicity", such as exemplified by machines en mechanisms. This type of complexity can be addressed through analytical methods. At the other end of the spectrum lies the area "unorganized complexity". Here the number of elements and the variety are so large that complexity can be addressed through statistical methods (such as with gas molecules in a closed space, or with certain aspects of traffic).

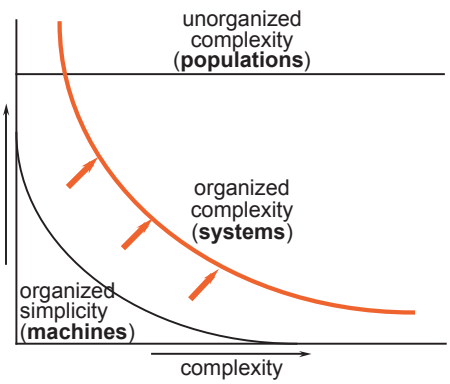


Figure 1.2 Weinberg's definition of organized complexity

The large area between these extremes is that of "organized complexity": too complex for analytical methods and too organized for statistical methods. According to Weinberg, this area – in which we position enterprises – is preeminently suitable for, and in definite need of, the system approach. The system approach offers a formal methodology to address the enterprise as a whole, while recognizing its constituent parts and their mutual relationships, in order to safeguard a unified and integrated system. Many authors endorse the point of view that the system approach is the only meaningful way to address aforementioned core problem of modern science [7, 10, 25, 44]. Therefore, Ackoff argues that the failing strategic initiatives as mentioned earlier are due to the fact that the initiatives are fundamentally "anti-systemic" [1]. In our conviction, adopting the enterprise engineering paradigm makes it possible to extend the area of organized simplicity such that it includes a part of the area of organized complexity, thus of enterprises (cf. Figure 1.2).

As indicated, unity and integration is crucial for the success of the enterprise as a *whole*. There will be little debate about the conviction that unity and integration is not brought about spontaneously but has to be *designed* intentionally. This begs the question of *how* the enterprise, in view of its goals and the required unity and integration, must be designed. Finding an answer to this question is the central subject of this book. For now, we like to emphasize that any answer is essentially *normative*. We agree with Jackson arguing that the normative aspect of system design must be made explicit [33]. The notion of architecture satisfies this need.

The remainder of the book is organized as follows. In Chapter 2 the theoretical foundations are laid that are considered indispensable for thoroughly understanding the following chapters. It centers on the notions of system and model. The notion of ontology, in particular enterprise ontology, is introduced and discussed in Chapter 3. As will be shown, this notion offers the basis for understanding the construction and the operation of organizations in a coherent, consistent, complete, and yet concise way. In Chapter 4, a conceptual framework is introduced and discussed which appears to be of great help in understanding the interplay between ontology, architecture, and technology in the development of systems of any kind. The framework is named Generic System Development Process (GSDP). The main subject of the book is contained in Chapters 5 and 6. In Chapter 5 the notion of architecture is discussed, within the conceptual framework of the GSDP. Moreover, attention is paid to the activity of architecturing, i.e., devising architectures. Chapter 6 is about architecture frameworks. The Extensible Architecture Framework (xAF) is introduced and discussed. It will turn out to be a generic framework for classifying and comparing architecture frameworks. As an illustration, the Integrated Architecture Framework (IAF[6]) will be placed in the xAF. In order to demonstrate the feasibility of the major findings of the book, these are applied to a case in Chapter 7. Lastly, the conclusions that can be drawn from the preceding chapters are formulated in Chapter 8.

---

[6] The IAF is a well-known architecture framework, which has originated within Capgemini.

2 System and Model

The notions of system and model are fundamental notions in systems development. At the same time, their definitions are often not very precise, leading to confusion and suboptimal development. As we want to be very precise about the core ideas of ontology and architecture (because we need to be), we will start with a thorough introduction and discussion of the notions of system and model, and of their relationships.

2.1 Introductory considerations

When studying the way in which the notion of system is used in the distinct scientific disciplines, it appears that two quite different system notions exist, which we will refer to as the teleological and the ontological ones. The teleological system notion is about the function and the behavior of a system, whereas the ontological one is about its construction and operation (the Greek word "τελεοσ" means purpose or objective, and the Greek word "οντοσ" means being and essence). Unfortunately, they are often mixed up, which gives rise to a lot of confusion in current literature. For example, Wikipedia, which usually contains pretty good starting material for a discussion, provides the next definition of system (slightly paraphrased):

> A system is a set of interacting or interdependent entities, real or abstract, forming an integrated whole. There are natural and man-made (designed) systems. Man-made systems normally have a certain purpose. They are designed to work as a coherent entity. Natural systems may not have an apparent purpose.

The mix of ontological and teleological thinking is immediately apparent. The first sentence is ontological; the other ones are teleological. Next, it seems from the definition that designed systems may have a purpose but may lack one also, whereas natural systems may lack a purpose but also may have one. Apparently, having or lacking purpose is not a

useful criterion to distinguish sharply between classes of systems. As we will see later, this problem is clarified if one considers purpose to be a relationship between a system and its user instead of an inherent system property.
Wikipedia provides some additional thoughts on the notion of system:

> A system is a fundamental concept of systems theory, a way of thinking about the world, a model. We determine a system by choosing the relevant interactions we want to consider, plus choosing the system boundary or, equivalently, providing membership criteria to determine which entities are part of the system, and which entities are outside of the system and are therefore part of the environment of the system.

Although this piece of text might appear clear and harmless at first sight, it is confusing and harmful too. Firstly, it seems as if one could consider anything a system. If that were true, it would make little sense to apply the notion at all, since one could then consider something a system that does not conform to the definition of a system, regardless of the particular definition one adheres to. Secondly, equating a system with a model also does not seem to be fruitful. At least it raises the question what the difference is between system and model (if there is one at all). All of these issues will get ample attention in the remainder of this chapter.
Almost as important as the notion of a system is the notion of a *model* of a system. Investigating systems mostly comes down to building models and analyzing the behavior of these models. Mathematics and logic provide the means to build complex but exact conceptual models and to analyze the behavior of these models thoroughly. The only weak point is the validation of the model against the modeled system. Is the model appropriate? Is it sufficiently accurate? What are the ranges of reliability? Can one predict the behavior of the system on the basis of the behavior of its model?

2.2 The Ontological System Notion

As we have seen, the ontological system notion is concerned with the construction and the operation of a system. It is adequate and necessary for the purpose of building and changing systems. Therefore, it is the dominant system notion in all engineering sciences. To these sciences we count e.g., information systems engineering and enterprise engineering, including organizational engineering, business process engineering etc.
In our opinion, the most crisp and precise definition of the ontological system notion is the one provided by Bunge [10]. He starts by distinguishing between aggregates and systems. Both are collections of items. In an *aggregate*, the items are not held together by interaction bonds, in a *system* they are. A system therefore has unity and integrity, whereas

an aggregate lacks these properties. An example of a concrete aggregate is a random sample of a biological population. An example of an abstract aggregate is the mathematical set. The basic and indispensable property of a system is that the elements influence each other. The existence of just a relationship between the elements is insufficient. Thus, according to this definition, neither the well-known Periodic System of Elements in chemistry, nor groups of people that are related by their age difference are systems for example. Both are aggregates. Examples of concrete systems are human bodies, the sun, and computers. A system has a composition, an environment, and a structure, which consists of the interaction relationships among the elements in the composition and the environment [10]. We add to this definition that a system has a production. By this, we mean what is brought about by the elements of the system through their interaction. Put in functional terms (so applying the teleological system notion), one would say that the production of a system consists of providing services to its environment.

Usually, the effect of the interaction between the elements of a system is conceived as a change of the system's state. The notion of system state appears to be ambiguous, however, since changes in the composition or structure of a system may also be considered as state changes. We therefore introduce the notion of world and we distinguish between the coordination world and the production world of a system. A world is at every moment in a particular state, where a state is simply defined as the set of facts that hold (or that are current) at that moment. The state of the *coordination world* reflects the progress of the interactions among the elements by means of their structural relationships. The state of the *production world* reflects the effects of the production acts that are performed by the elements of the system.

In order to elaborate the ontological system notion, we take enterprises as an example, while following the lines of thought in [10]. The composition and the environment of an enterprise consist of socially interacting human beings. The brains of these individuals are parts of them but do not qualify as members or components of a social system because they do not enter independently into social relations: only complete human beings can maintain social relations. Likewise, brains are a biological system but the molecules in the neural cells do not qualify as components of a biological system; they are physical systems. In other words, we need to introduce the notion of proper element of a system, and, corresponding to it, clarify the notion of system category. The *proper elements* of a system are those parts of it that can be engaged independently in mutually influencing relations. The type of these relations determines the *category* to which the system belongs. The elements of a system are atomic with respect to the system's category. Thus, the *composition* of a social system is a set of social individuals (human beings[1]), whereas the composition of a biological system is a set of cells and the composition of a physical system is a set of physical particles. Next, the *environment* of a system is the set of proper elements from the same

[1] Throughout the book, we consider human beings to be the only possible social individuals. Therefore, we disregard primates and other highly developed mammals, as well as all artificial intelligent agents.

category that are not contained in its composition but that act on or are acted upon by elements of the composition. Third, the *structure* of a system is the set of mutually influencing relations, as determined by the system category, among the system's elements as well as between them and the elements in the environment. Finally, by the *production* of a system we understand what is brought about by the elements in the composition and transferred to the elements in the environment through interaction links.

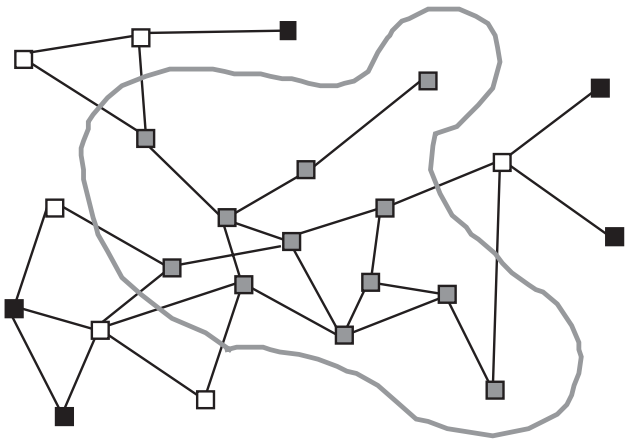


Figure 2.1 The construction of a system

The composition, the environment, and the structure, are collectively called the *construction* of a system. The construction of a system can thus be described by enumerating the elements in the composition and the environment, as well as the relationships in the structure. Figure 2.1 illustrates the construction of a system. The composition consists of the gray-colored elements; the environment consists of the white-colored elements. The gray line that separates the composition and the environment is called the *system border*. The lines connecting the elements represent the structural bonds between them. Only the bonds among the internal elements and the bonds between these elements and elements in the environment belong to the structure of the system. Therefore, the black elements are external; they do not belong to the system because they do not have influencing bonds with elements in the composition. The composition, together with the structural bonds between its elements, is also called the *kernel* of the system.

Clearly, this definition of system defines a type of thing (like the type car or the type tree), which means that concrete things may conform to this type or not. Anything that conforms to the defined type may rightly be called a system; anything that does not conform to the type is not a system. To elaborate this point, we consider it misleading to say that one can view something as a system. That would be like saying that one views a bird as a mammal: it makes little sense, since a bird is not a mammal. Another aspect we like to emphasize is the inclusion in our system definition of the environment. It may sound counterintuitive to do this, since the operation of the system then becomes dependent on this environment. However, on closer sight, this is fully correct; one even cannot do otherwise because there is

interaction between (the elements in) the environment and (the elements in) the composition. To top that, the structural influence bonds between the elements in the composition and the elements in the environment through which these interactions take place, inherently co-determine the operation of the system. Put differently, if the kernel of a system is placed in another environment, the whole (system) behaves differently.

The definition provided above applies to *homogeneous* systems, i.e., systems that belong to exactly one category. However, the most interesting systems are not such simple homogeneous systems but complex combinations of homogeneous systems, called *heterogeneous* systems. For example, a human being is a heterogeneous system, composed of at least a physical system, a chemical system, and a biological system. These systems are related to each other in a layered nesting [10]. The chemical system builds on the physical system and the biological system builds on the chemical system. Likewise, a car is a heterogeneous system, composed of at least a mechanical system, an electrical system, and a chemical system. The way in which a collection of homogeneous systems constitutes a heterogeneous system is not trivial and not simple. This holds, in particular, for organizations. It is obvious that an organization is a heterogeneous system, consisting of several homogeneous systems. But how exactly? What kinds of partial systems should we distinguish? We will address these questions in Chapter 3.

Another notion that is often ill defined is the notion of subsystem. Intuitively, a subsystem encompasses only a subset of the composition of a system but that is neither a complete nor a precise definition. One also has to tell how the environments and the structures of the two systems are related. On the basis of the definition of a system's construction above, we are able to provide the next precise definition of the notion of subsystem (in which C stand for composition, E for environment, and S for structure):

Let there be a system σ_1 with the construction $< C(\sigma_1), E(\sigma_1), S(\sigma_1) >$ and a system σ_2 with the construction $< C(\sigma_2), E(\sigma_2), S(\sigma_2) >$. Then system σ_2 is a *subsystem* of system σ_1 if and only if

$C(\sigma_2) \subseteq C(\sigma_1)$
$E(\sigma_2) \subseteq (C(\sigma_1) \setminus C(\sigma_2) \cup E(\sigma_1))$
$S(\sigma_2) \subseteq S(\sigma_1)$

The collective activity of the elements in the composition and the environment is called the *operation* of the system. Thus, one could also say that the operation of a system is the manifestation of its construction in the course of time, once the system is 'set to operate'. As we have discussed earlier, the elements perform two kinds of acts: production acts and coordination acts. Their effects are changes in the state of the production world and the coordination world respectively. In the next section, we will discuss in a more precise way what the state of a world is, and what state changes are.

2.3 World Ontology

In this section we introduce the notion of world ontology as opposed to (but also related to) the notion of system ontology, which is the subject of the next chapter. The notion of world we adopt is the discrete event world, so we assume a discrete linear time dimension. It means that the time difference between any two consecutive points in time is the same. This time difference is called the basic time unit. The choice of the basic time unit is not prescribed; it may be freely chosen, in accordance with the nature of the corresponding system. For example, to understand appropriately the operation of a modern computer system, the basic time unit should probably be a nanosecond but for many manufacturing systems an hour or even a day may be appropriate. Note that in the latter case we would call an hour or a day a point in time. Whatever happens during a particular hour or a particular day happens *in* that hour or *on* that day. Given this choice of the basic time unit, it is not possible to be more precise, and thus to distinguish between the occurrence times of two events that occur in the same basic time unit.

A *world* can formally be defined as a pair < C, B > with:
C: a set of objects, called the *composition*.
B: a set of facts, called the *state base*.

The notion of *object* corresponds to the notion of bare individual in [9]. It means that we assume the 'existence' of an object even if we cannot know anything about it. The number of objects in C is denumerable infinite; so, there will never be a shortage of objects: all things that are interesting or that may become interesting are already there.
A *fact* is a particular arrangement of one or more objects. Depending on the number of objects that are involved in a fact, we speak of unary, binary, ternary, etc., facts. An example of a unary fact is that Beatrix[2] is the Queen of the Netherlands. Another example of a unary fact is that Beatrix is a human being. Both facts regard the same bare individual. An example of a binary fact is that Willem Alexander is a son of Beatrix. A fact is said to be *current* at a particular point in time if it is the case at that point in time.
At any moment, a world is in some state. The *state S* at time *t* is defined as the set of facts that are current on *t*; $S \subset B$. A state change is called a *transition*. A transition is defined as an ordered pair of states, e.g., $T_1 = <S_1, S_2>$ is the transition from the state S_1 to the state S_2. The occurrence of a transition is called an *event*. An event therefore can be defined as a pair $< T_1, t >$, where T_1 is a transition and *t* is a point in time. Consequently, a transition can take place repeatedly during the lifetime of a world; events however are unique: they occur only once. Often, one talks about *event type* instead of transition.

[2] Throughout the book we assume every name (or other sign) that is used to designate an individual concept denotes the corresponding object uniquely. Ontologically, names are irrelevant; we only consider objects and we assume that they are identifiable.

In order to understand profoundly what a state of a world is, and what a state transition is, we will distinguish between two kinds of facts: dependent and independent facts. An *independent* fact is something that is the case and that will always be the case. For good reasons one can even say that it has always been the case. In other words, it is an inherent property of an object or an inherent relationship between objects. A *dependent* fact is something that becomes the case as the effect of an act in the corresponding system.
Examples of dependent facts in the context of a library [17] are the ones expressed in the next *assertive* sentences (in which the variables, represented by capital letters, are placeholders for object instances):

> "the author of book title T is A"
> "the membership of loan L is M"

On the one hand, the existence of these facts depends on the existence of the corresponding book title and loan respectively. On the other hand, dependent facts are timeless. For example, a particular book title has a particular author (or several authors). If this is the case at some point in time, it will forever be the case. One might even say that it has always been the case, that it was only not knowable before some point in time (namely before the book was written). We will take this position, because it makes life a lot easier. A similar remark holds for every *defined* fact. A defined fact is determined by its definition. The specification of this definition is the only necessary and sufficient condition for the existence of the defined fact. This marks an important difference between a world and an information system about that world. For example, the age of a person (which is a defined fact) ontologically just exists at any moment; in the corresponding information system, it has to be derived, i.e., computed when it is needed.
Dependent facts are subject to *existence laws*. These laws require or prohibit the coexistence of facts (in the same state of a world). For example, if the (single) author of some book is "Ludwig Wittgenstein", it cannot also be "John Irving".
Contrary to a dependent fact, an independent fact is the result or the effect of an act. Examples of independent facts in the context of a library are the ones expressed in the next *perfective* sentences (the variables are again placeholders for object instances):

> "book title T has been published"
> "loan L has been started"

The becoming existent of an independent fact is an *event*. Before the occurrence of the event, it did not exist (i.e., it was not the case); after the occurrence it does exist (i.e., it is the case and it will forever be the case). Events are subject to *occurrence laws*. These laws require or prohibit sequences of events in the course of time. For example, some time after

the occurrence of the event "loan L has been started", the event "loan L has been ended" might occur, and, in between, several other events may have occurred, such as "the fine for loan L has been paid". Therefore, events can best be conceived as status changes of a thing. Having presented the distinction between dependent and independent facts, we will from now on not mention this distinction explicitly, assuming that it is clear from the context with what kind we are dealing.
We are now able to provide a precise definition of the ontology, or, more precisely, the ontological model of a world:

> The ontological model of a world consists of the specification of its state space and its process space. Both are expressed in business rules.

By *state space* is understood the set of allowed or lawful states. It is specified by means of the state base and the existence laws. By the *process space* is understood the set of allowed or lawful sequences of events. It is specified by the event base and the occurrence laws. The *event base* is the set of event types of which instances may occur in the world. Every such instance has a time stamp, which is the occurrence time of the event. Existence laws and occurrence laws are expressed in *business rules*. There are two possible shapes of business rules: declarative and imperative [20]. An example of a declarative rule, in a library, is:

> no more than 5 books can be borrowed at the same time under one membership

The same rule can be expressed imperatively in the next way:

> ***when*** a loan is requested under some membership
> ***if*** there are already 5 books in loan under this membership
> ***then*** decline the request
> ***else*** promise the request

Note that both the declarative and the imperative formulation are rather informal; this is done for the sake of simplicity. The choice for one of the two kinds of formulation in practice is highly dependent on the (presumed) competences of the actors who are going to apply them. The notions of request, decline and promise are explained in Chapter 3. Lastly, the imperative shape is the most appropriate candidate for automation.

2.4 The Teleological Notion of System

The *teleological system* notion is concerned with the function and the behavior of a system [10]. It is adequate for the purpose of *using* or *controlling* a system. It is therefore the dominant system notion in the social sciences, including the organizational sciences. As will be revealed in the next section, the teleological system notion is actually identical to one of the model types that are applied to systems: the *black-box model*. Therefore, we confine ourselves to single out the distinction between function and purpose, as this is an apparent issue in practice.

Many people, particularly those who adhere to the teleological system notion, like to call the production of a system its purpose. We want to make a clear distinction, however, between function and purpose. Every designed system has a *function*, which is the intended use of the system by its designer. Through its function, a system is able to support some other system, which is said to use the function. On the other hand, *purpose* is a relationship between a system and a stakeholder. For example, my purpose in using (the function of) my car could be to go to work but I could just as well go to the red-light district of Putten.

As an example of a natural system, consider the circulation of blood in the human body. In explaining it, a teacher may very likely say that it is the purpose of the heart to pump the blood around, such that it can exchange things with other parts of the body, like exchanging oxygen and carbon dioxide with the lungs. We consider such an explanation as a typical anthropomorphic explanation: only because we, human beings, can have purposes, we transpose this idea to other things without even thinking about the appropriateness of doing it. From the teleological point of view it is acceptable to say that it is the function of the heart to pump[3], and that the effect of the pumping is that the blood flows through the veins, and so on. Note however that the very notion of pumping is a functional notion, something we assign to the heart.

From the ontological point of view, such functional notions do not have meaning. Obviously, the heart does not 'know' that it 'pumps'. We just observe that some constructional components (called the heart muscles) contract and release, and that other constructional components (called valves) open and close synchronously to these muscle movements, resulting in a pulsating flow of blood, etc. Based on this knowledge one can easily explain what people in the functional perspective mean by pumping.

This is the basic relationship between the function and the construction of a system: the construction brings about and explains the function. At the same time, one should keep in mind that construction is objective, it does not need human involvement, whereas function is subjective: it is in the eye of the user. A good system design reveals its function immediately and clearly.

[3] Although it is of course questionable that the heart has been designed.

2.5 The Notion of Model

Colloquially, the term "model" is used in many different ways. For example, people say that an information system is a model of a Universe of Discourse or world because it reflects the state and the state changes of that world. However, a prototype of a system to be built is also called a model. These seemingly diverse notions of model are reconciled by the definition given in [3], which we quote verbatim below:

> Any subject using a system A that is neither directly nor indirectly interacting with a system B, to obtain information about the system B, is using A as a model for B.

Therefore, the notion of model is a *role* notion; something is not a model per se, in some absolute sense but it may be used as a model. In general, three gross kinds of systems can be distinguished: concrete systems, symbolic systems, and conceptual systems. Their relationships can conveniently be represented in the model triangle, as shown in Figure 2.2, an adaptation of the one in [8]. There is an invisible horizontal line in the middle of the figure. Things above it are subjective and abstract; they only exist in the human mind. Things below it are objective and concrete. Symbolic systems are a subclass of concrete systems. Their distinctive property is that the elements are signs. For a convenient explanation of the model triangle, let us call an X-type system that is used as a model for some system S an X-type model of S. Thus, for example, a conceptual system that is used as a model of a car is called a conceptual model of the car. We will briefly go through all relationships between the three kinds of systems.

A concrete model of a concrete system is called an *imitation*. Examples: a scale model of an airplane or a ship or any other concrete thing. The reason for building an imitation of a system is generally that it is easier, cheaper, less dangerous, etc., to study the model instead of the system itself.

A conceptual model of a concrete system is called a *conceptualization*. It plays a major role in all sciences. Examples: the geometrical sphere as a model for celestial bodies; the (feedback) control system as a model of biological or technical or managerial processes; the Petri Net as the conceptualization of the processes in a computer. A concrete model of a conceptual system is called an *implementation*. Examples: the pyramids of Giza are an implementation of the geometric concept of pyramid (though, one might argue whether actually the reverse is true); James Watt's steam engine regulator as an implementation of the (feedback) control system; a computer process as an implementation of the Petri Net.

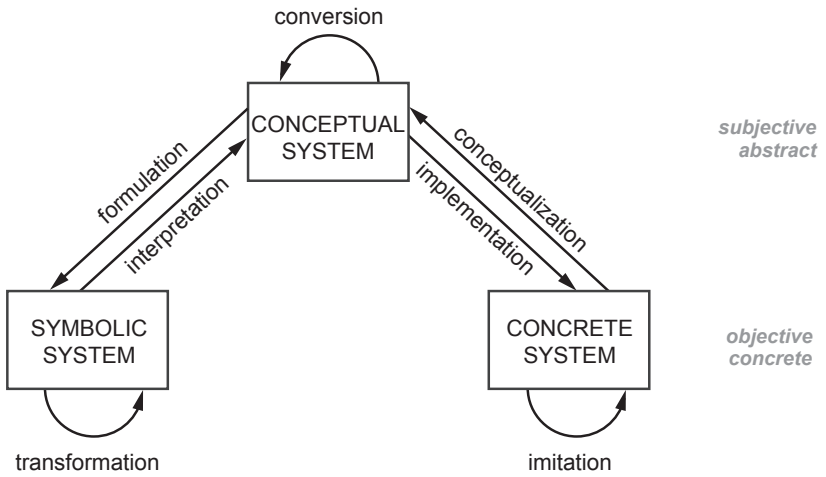


Figure 2.2 The model triangle

A conceptual model of a conceptual system is called a *conversion*. Examples: the algebraic concept of a circle ($x^2 + y^2 = r^2$) is a conversion of its geometric concept (and vice versa); $v = a.t$ is the algebraic conversion of the physical concept of the uniformly accelerated movement.

A symbolic model of a conceptual system is called a *formulation*. A symbolic system is expressed in some formal language. People often become so used to the notations in languages that they equate the expressions with the conceptual models they represent. A splendid example is the algebraic concept of a circle. We referred to it above by the equation $x^2 + y^2 = r^2$, meaning the notion that is expressed by it, and not the notation. As the symbolic model of it, however, we mean the notation. Other examples: the Peano-Russell notation of a logical formula; the Actor Transaction Diagram notation of an Interaction Model (see Chapter 3).

A conceptual model of a symbolic system is called an *interpretation*. Since interpretation is the reverse of formulation, we refer to the examples given above. Other examples: educing the meaning of a data flow diagram; the deciphering of the Rosetta Stone.

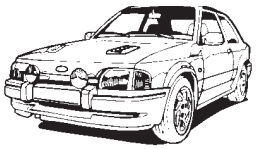
A symbolic model of a symbolic system is called a *transformation*. It is also often called translation. However, one should be cautious in doing that. Translation presupposes understanding of what is being written (or spoken); one goes back and forth interpretation and formalization, so to speak. By transformation we really mean transcoding, e.g., from ASCII to EBCDIC, or from Morse to the Roman notation of letters, without needing to understand the meaning of the coded text.

There are two fundamentally different types of conceptual models: the white-box (WB) and the black-box (BB). They are both discussed hereafter. One should keep in mind that a conceptual model is something in the mind. It is distinct from the symbolic system (sketch, diagram, etc.) in which it may be formulated for the sake of communicating it to somebody else or to oneself at a later point in time.

2.6 The White-Box Model

A *white-box* (WB) model is a direct conceptualization of the ontological system definition. It captures the construction and the operation of a system, while abstracting from implementation details; these are assumed irrelevant. An example of a WB type of model is the model of the physical atom by Niels Bohr. It is a model that explains the inner structure and dynamics of an atom: a kernel of protons and neutrons surrounded by electrons that revolve around the kernel in specific orbits. The *white-box model* is adequate for the purpose of *building* or *changing* a system. It is therefore the dominant type of model in all engineering sciences. Figure 2.3 exhibits the white-box model of a car, which conveys the mechanic's perspective.

A WB model conveys the *construction perspective* on a system, which means that one is interested exclusively in the construction and the operation of the system. Contrary to this, a black-box model conveys the *function perspective* on a system. Now one is exclusively interested in its function and its behavior. The relationship between the two perspectives is that the function and the behavior are brought about, and explained, by the construction and the operation of the system. Function is just a name for referring to the behavior of a system in terms of its using system[4].



the mechanic's perspective

construction:
the components and their interaction relationships

operation:
the manifestation of the construction in the course of time

constructional (de)composition

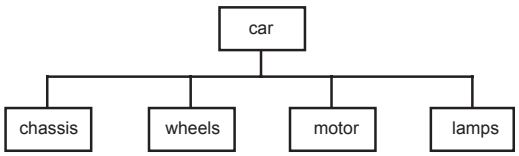


Figure 2.3 White-box model and constructional (de)composition

In order to keep WB models of complex systems manageable, the technique of *constructional (de)composition* is often applied. It is, in fact, the technique to compose a system as a construction of parts (elements or subsystems). There is only one way to do this, which is to be faithful to how the system is (or was, or has to be) constructed. Any other way of composing would not yield the same system (as everyone who has ever tried to repair a mechanical alarm clock or some other device knows). Actually, what one does by identifying

[4] The concept of using system will be discussed in Chapter 4.

higher order components is identifying subsystems and covering, as it were, their kernels, thus only taking into account the interactions between that component and elements or components in its environment. The advantage of a constructional composition is that one can focus on one or a few parts (elements or components) without having to bother about the other parts.

2.7 The Black-Box Model

A *black-box* (BB) model is a conceptual system that has no direct relation with the construction and operation of the concrete system that it models. Basically, in a BB model, only the interactions between the composition and the environment are taken into account but in an abstract way: they are represented as aggregated values of input and output variables. Therefore, one may very well make and use BB models of a system without knowing its construction and operation. A BB model of a system corresponds with the *function perspective* on a system, which means 'looking at it' from the point of view of the using system.

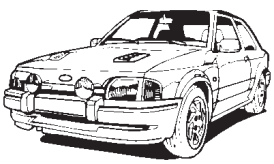
Figure 2.4 Representation of the black-box model

Figure 2.4 shows a well-known representation of the BB model. The internal interactions as well as the production of the system are condensed in the *transfer function*. It defines the relationships between the input and output variables. Ideally, the transfer function is a mathematical function from the domain of the input variables to the domain of the output variables. Knowing the transfer function means knowing how the system responds to variations in the values of the input variables by changing the values of the output variables. An example of a BB type of model is the control or management model of enterprises. It presents any enterprise as to consist of two subsystems: the managing system and the managed system. Note that this model is in fact the application of the general control model from physics to enterprises: the managing system equals the controlling system and the managed system equals the controlled system.

One must be cautious with respect to black-box models. It appears that this kind of model has been so dominant, particularly in the natural and the social sciences (including economics, business administration, and management sciences), that it is time and again confused with the notion of system. This confusion has led to the 'invention' of the *teleological* system notion. According to that notion, one specifies the *function* of the system in terms of the output variables and their interrelationships with the input variables. Obviously,

this means that one has created a BB model of the system! To avoid any misunderstanding, we will therefore refer to the system notion in this book as the *ontological* system notion, which is actually the only system notion. Lastly, regarding the term "behavior" in connection with BB models, we will use it as the manifestation of the (transfer) function in the course of time.

Figure 2.5 exhibits the black-box model of a car. It corresponds with the driver's perspective on a car. Therefore, the driver is the using system and the car is the used system. Through changing the values of the input variables (e.g., the position of the steering wheel), the driver is able to change the values of the output variables (e.g., the direction of the car). Ideally, as we have seen, the (transfer) function is a mathematical relationship between the input variables and the output variables. However, for most concrete systems, e.g. for cars, this is hardly possible. Therefore, in practice, the notion of function is a rather informal, loosely defined understanding of what kinds of (functional) behavior can be caused through manipulating the input variables. In this respect one also often uses the term "functionality", a term that should be preferred to "function".



the driver's perspective

function:
relationship between
input and output

behavior:
the manifestation of the
function in the course of time

functional (de)composition

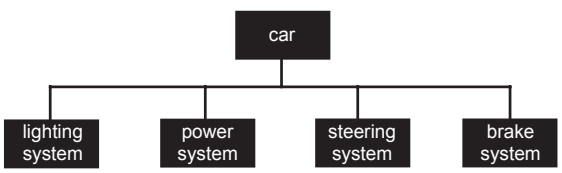


Figure 2.5 Black-box model and functional (de)composition

If the transfer function is too complicated to understand, the technique of *functional (de)composition* can be applied, through which the black-box model of a system is replaced by a structure of sub models with more readily understandable functions. Figure 3.4 shows a possible decomposition of a car. If a component is still too complicated, it can be further decomposed. The demonstrated decomposition of a car could be very helpful for a driving instructor to explain its functionality to a new student. Note, however, that the knowledge that one acquires about a system by means of functional decomposition is only functional knowledge, nothing less and nothing more. The only thing one does in functionally decomposing is helping to explain the functionality of a system, as we showed for the car example.

The idea that functional decomposition ultimately leads to knowing the construction of the system is a widespread and persistent misunderstanding. To elaborate this, a BB model is a purely conceptual division of the function or functionality of a system, completely independent of its construction and operation. Therefore, one can make virtually any decomposition one likes, and one can freely add or remove functional components. For example, we could add the security subsystem to the functional decomposition of the car. This explains why a number of (say N) analysts, given the task to model a part of an enterprise by applying a function-oriented technique, will end up with N different models, i.e., decompositions. Well-known examples of function oriented, or black-box model based, techniques are the DFD (Data Flow Diagram) [53] and SADT or IDEF0 [46]. So, one should be careful in applying these techniques for making constructional (WB) models, since they are useless for that purpose.

Next, it appears that people sometimes say that they are making a functional decomposition while they actually are making a constructional (de)composition. An example of this is the decomposition of a software system that is built according to a component-based design. In such a case, one better avoids the term "functional" and also bears in mind that constructional decomposition is just the reverse of constructional composition. This implies that there is no freedom whatsoever to choose the components; they are fully determined by the actual assembly.

Finally, it is a misunderstanding that one could choose the components in a functional decomposition such that they coincide with constructional components; that is just absurd. BB models and WB models are fundamentally different types of models. There is no way of simply mapping one to the other. If you are still not convinced, then ask the car mechanic, next time you bring your car to the garage for maintenance, to disassemble your car according to the functional decomposition as given in Figure 2.5, or to any other one. The only thing you may achieve is that you drive the mechanic crazy!

As a final illustration of the distinction between function and construction, let us consider a clock. The function perspective of a clock is quite simple and therefore known to almost everybody: it tells the time, measured in the globally standard way in which a cycle of a day and a night is divided into 24 hours, an hour into 60 minutes, and a minute into 60 seconds. The way in which the measured values are displayed may differ, which means that one has to know a few different user interfaces, of which the two most well known are the analogue display (clock face and hands) and the digital display (decimal digits). Although identical in their function, clocks can differ very much in the way they are constructed. There are mechanical clocks, electronic clocks, water clocks, pneumatic clocks, etc. Clearly, the function of a clock has no intrinsic relationship with its construction. The function is described in terms of the user of the clock: time, hours, minutes, etc. The construction is described in terms of components and elements and their interactions. These components and elements are not 'aware' of hours and minutes and seconds. Instead, they are 'aware' of mechanical forces and speed, or of electrical tension and current, etc. It is the craftsmanship of the constructional designer of a clock that bridges the mental gap between the function and the construction, such that the operation of the construction brings about the desired functional behavior of the clock.

3 The Notion of Ontology

In this chapter, the notion of (system) ontology is introduced and discussed. The notion implies two powerful abstractions. The first one is the universal transaction pattern. Applying it provides a reduction of complexity of over 70% in terms of the size of the resulting models. The second abstraction is the distinction between three human abilities. It results in an additional reduction of complexity of over 70%. The potency of ontological modeling will by illustrated on the well-known Ford case.

3.1 Actors – Coordination – Production

Following the distinction that has been drawn in Chapter 2 between the function and the construction of a system, we call the collective services that an enterprise provides to its environment the *business* of the enterprise; it represents the function perspective. Likewise, we will call the collective activities in which these services are brought about and delivered, including the human actors that perform these activities, the *organization* of the enterprise; it represents the construction perspective. As was recognized already, organizations are designed and engineered artifacts, like cars, space shuttles, and information systems. The distinctive property of organizations is that the active elements are human beings, more specifically human beings in their role of social individual or *subject*.

These subjects perform two kinds of acts: production acts and coordination acts. These acts have definite results: production facts and coordination facts, respectively.

By performing *production acts* (P-acts for short) the subjects bring about the goods and/or services that are delivered to the environment of the organization. A production act is inherently either material or immaterial. Examples of material acts are all manufacturing acts, as well as storage and transportation acts. Examples of immaterial acts are the judgment by a court to sentence someone, and the decision to grant an insurance claim. By performing *coordination acts* (C-acts for short) subjects enter into and comply with commitments towards each other regarding the performance of production acts. A subject in its fulfillment of an actor role is called an *actor*. In general, actor roles do not coincide with

or map directly to the existing organizational functions, like salesperson, secretary, and accountant.

In conformance with the distinction between production acts and coordination acts, we distinguish between two worlds in which each of these kinds of acts have effect: the *production world* or P-world, and the *coordination world* or C-world, as discussed in Chapter 2. A state of the P-world is a set of P-facts, and a state of the C-world is a set of C-facts. To be more precise, the state of the P-world at a particular point in time is the set of P-facts that have been created up to that point in time, and the state of the C-world at a particular point in time is the set of C-facts that have been created up to that point in time. Therefore, we keep track of the complete history of both worlds. This is fully compliant with the basic notion of fact: a fact can be created but it cannot be undone; this would be a violation of reality[1].

Figure 3.1 contains an example of a coordination act. It concerns the request by the subject John to the subject Mary to become member of the library. For the sake of a simple unique identification of persons who are members of the library, we choose to apply the notion of membership. Therefore, every new case of a person becoming a member is the creation of a new instance of the object type membership. In the example, the instance created is the one referred to as #387. One of the related facts of this membership is that John is the member. The time "1-4-2002" is the intended creation time of the fact; in more conventional language, it is the starting date of the membership.

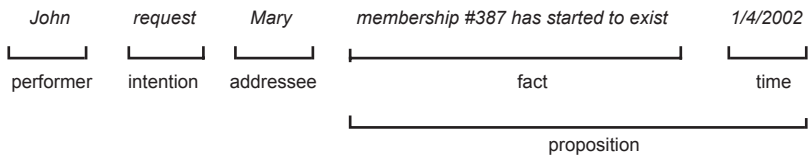


Figure 3.1 Standard notation of a coordination act

The way in which the C-act in Figure 3.1 is formulated is called the *standard notation*. This notation is very helpful in analyzing conversations, since it makes the intention and the proposition explicit, contrary to a formulation in colloquial language, where John might say to Mary: "I'd like to become a member". Moreover, the sequence of the parts in the standard notation is such that it can be read in a more or less natural way.

---

[1] This does not mean that an organization must keep records of all its acts over its entire life. For practical reasons, one has to make decisions about the storage time of records (or it is enforced by law). The point is that deleting a record does not undo the represented fact; it only removes the record of it.

3.2 The Universal Transaction Pattern

Coordination acts are performed as steps in universal patterns, called *transactions*. These patterns always involve two actor roles and are aimed at achieving a particular result. A transaction evolves in three phases: the *order* phase (O-phase for short), the *execution* phase (E-phase for short), and the *result* phase (R-phase for short). One of the two partaking actor roles is called the *initiator*, and the other is called the *executor* of the transaction. In the order phase, the initiator and the executor work to reach agreement about the intended result of the transaction, i.e., the production fact that the executor is going to create as well as the intended time of creation. In the execution phase, this production fact is actually brought about by the executor. In the result phase, the initiator and the executor work to reach agreement about the production fact that is actually produced, as well as the actual time of creation (both of which may differ from what was originally requested). Only if this agreement is reached will the production fact come into existence.
Let us have a look at a simple transaction, conducted in face-to-face communication between two subjects. It is about buying a bouquet of flowers by a client (C) from a florist (F):

(1)	C:	I'd like to have such a bouquet
(2)	F:	Very well, sir
(3)		< actual delivery of the bouquet to the client >
(4)	F:	Here you are
(5)	C:	Thanks

Clearly, C is the initiator of the transaction and F is the executor. Lines 1 and 2 constitute the order phase of the transaction; line 1 contains the request of the proposition that the client possesses a particular bouquet, and line 2 the corresponding promise. Line 3 constitutes the execution phase; it refers to the production act and fact. Lines 4 and 5 constitute the result phase of the transaction; line 4 contains the statement that the client has got a particular bouquet, and line 5 the corresponding acceptance. That these five lines constitute a transaction becomes fully clear if we transform the spoken sentences in the standard notation (cf. Figure 3.1), as is done below:

(1)	C:	request:	F:	Client has a bouquet B: ASAP
(2)	F:	promise:	C:	Client has a bouquet B: ASAP
(3)		< actual delivery of the bouquet to the client >		
(4)	F:	state:	C:	Client has a bouquet B: now
(5)	C:	accept:	F:	Client has a bouquet B: now

From this standard notation, one immediately sees that the fact part of the proposition is identical in the lines 1, 2, 4, and 5. It is also more precisely specified: it refers to a named type of bouquet, called B. What also becomes explicit now is the time part of the proposition; they differ. In the order conversation, the time is "ASAP" (as soon as possible). It must be considered as the default value that is rarely made explicit in the actual (verbal or nonverbal) communication. The same holds for the value "now" in the result conversation. The reader might ask himself or herself why the payment for the flowers is omitted. The answer is that this would be another complete transaction. This also makes clear that, although payment for some delivery of goods or a service is generally connected to it, this is not a necessity.

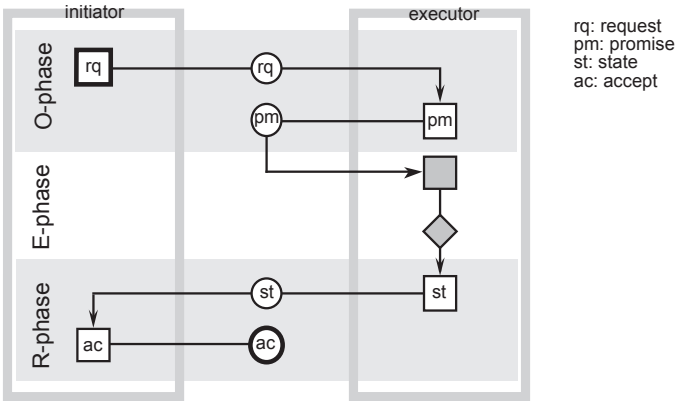


Figure 3.2 The basic pattern of a transaction

The example of buying a bouquet of flowers follows completely the basic pattern of a transaction, as shown in Figure 3.2. An open or white box represents a C-act type and an open or white disk represents a C-fact type. A gray box represents a P-act type and a gray diamond a P-fact type. The initial C-act is drawn with a bold line, as is the terminal C-fact. In the background, the three phases are indicated. Every act and fact belongs to one of these. The gray-colored frames, denoted by "initiator" and "executor" represent the *responsibility areas* of the two partaking actor roles. Line 1 in the florist example corresponds with the request act and fact, line 2 with the promise act and fact, line 3 with the P-act and the P-fact, line 4 with the state act and fact, and line 5 with the accept act and fact.

The basic pattern is the course a transaction takes when the initiator and the executor keep *consenting* to each other's acts. However, they may also *dissent*. Instead of promising, one may respond to a request by declining it, and instead of accepting, one may respond to a statement by rejecting it. It brings the process to the C-world state "declined" or "rejected" respectively. These states are indicated by a double disk, meaning that they are *discussion* states. If a transaction ends up in a discussion state, the two actors must 'sit together', discuss the situation at hand, and negotiate about how to get out of it. The outcome can be a renewed request or state, or it can be an unsuccessful termination of the transaction (a

stop). The basic pattern from Figure 3.2, extended with the two dissent patterns, is called the *standard* pattern. It covers the middle part of Figure 3.3.

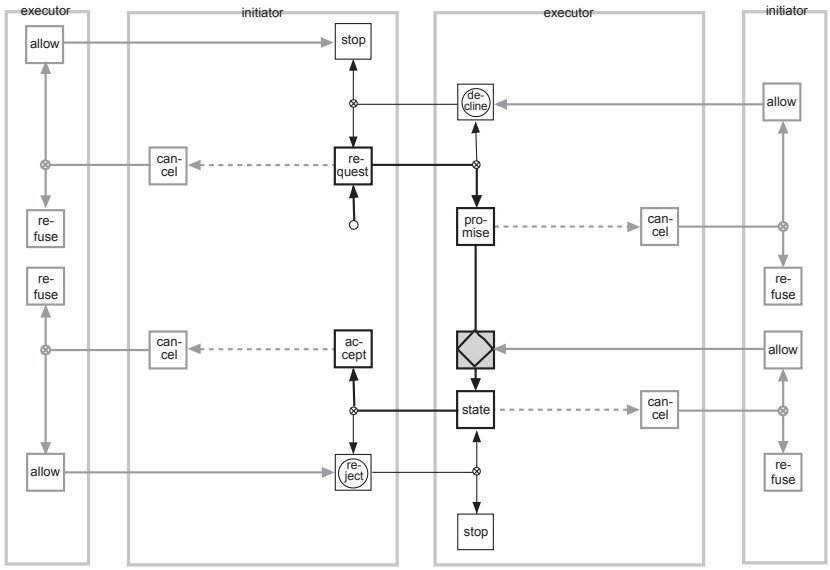


Figure 3.3 The complete pattern of a transaction

In practice, it is quite common that either the initiator or the executor of a transaction wants to cancel an act (which may result in a partial or complete rollback of the transaction). This is accommodated by the option to cancel any C-act in the basic pattern at any time. This gives rise to the four cancellation patterns, on the left and the right side of Figure 3.3. The dotted arrows represent preconditions. So, for example, in order to cancel an accept, the accept act must have been performed.

Every *transaction process* is some path through this complete pattern, and every *business process* in every organization is a tree structure of such transaction processes. This holds also for processes across organizations, like in supply chains and networks. That is why the transaction pattern is universal: people always and everywhere conduct business (of whatever kind) along this pattern. Moreover, coordination steps are considered to be the ontological *atoms* of (the business process of) an organization, the transactions being the *molecules* [14].

To conclude this section, we like to emphasize that C-acts can be performed in three different ways. Firstly, they can be performed *explicitly*. This means that some observable act counts as performing the C-act. An example is the ordering of a bouquet of flowers by speaking to the florist. Next, C-acts can be performed *implicitly*. It means that performing some C-act also counts as performing another C-act. An example is that the putting in front of the customer of the bouquet by the florist not only counts as performing the state act but also as the request in the payment transaction. Lastly, C-acts can be performed

tacitly. It means that there is no observable act that could count as performing the C-act. An example is that there is no response to a written request for payment (an invoice) that could count as performing the promise (which is quite common, applying the rule that no news is good news). It is important to understand, however, that tacitly performing a C-act is as valid as explicitly performing it!

3.3 Forma – Informa – Performa

Three distinct *human abilities*, called *forma*, *informa*, and *performa*, play a crucial role in the operation of actors. These are pictured in Figure 3.4. The *forma* (Latin for "form") ability concerns the form aspects of communication and information. In other words, we are talking now about such things as the uttering and perceiving of sentences in some language, the syntactical analysis of such sentences, coding schemes, transmission of data, and storage and retrieval of data or documents. For the sake of convenience, we consider the forma ability for also comprising the physical substrate in which information is encoded. The *informa* ability concerns the content aspects of communication and information ("informa" means, in Latin, what is in the form). Thus, we are dealing now with communication and information while fully abstracting from the form aspects. In other words, we are concerned with such things as the sharing of thoughts between people, the remembering and recalling of knowledge, and reasoning. The *performa* ability ("per-forma" means, in Latin, through the form) concerns the bringing about of new, original things, directly or indirectly by communication. We are talking now about engaging into commitments, and about decisions, judgments, etc. We consider the performa ability as the *essential* human ability for doing business of any kind.

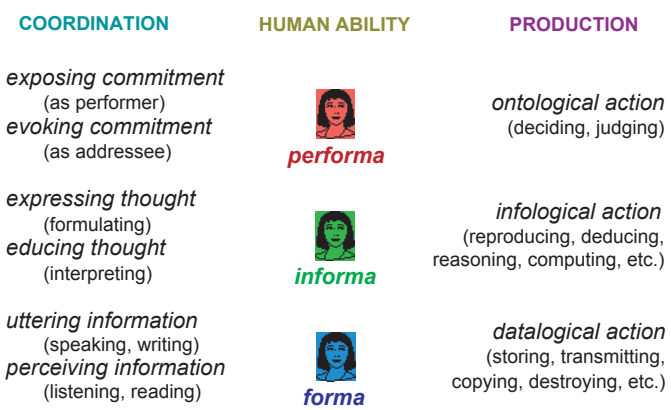


Figure 3.4 The three human abilities

3.4 The Organization Theorem

The process of performing a coordination act is not trivial at all. Figure 3.5 exhibits the typical kinds of communicative acts that have to be performed in order to successfully perform a coordination act. There are three kinds of communicative acts, corresponding with the three human abilities as contained in Figure 3.4: performative acts, informative acts, and formative acts.

In order to have the coordination act from P to A successfully performed, P must expose his commitment in a performative act, addressed to A, and A has to evoke in her the corresponding commitment to respond adequately. This act is part of the *performative exchange* between P and A. As a rule, it is also the only act in this exchange. Additional acts are generally about clarifying the mutual *social understanding* of the coordination act, in particular, of its intention. Arriving at a common social understanding of the coordination act is called the *performa condition* for successful coordination.

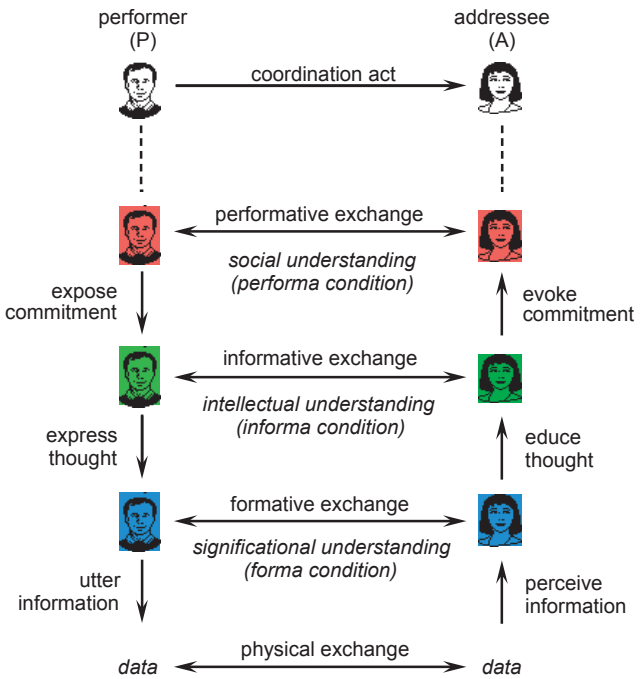


Figure 3.5 The process of performing a coordination act

The only way for P to expose its commitment and to make it knowable to A, is to express it through its informa ability, followed by the inducement in the mind of A of an equivalent thought, by means of its informa ability (because we are now at the informa level of abstraction, we use the term "thought" as the most general designation of a state of mind). P does so in an *informative exchange* with A. Such an exchange consists of informative acts,

which aim at achieving the *intellectual understanding* of the coordination act between P and A. There are at least two informative acts. One is the act in which P is the speaker and A the hearer, and in which P informs A about the content of the coordination act. The other one is the act in which A is the speaker and P the hearer, and in which A confirms to P that it has (intellectually) understood it. More acts may be needed, however, in case P did not completely specify its commitment in the first informative act. Arriving at a common intellectual understanding of the coordination act is called the *informa condition* for successful coordination.

Expressing a thought can only be done by formulating it in a sentence in some language, and at the same time uttering it in some form, such as speaking or writing. A similar reasoning holds for educing the thought in A's mind. This requires that there be something to be perceived, for example, through listening or reading. So, there is another level at which P and A have to arrive at a common understanding, the forma level. We now speak of *significational understanding*, which means that P uses a language that is known to A so that A is able to recognize what it observes as utterances in that language. Significational understanding is arrived at in a *formative exchange*. A formative exchange consists of at least two formative acts. One is the uttering by P of the sentence to A, and the other is the acknowledgement by A of having perceived this sentence. There is a formative exchange needed for every informative act. This is logical, since every informative act has its own content to be uttered and perceived. Arriving at a common significational understanding of the coordination act is called the *forma condition* for successful coordination. Lastly, there is a physical exchange needed of the substrates in which the utterances of P are encoded and which can be transported to A in order to perceive the encoded information from the structures of the substrate. We will not elaborate this physical level; we just assume that there is some (physical) channel for exchanging data.

A similar distinction between the three levels of abstraction can be made on the production side. The *forma* ability now concerns the ability to deal with recorded information items, called data or documents (see Figure 3.4). Examples of P-acts regarding data or documents are storing, copying, transmitting, and destroying. Collectively, we will call these acts datalogical acts, following the distinctions made by Langefors [38]. So, the transactions in which datalogical acts occur are called *datalogical transactions*. By the distinction Langefors made between information and data, he clarified the problems that information system developers and researchers encountered in the 1960's. The true but yet shocking discovery he made was that data do not contain information, as everyone believed until then. Information is something that is created when data are interpreted. Due to this discovery, Langefors had great influence on the information system development methods in the 1970's and later.

The *informa* ability on the production side concerns the ability to reason, compute, derive, etc., as well as to reproduce remembered knowledge. These acts are collectively called infological acts, and the transactions in which they figure as P-acts are called *infological transactions*. Lastly, the *performa* ability concerns the ability to establish original new things, like creating material products or making decisions. Because these acts are at the core of

doing business (on the production side), they are called the essential or *ontological* production acts. Accordingly, the transactions in which they occur as P-acts are called *ontological transactions*. The infological transactions are considered to be a matter of realization of the ontological transactions, because their only effect is the reproduction or the derivation of knowledge that is needed by the ontological actors. In a similar way, the datalogical transactions are considered to be a matter of realization of the infological transactions, since their function is to care for the storage and retrieval, the copying, and the transmission of documents containing the information that is needed by the infological actors. These interrelationships are elaborated in the organization theorem, which is represented in Figure 3.6.

The *organization theorem* states that the organization of an enterprise is constituted as the layered integration of three systems: the B-organization (from Business), the I-organization (from Intellect), and the D-organization (from Document). The relationships among them are that the D-organization supports the I-organization, and the I-organization supports the B-organization. The integration is established through the cohesive unity of the human being. All three systems are in the category of social systems, which means that they are similar as far as coordination is concerned: the elements are subjects that enter into and comply with commitments to each other regarding production acts. They differ only in the kind of production: the production in the B-organization is ontological, the production in the I-organization is infological, and the production in the D-organization is datalogical. As a consequence, we will speak of the ontological model of the B-organization, the I-organization, and the D-organization, meaning that we consider coordination only at the 'red' level in Figure 3.5[2]. The B-organization, the I-organization, and the D-organization are called aspect systems of the (total) organization of the enterprise.

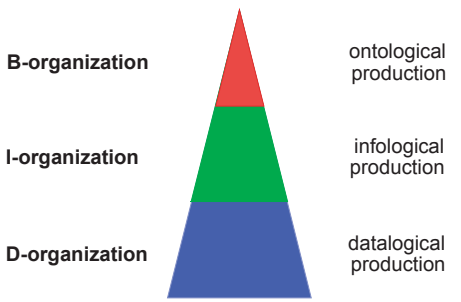


Figure 3.6 Representation of the organization theorem

Of course, an enterprise is more than just a well-established integration of these three aspect organizations. To begin with, human beings are biological beings as well, which means that they need a particular environment to live in, as well as specific facilities to make their

[2] Note that we also call the production in the B-organization ontological, meaning that the P-facts are original new things.

biological lives comfortable. Being a biological individual includes being a physical thing. Therefore, certain physical requirements must be met as well, such as the need for workspace and mobility services. Moreover, a human being is an emotional being, a psychological being, and so on. While we do recognize that these additional aspects must be considered, they are not our concern now because they do not directly relate to the notion of enterprise that we have adopted. We are aware, however, of the precondition that all these other perspectives on an organization must have been dealt with in a satisfactory way.

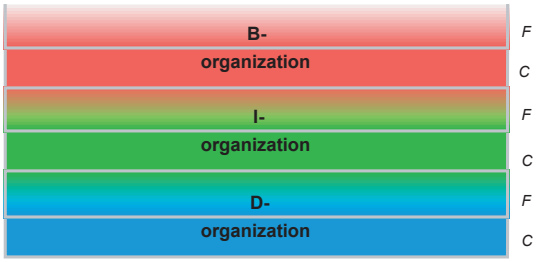


Figure 3.7 The layered integration of an organization

Figure 3.7 exhibits in another way the layered nesting of the three aspect systems. The distinction of the function perspective (F) and the construction perspective (C) serves to exhibit their layered nesting in a more precise way. As the figure shows, the function of the I-organization supports the construction of the B-organization, and the function of the D-organization supports the construction of the I-organization. To emphasize the intermediate nature of the function perspective, the corresponding bars have the colors of the bar above and the bar below run into each other. One must think of the construction of the environment of the enterprise as the bar above the F-bar of the B-organization. In a practical sense, it consists of the concrete and explicit specification of the societal context in which the enterprise operates.

In order to discuss the exact meaning of the integration of the three aspect organizations, we choose the I-organization as the starting point. When we take the function perspective, we see that the I-organization provides information services to the construction of the B-organization, i.e., to the B-actors. It is crucial to recognize that the information service provided is determined by these B-actors and that it is thus (functionally) described in terms of the operation of the B-organization. For example, some B-actor in a company might want to know at the end of every working day the turnover of that day. Turnover is a notion that means something to him or her. It fits into a framework of other economic notions that are basic for his or her role in the B-organization. In contrast with this B-actor, the I-actors principally do not know about turnovers. For instance, they will not get nervous if the turnover figures drop. They only use the term for the sake of the service delivery. It is no more than a label for the outcome of a particular calculation. However, as I-actors they are part of the construction of the I-organization, and therefore concerned with their duty as executor of I-transactions in which I-products are produced. An example of an

I-transaction is one in which the daily total of a well-defined set of values is calculated, and in which the result, labeled "turnover", is delivered to the initiator of that transaction. So, for the executing I-actor, turnover is defined as the sum total of something, contrary to the initiating actor. But who is this initiating actor? Obviously, it is the B-actor who needs to know the turnover; but how can he or she be the initiator of an I-transaction? The answer to this question is the key to understanding the layered nesting of the B-organization and the I-organization. It is as follows. The subject who fulfils the B-actor role disposes of all three human abilities, as discussed before. In performing ontological production acts, he or she exclusively uses the performa ability. On the other hand, as initiator or executor of B-transactions, he or she needs all three abilities for performing coordination acts (see Figure 12.3).

Apparently, it is not so hard to switch between abilities. This is exactly what happens: the subject temporarily takes the shape of I-actor, and in that shape is perfectly fit to participate in I-transactions. After having done this, and after having accepted the result of the I-transaction, the subject switches back, so to speak, to his or her B-actor shape, and resumes his or her work in that shape, supplied now with the knowledge of the turnover figure.

Let us have a closer look at the I-actor who produced the turnover value as the result of the conducted I-transaction. The competence of this actor clearly is to do additions (and perhaps to perform other mathematical operations). In order to compute the sum of the individual figures, he or she must dispose of the figures. How does this actor get these figures, or, more generally, how does an I-actor get data? The answer is quite similar to the answer that was given to the question of how a B-actor acquires knowledge. It reads that the subject fulfilling the role of I-actor has to take his or her shape of D-actor, and, in that shape, initiate a D-transaction of which the result is the delivery of a document that contains the desired information. After having accepted this result, the subject steps back in his or her shape of I-actor, in which he or she is able to interpret the information provided and to perform the necessary infological acts.

3.5 Ontological Modeling

The specific methodology that will be used for constructing the ontology of an organization is DEMO (Design & Engineering Methodology for Organizations) [20, 21, 13, 45]. In this methodology, the complete ontological model of an organization is understood as the model of its B-organization. It consists of four aspect models, which are represented by particular diagrams and tables (Figure 3.8). The triangle in this figure is identical to the top triangle of Figure 3.6.

The Construction Model (CM) specifies the composition, the environment and the structure of the organization. It contains the identified *transaction types*, the associated *actor roles* as well as the *links* to *production* or *coordination banks* where the production and coordination facts are stored. It is usually split into the InterAction Model (IAM) and the InterStriction Model (ISM). The IAM contains the interaction relationships between actor

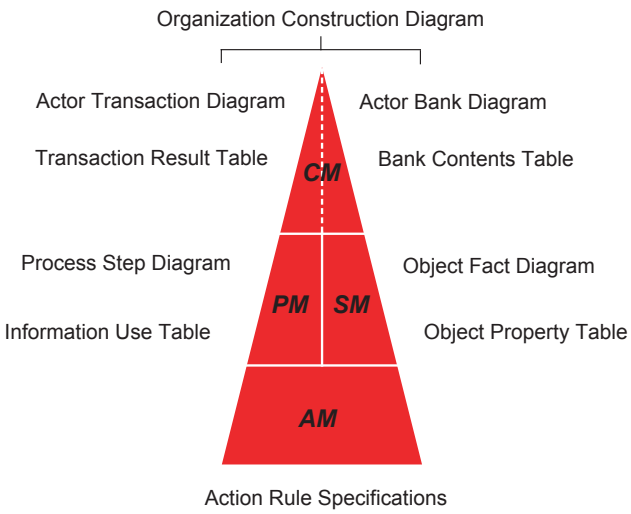


Figure 3.8 The four aspect models in DEMO

roles, thus the transaction types. The ISM contains the information links between actor roles and information banks (the interpretation of a transaction symbol as a production facts store and a coordination facts store). The Process Model (PM) details each single transaction type of the CM according to the universal transaction pattern. In addition, the initiating and waiting relationships between transactions are shown. The Action Model (AM) specifies the imperatively formulated *business rules* that serve as guidelines for the actors in dealing with their agenda[3]. It contains one or more action rules for every agendum type. The State Model (SM) specifies the *object classes*, the *fact types* and the declarative formulations of the *business rules*.

In order to demonstrate how applying Enterprise Ontology reveals the deep, essential structure of an organization, we take the well-known Ford Case [29] as an example:

> When Ford's purchasing department wrote a purchase order, it sent a copy to accounts payable. Later, when material control received the goods, it sent a copy of the receiving document to accounts payable. Meanwhile, the vendor sent an invoice to accounts payable. It was up to accounts payable, then, to match the purchase order against the receiving document and the invoice. If they matched, the department issued payment. The department spent most of its time on mismatches, instances where the purchase order, receiving document, and invoice disagreed...One way to

[3] Agenda is the plural form of the Latin word "agendum", which means what has to be done. Therefore, an agendum is a 'to do' item. Actors constantly loop through the so-called actor cycle, each time trying to deal with an agendum. Examples of agenda are the being requested or the being promised of a transaction.

improve things might have been to help the accounts payable clerk investigate more efficiently but a better choice was to prevent the mismatch in the first place. To this end, Ford instituted 'invoice less processing'. Now when the purchasing department initiates an order, it enters the information into an on-line database. It does not send a copy of the purchase order to anyone. When the goods arrive at the receiving dock, the receiving clerk checks the database to see if they correspond to an outstanding purchase order. If so, he or she accepts them and enters the transaction into the computer system. (If receiving can't find a database entry for the received goods, it simply returns the order.)'

The result of modeling the Ford case is presented in Figure 3.9. Only the IAM and the PM are produced. The upper part exhibits the IAM. Three essential transaction types are identified. The first one (T1) concerns the transfer of property of the goods delivered. Let us call this "order delivery". This transaction has to be distinguished from the physical shipping of the goods to the Receiving department of Ford, as represented by transaction type T2, called "order shipment". The third transaction type (T3) concerns the payment of orders and is therefore called "order payment". Because we abstract at the ontological level also from organizational functions and departments and even from company borders, the actor roles get 'ontological' names. In this case, the initiator of T1 (A0) is called "Customer" and the executor of T1 (A1) is called "Deliverer". Actor role A1 appears to be the initiator of both T2 and T3: every transaction T1 (delivery) encloses a transaction T2 (transport) and a transaction T3 (payment). The executors of these transaction types are called "Shipper" and "Payer" respectively.

The PM of the Ford case is shown in the lower part of Figure 3.9. From the coordination state "T1 is promised" (T1/pm), actor A1 performs three acts. Firstly, it requests a transaction T2, in order to get the goods of the order shipped. Secondly, it requests a transaction T3, in order to get paid for the delivery. Performing this request has to wait for T2 to be accepted (this is indicated by the dashed arrow from T2/ac to T3/rq). This waiting condition reflects the agreement between Ford and the vendors that deliveries are paid after the goods have been received. Thirdly, A1 performs the P-act of T1, thus the actual decision to transfer the ownership of the transported goods to the customer. This act has to wait until T3 has been accepted (indicated by the dashed arrow from T3/ac). The actual transfer of ownership is achieved through its acceptance by the customer (T1/ac).

In order to see by which organizational units the atomic acts in the PM are performed in the new situation at Ford, the departments in Ford and in the vendor company who fulfill the identified actor roles are mentioned next to the corresponding steps. To avoid confusion, the names of the departments within the vendor company are put between brackets. Next, a department's name is printed in italics if it performs the corresponding C-act tacitly.

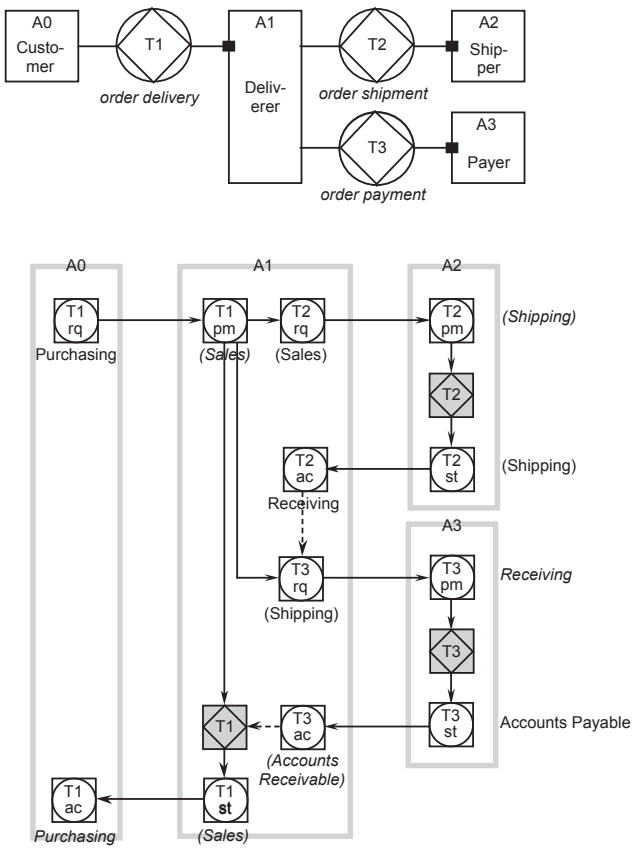


Figure 3.9 Part of the ontological model of the Ford Case

The Ford case was presented in [29] as an example of radically re-engineering business processes, in conformity with the paper's title "don't automate, obliterate". From the analysis of the case with DEMO we make the following observations. *Firstly*, the ontological model of the situation is the same before and after the radical(!) change. This demonstrates the stability of the essential model. It makes having such a model a precious possession for every organization. *Secondly*, the 'invoice-less' process in the new situation at Ford is only invoice-less to the extent that no paper or electronic invoices are sent to Ford by the vendor. However, the essence of an invoice still exists, being the request for payment. In the new situation, this request is just performed implicitly, which means that there is some other act that counts as performing the request for payment. The analysis tells us that the statement of the shipment transaction (T2/st) counts as the request for the payment transaction and that the accept act (T2/ac) counts as the promise. Therefore, instead of calling the new situation 'invoice less' it would have been more appropriate to say that the received shipment document is (also) the invoice. *Third*, only the deep insight in the business process as provided by Figure 3.9 can explain the conflicts that occur as a consequence

of delegation of authority, both by Purchasing to Receiving in T2/ac and by Accounts Payable to Receiving in T3/pm. The practical decision to do this and thus to deviate from the 'ideal' organizational implementation is fully justified. At the same time, it means that these departments need regular meetings for discussing and tuning in the backgrounds from which they enter into and comply with commitments. This need may be well known to you already from your practical experience in cases of delegation, now you know also profoundly why.

4 System Development

By system development, we understand the bringing about of a new system or of changes to an existing system. The system development process comprises all activities that have to be performed in order to arrive at an implemented system. In this chapter, we will investigate the nature of this process, which will lead to the conception of a generic model of the system development process. Particularly, we will study the role of ontology and technology in the development process.

There are two major actor roles involved in any development process: the customer and the supplier. The *customer* places the order with the supplier for developing the system. The *supplier* executes the development activities and delivers in the end the system to the customer. By conceiving customer and supplier as actor roles, as introduced in the previous chapter, their authorities and responsibilities are clarified, as well as their required competences.

4.1 Designing

Any development process concerns two systems, which we will call the using system and the object system. By the *object system* (OS) is understood the system which is going to be developed. By the *using system* (US) is understood the system that is going to use the services (the functionality) of the object system.

As the explanatory example of the US we take the *sales department* of an enterprise. One of the services that the US takes from the OS is the monthly turnover of the enterprise. As the explanatory example of the OS we take a *sales information system.* One of the monthly services it produces is the sum of a number of numerical values. This sum is interpreted by the US as the monthly turnover.

We will divide the complete system development process in three major sub processes or phases: design, engineering, and implementation. The goal of this section is to clarify the design phase. To start with, let us have a closer look at the process of designing a system in a very general way. Figure 4.1 exhibits the basic view that we consider appropriate for deeply understanding the process of designing.

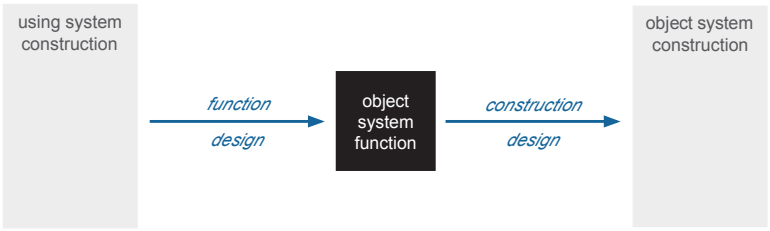


Figure 4.1 The system design process (1)

The figure shows that the function perspective and the construction perspective on systems, as well as the corresponding black-box model and white-box model (cf. Chapter 2), play a crucial role in understanding the system design process. Apparently, two major activities can be distinguished in the design of the OS. The first one is called "function design" and the second one "construction design". *Function design* starts from the construction of the US and ends with the function of the OS. The function of the OS is considered to be a black-box model of the OS. In line with what has been said about black-box models in the previous chapters, this means that the specified function of the OS does not contain any information about or even any clue to the construction of the OS. In other words, the function of the OS must be specified fully and only in terms of the construction of the US.

As said, the function of the OS supports the construction of the US. Put the other way around: the construction of the US uses the function of the OS. Why the construction of the US, and not its function? The answer is that there must always be an alternation of function and construction in the support-use relationship of systems. A function cannot support another function directly, because functions cannot have a need for support; only constructions can. To exemplify this, the sales department as a part of the construction of the enterprise can have a need for a sales information system because the salespersons are elements in this construction and they can be in need of such a support. The sales department as a black box is a functional abstraction that may be useful for some construction above it but it does not concern the operational activities of selling. Conversely, the function of the OS is always, and necessarily, expressed in terms of the construction of the US. So, for example, the salespersons in (the construction of) the US may have a need to know the monthly turnover of the enterprise. The notion of turnover is an economic notion that takes its real meaning from the larger economic context in which the enterprise operates. The value of the monthly turnover is something by which people in the sales department may become excited or depressed. In specifying the function of the OS, it is this rich meaning of turnover that the salespersons will express to the functional designers of the OS. It is understood that the functional designers must possess sufficient knowledge of the enterprise in order to be well-matched opponents for the salespersons in the process of function design.

The nature of the second design sub phase, *construction design*, is quite different from the first one. It starts from the specified function of the OS and it ends with the construction

of the OS. Without intending to wrong the functional designers, the job of the constructional designers[1] is definitely more creative, because they have to bridge the mental gap between function and construction. This is nothing less than establishing a correspondence between systems of different kinds: the system kind of the US and the system kind of the OS. To illustrate this, the notion of turnover must now be viewed as the result of a calculation. The constructional designer is aware of the fact that the sales information system that he or she is going to design will never have any 'understanding' of turnover. The name "turnover" will at best be a label, assigned to a variable in the computer program; however, this variable can equally well be named "Mickey Mouse". It is the task of the constructional designer to get the precise definition of turnover from the functional designer. Basically, this means that the real semantics of the notion of turnover is replaced by its formal semantics. From these formal semantics, the constructional designer devises an appropriate calculation rule.

The result of the activity function design is a functional model of the OS, which by nature is a black-box model. It contains all *functional specifications* for the OS to be built. They include the specifications for the interface through which the services of the OS are provided to, or acquired by, the US, next to the specifications of the services themselves. The major input for the activity function design is the *functional requirements*, provided by the US, as shown in Figure 4.2. They need not be equivalent to the functional specifications, as contained in the functional model of the OS, for several reasons. Firstly, requirements may be unfounded. Fortunately, the ontological model of the US provides an objective yardstick for determining unfounded requirements, since the ontological model of the B-organization specifies the information that is needed for every actor role. All other information that the fulfillers of an actor role may want to have, they do not need. Next, determining requirements includes their validation, something that can be achieved only in thorough discussions with people 'in the business'. Again, the ontological model of the B-organization offers great help, because one cannot forget essential requirements.

Secondly, both the functional and, particularly, the constructional specifications must be feasible. By this is meant that the ultimate OS can be implemented, given the available technology. This is only one aspect of feasibility. Other, most important, ones are that the whole development process can be executed within the agreed upon time and for the agreed upon costs.

So, in general, designing includes negotiation, in order that the end result is a *balanced compromise* between (reasonable) requirements and (feasible) specifications. This holds, mutatis mutandis, also for the construction design. The major input for this phase are the *constructional requirements*, provided by the US. These are often also called *non-functional* requirements. Constructional requirements regard the performance characteristics with which the OS is going to provide its services. Examples of such characteristics are response time, robustness, and security. In practice, the constructional requirements are often made

[1] Note that both functional designer and constructional designer are actor roles. They may be fulfilled by the same person or by different persons.

clear by means of service level agreements (SLAs) regarding each of the specified performance characteristics.

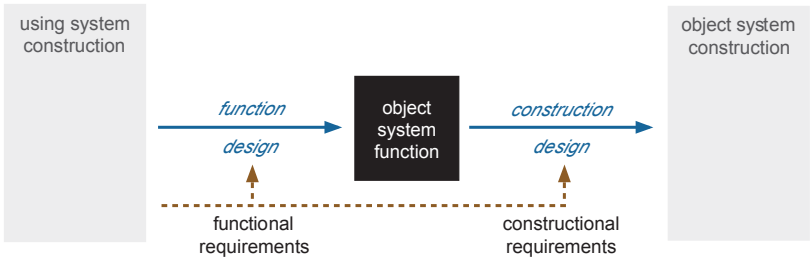


Figure 4.2 The system design process (2)

The negotiation, as mentioned above, is formally conducted by the customer and the supplier but in practice delegated to respectively the future users of the OS and the designers. As said before, the reasonability of the provided functional and constructional requirements can be fairly well determined on the basis of the ontological model of the US. The feasibility of the ultimate functional and constructional specifications can only be determined in iteration between function design and construction design, as shown in Figure 4.3.

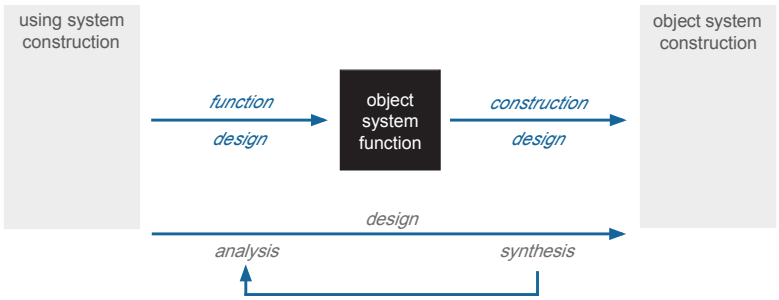


Figure 4.3 The system design process (3)

The figure conforms to the insight of Alexander [2], who understands design as a process of alternate analysis and synthesis steps. An analysis step is one in which the problem is better understood; a synthesis step is one in which the solution becomes more clear. It clarifies that designing is not a one-way process but that there will usually be iteration. Put differently, the devising of the functional specifications and the devising of the constructional specifications will usually take place incrementally and alternately. The distinction between analysis and synthesis is a perfect example of the separation of concerns in the design process ([17]). During an analysis step, the concerns are the correct understanding of the semantics, the completeness, and the consistency of the function of the OS. In a synthesis step, the concerns are the correct replacement of functional semantics by formal semantics and the efficiency of the construction of the OS. Unfortunately, conducting the design

process as an alternation of analysis and synthesis steps may lead to suboptimal designs, as explained in [12].

Alexander's view on the design process supports the recognition that a good design is always a balanced compromise between reasonable requirements and feasible specifications [15]. As an almost logical consequence, it means that one already needs a substantial amount of technical knowledge for fulfilling adequately the role of functional designer. This outcome corresponds with our observation that consultants who have only an economic or managerial or business education commonly have little added value in the design phase. In general, people with both an organization- and a technology-oriented education appear to play a crucial role in design processes since they are able to understand the distinction between the two phases as well as their interconnection.

4.2 Engineering and Implementing

The development phases after designing are engineering and implementing. Next to ontology, another notion plays an important role now, namely *technology*. Let us start by focusing on the role of ontology. To this end, we use the picture shown in Figure 4.4. As we know from the previous chapter, the ontology or the ontological model of a system is a model of its construction that is completely independent of the way in which it is implemented. By definition, it is the highest-level constructional model of a system.

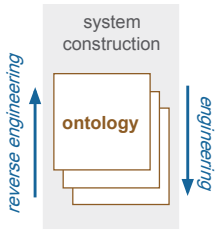


Figure 4.4 The role of ontology in engineering

The *engineering*[2] of a system is the activity in which a series of constructional models are produced. Every model is fully derivable from the previous one and the available specifications. Contrary to designing, engineering is rather a matter of craftsmanship than of creativity. Engineering starts from the ontological model and ends with the implementation model. In practice, it is often the case that one or more constructional models are missing, a typical situation being that only the implementation model is available (which is the case for many legacy software systems). The process of reconstructing the higher-level models from the implementation model is known as *reverse engineering*.

[2] Note that we use the term "engineering" here in the narrow sense, contrary to it use in e.g., mechanical engineering and enterprise engineering. A well-known synonym for engineering in the narrow sense is "technical design".

Lastly, *reengineering* a system means redoing the engineering process, starting from the ontological model, which may or may not have been changed as a result of redesigning the system. Consequently, redesigning a system inherently implies reengineering it.
By *implementing* a system is understood the assignment of technological means to the constructional elements in the implementation model. Once implemented properly, the system can be put into operation. This activity is shown in Figure 4.5.

Figure 4.5 Implementing the construction of a system

From the discussion of ontology in Chapter 4, we have learnt that the core elements in the ontological model of an enterprise are actor roles, coordination acts/facts, and production acts/facts. Let us call the technology needed to implement each of these three kinds of constructional elements *actor technology*, *coordination technology*, and *production technology*, respectively. The (only possible) actor technology is human beings, since they are the only 'things' that are able to enter into and comply with commitments. The engineering of actor roles usually consists of mapping them first to organizational functions and then to individual persons. The coordination technology, and production technology will be elaborated in the next section.
With reference to our OS (the sales information system), the ontological model from which engineering starts is the ontological model of the corresponding part of the I-organization, as shown in Figure 4.6.
In the *software engineering* process, as depicted in Figure 4.6, it appears to be most convenient to transform the ontological model of the I-organization (according to the Ψ-theory) into an ontological model according to the SMART meta model [16]. This meta model is a (formalized) simplification of the Ψ-theory for non-social actors, i.e. information processors. Software engineering thus basically consists of producing a series of subsequently more detailed white-box models up to the implementation model. If the technology to be applied is ICT, then the implementation model is a computer program in a programming language that can be compiled or interpreted by the ICT platform on which the sales information system is going to be operational.

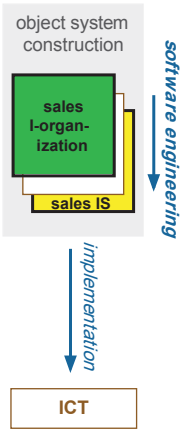


Figure 4.6 Depiction of software engineering

4.3 The System Development Process

Combining Figure 4.2, Figure 4.4, and Figure 4.5 results into Figure 4.7, which we will refer to as the *system development process*. As we have seen, the process starts from the construction of the US.

Ideally, it starts from the ontological model of the US, because only this model provides the appropriate help in guaranteeing that the requirements are necessary (all requirements are elicited) and sufficient (redundant or unreasonable requirements are eliminated). If this ontological model is unavailable, it may be (re) produced through reverse engineering. This activity would then be added to the development process. The rest of the process has been discussed extensively in the previous sections.

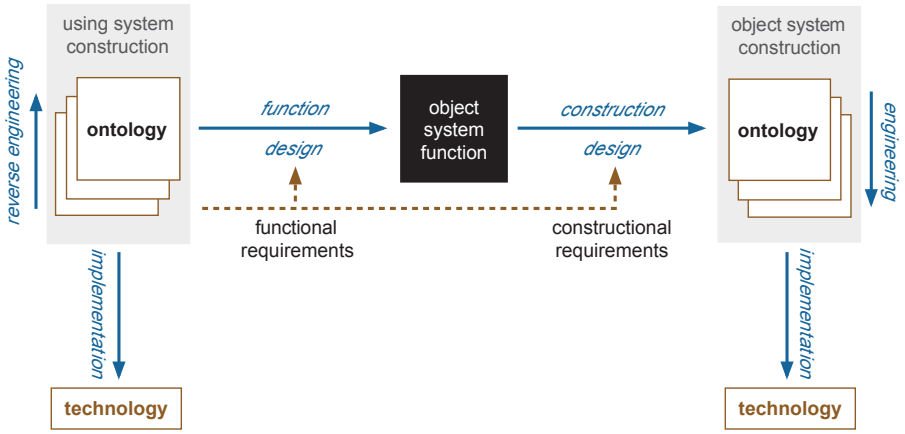


Figure 4.7 The system development process

Since enterprise ontologies are hardly applied in practice, the first step in the development of our sales information system would be producing the ontological model of the sales department on the basis of one or more lower level constructional models. Candidates for such models are Flow Charts or similar models (like EPCs and Petri Nets). A reliable methodology to produce an enterprise ontology in this way is DEMO [17]. Although it will rarely occur, one can imagine the case where a brand new enterprise has to be created. From the development process of Figure 4.7, this means that the OS is the enterprise to be designed and engineered. The US is the commercial environment in which the enterprise is going to be operational, and the functional model of the OS contains the services that the enterprise will deliver to the market.

In Chapter 3 the realization of an organization has been discussed. Let us now investigate the relationship between realization and implementation, taking Figure 4.8 as the point of departure. The upper part of the picture is an adapted version of Figure 3.7. The bars of the aspect organizations are stepwise shortened in order to let them fit the three types of ICT applications distinguished that are pictured in the lower part, and that accordingly are stepwise lengthened. As will be clarified, the shortening and lengthening of the bars is not only a matter of graphical necessity but have additional meanings too. The figure also shows the hardware on which these applications run. The color for hardware is gray and the color for software is yellow. As was done for the aspect organizations, the distinction between function and construction is indicated.

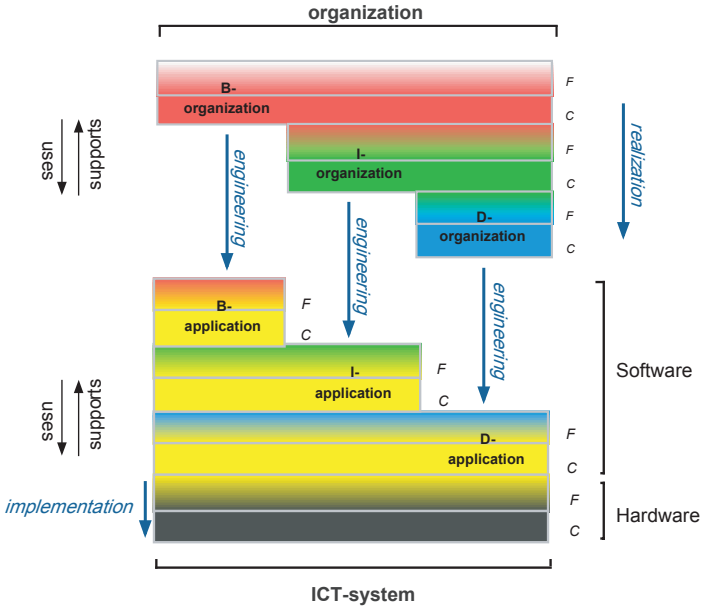


Figure 4.8 Organization and ICT system

Starting from the bottom, Figure 4.8 exhibits that the construction of the D-applications runs on the function of the hardware. By this class of software is understood all generic, i.e., not enterprise specific, software. Examples are text processors, as well as processors of other kinds of information, operating systems, and data base management systems. The D-applications may directly take over tasks of the D-organization (indicated by the fact that the I-application bars are shorter than the D-application bars).

The I-applications are put on top of the D-applications to express that they typically will not run directly on hardware but through the intermediate layer of D-applications. I-applications are by definition enterprise specific, although they may have a generic character, like accounting systems, human resource applications, and ERP (Enterprise Resource Planning) systems. Typically, I-applications can be called Management Information Systems. They are information systems in the true sense, i.e., they provide only information for monitoring an enterprise. Typical B-applications are Decision Support Systems. They are on top of the I-applications for the same reason as the I-applications are on top of the D-applications: they will commonly make use of the I-applications. And, for the same reason as the bars of the I-applications are shorter than the bars of the D-applications, the bars of the B-applications are shorter than those of the I-applications.

Up to now, we considered only ICT applications that support production acts, in the D-organization, the I-organization, or the B-organization. But of course, they can also support the performing of coordination acts. Since there is no distinction between the coordination acts in these three aspect organizations, the ICT applications that support coordination must be rather general. They are indeed but nevertheless they are usually referred to by different names. The D-applications are commonly referred to as network systems or e-mail systems. The I-applications are usually referred to as Work Flow Systems, in order to distinguish them from Business Process Management Systems that primarily support the coordination in the B-organization.

The question to be addressed next is to what extent ICT applications (or rather ICT systems) can replace parts of organizations. Put differently, to what extent can other technologies be used for the implementation of an organization, rather than the human actor role technology, the human production technology, and the human coordination technology? Let us start with *coordination*. In explaining the process of performing a coordination act (Figure 3.5), we did not mention the possible role of ICT. This was done on purpose, in order to deal with it later in a comprehensive way. The physical exchanges can very well be implemented by means of digital (electronic or optical) devices, generally referred to as hardware. Accordingly, the formative exchanges can very well be implemented by digital network services such as e-mail. These belong to the D-applications, as shown in Figure 4.8.

The next level becomes a little harder. Implementing informative exchanges with ICT means that one attributes to ICT systems the informa ability, i.e., to establish intellectual understanding. As is well known in Computer Science, this is actually not possible but that one can get around the problem by mimicking the informa ability. This is exactly what is done when making information systems (I-applications) interoperable. By using formal

languages instead of natural languages, this can quite easily be achieved. Needless to say that these cooperating I-applications have become artificially intelligent only, not truly intelligent of course.

The next level is the level of the performative exchanges. Would it be possible to implement them with ICT? Could machines be attributed the performa ability? Could they be able to enter into and comply with commitments? Could one hold machines responsible, in the real social sense? Although some Artificial Intelligence researchers are confident about eventually positive answers to these questions, our answers are definitely negative. The only trick one can think of is to get around the problem in the same manner as for informative exchanges. Then there would no longer be any distinction between the two levels. Well, this is exactly what is happening in modern integrated order and supply applications, such as automated stock replenishment systems, which are part of every ERP system. The consequence is a lot of confusion about responsibilities, data ownerships, etc.

Let us turn now to *production*. What role could ICT play here? It depends, of course, on the nature of the products. In the D-organization, the production acts are the storing, transporting, copying, etc., of documents. Obviously, if the documents are digital, the problem is solved. There is a plethora of D-applications that do the job: consider database management systems, file transfer systems, etc. After all, these were the very first kinds of use of programmed computers. In the I-organization, the production acts are the computing or deducing of new knowledge from existing knowledge. As we have seen when discussing the informative level in coordination, the availability of formal languages, in particular, of algorithmic languages, has solved the problem. I-applications just mimic the patterns of reasoning that the human mind applies.

The production acts in the B-organization offer the next problem. As we have said when discussing the implementation of performative exchanges with ICT, our point of view is that it is principally impossible to have machines perform original actions, like making decisions or judgments. However, many other people think differently about it, and consider its possibility only a matter of time. Often, one refers to the success of chess programs but that is fooling oneself, of course. Chess programs do not produce anything original, they only do what computers were meant for: they compute, albeit in a very sophisticated way. Clearly, chess programs do belong with the I-applications.

Let us turn our attention now to the *actor roles*. What role could ICT play in implementing actor roles? In the previous sections, we spoke only of subjects fulfilling actor roles, choosing more or less implicitly and exclusively, to use human beings as the actor role technology. There was an important reason for doing this, of course, which is that only human beings can enter into and comply with commitments, and that only human beings can be held responsible for their deeds. In dealing with coordination acts and production acts, we have concluded already that the limit to applying ICT lies at the ontological level (the performa ability). What does this mean for applying ICT to the implementation of actor roles? Well, obviously, it means that actor roles can never be taken over completely by ICT systems because of the performative exchanges in which they inevitably participate,

and because of the ontological production acts they have to perform (if the B-organization is considered).

Therefore, regarding the implementation of D-actor roles, the conclusion is that almost all of their acts can be taken over by agents (ICT systems), except for the performa ability needed in coordination. In a practical situation, this would mean that one can freely make use of databases and networks but it must always be clear who is eventually and really responsible. In other words, no actor role in the D-enterprise can ever be completely automated away. These roles must always be assigned to subjects. However, the effort that these subjects have to make in fulfilling the role is reduced to a minimum because of the maximum utilization of ICT in implementing those parts of the actor role that are suited for it. For example, in a modern enterprise, the internal post service and the archive are replaced by an e-mail service on the intranet and a document management system, respectively. Ideally, they are even fully integrated. However, there is, or should be, someone responsible for the well functioning of the e-mail service, and there is also someone responsible for the functioning of the document management system.

A similar conclusion can be drawn for I-actor roles. They must be assigned to subjects but most of the work can be taken over by ICT applications, and there must always be someone responsible for the functioning of these I-applications. With respect to B-actor roles, the conclusion is, and can only be, that they cannot be implemented with ICT, only with human resource technology[3]. But what about the B-applications in Figure 4.8? Well, they are precisely what the figure displays: B-organization supporting ICT applications, with decision support systems as the typical example of such applications. B-applications, in the sense that they take over production acts or performative coordination acts, cannot really exist. Of course, there are numerous ICT applications that very much seem to be 'real' B-applications, like Business Process Management Systems and ERP systems. They are just I-applications but we will attribute to them the separate status of B-application in order to distinguish them from the pure (I-organization supporting) I-applications. The only difference is that the results of their calculations are taken as 'real' decisions or judgments. This is acceptable as long as it is fully clear who is responsible for making these decisions or judgments.

To illustrate the foregoing, Figure 4.9 shows the projection of the relevant parts of our example (sales department and sales information system) on the picture of Figure 4.8.

---

[3] Since the introduction of the term "human resource", I have disliked it. It conveys the low opinion of human beings that could only emerge in the era of the industrial revolution. However, it is so widely spread that I adhere to the common practice.

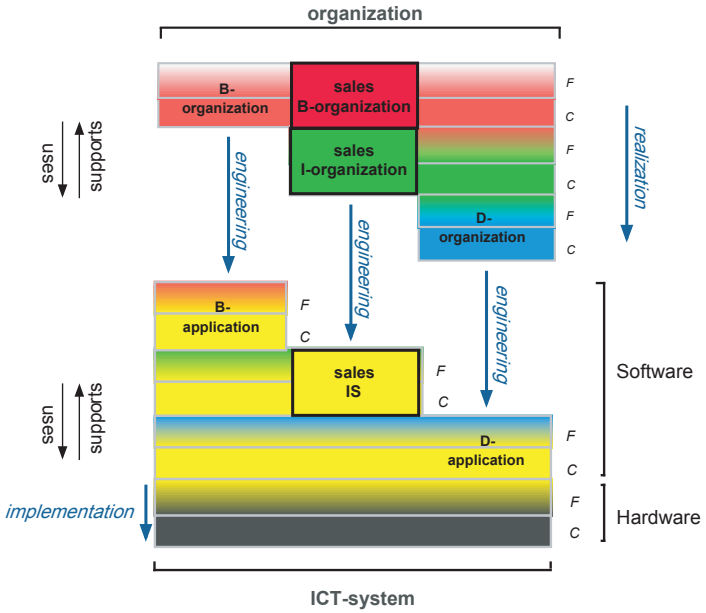


Figure 4.9 Engineering the sales information system (1)

The red box, labeled "sales B-organization", represents the B-organization of the sales department. The green box, labeled "sales I-organization", represents the part of the I-organization that supports the sales B-organization, and the yellow box, labeled "sales IS", represents the (I-application) software of the sales information system. It can run on a platform that includes the needed D-applications, like an interpreter, a DBMS, and an operating system.

Figure 4.10 exhibits how the three systems can be projected on the system development process of Figure 4.7. Note that the three boxes are not fully identical to their counterparts in Figure 4.9. The difference is that now only the construction perspective is shown. To be more precise, the red box, labeled "sales B-organization", represents the ontological model of the sales B-organization, and the green box, labeled "sales I-organization", represents the ontological model of the part of the sales I-organization that supports the sales B-organization. The yellow box, labeled "sales IS", represents the (I-application) source code of the sales information system.

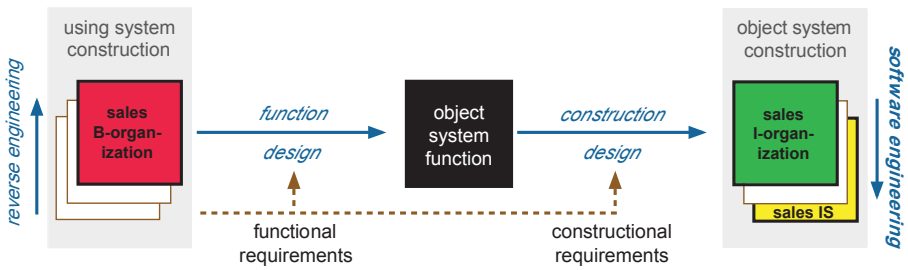


Figure 4.10 Engineering the sales information system (2)

5 The Notion of Architecture

At last, we can start to introduce and discuss the main subject of the book: the notion of architecture. It will be added to the other crucial notions in system development, i.e., ontology and technology, and it will get its place in the system development process. The result is a truly generic system development process (GSDP). We will apply it to the development of the B-organization, the I-organization, and the D-organization of an enterprise. Accordingly, three kinds of architectures are distinguished: business architecture, application architecture, and technical architecture.
Next, the process of devising architectures will be discussed, referred to as "architecturing", in order to avoid confusion with the term "architecting" as applied amongst others by [44], which has a very different meaning.

5.1 The current understanding of architecture

The second intellectual tool for mastering the complexity of contemporary enterprises, next to Enterprise Ontology, is Enterprise Architecture. Contrary to Enterprise Ontology, it is already extensively discussed in the literature about managing organizational change. Unfortunately, the term "Enterprise Architecture" has many meanings as well, meanings that are quite diverse and sometimes even contradictory. In addition, the acceptance and application of Enterprise Architecture differs largely. Gartner reports [22]:

> People talk of enterprise architecture as if it is easily understood by technology and business professionals alike. In reality, technology professionals have a wide-ranging view of enterprise architecture … In contrast, business professionals tend to ignore the term as an IT-only issue.

Before presenting and discussing, what we think should be the meaning of the term "Enterprise Architecture" and of "architecture" in general, we will review some apparently

influential definitions of the term that go around briefly and try to identify differences and commonalities between them. Probably the most influential definition of (Enterprise) Architecture is the one provided by Zachman [54]. It reads:

> Architecture is that set of design artifacts, or descriptive representations, that are relevant for describing an object, such that it can be produced to requirements as well as maintained over the period of its useful life.

There are two interesting observations to be made regarding this definition. Firstly, architecture is not conceived as a property of an object but rather as a document or a set of documents describing the object. This is quite remarkable since its more general and colloquial meaning, emerged from the field of constructional engineering, definitely includes that it is a property of an object. For example, you can say that a church is a roman church or a gothic one (you would not say that the drawing of the church is roman or gothic). Secondly, the architecture of an object apparently contains the information that is needed to produce the object and to maintain it. The question that comes to the mind then is in what sense or to what degree this notion of architecture differs from the notion of design. The referred paper does not provide an answer to this question, unfortunately, which leads us to the opinion that Zachman's notion of architecture is at best, and with a lot of imagination, akin to another influential definition, namely IEEE Standard 1471 [39]. It reads:

> Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

In our view, this definition is ambiguous; it tries to accommodate two very different points of view. One is architecture as the global composition and structure of a system, abstracted from details. The other one is architecture as a restriction of its design and engineering freedom. IEEE Standard 1471 is quite similar to the second meaning assigned to the term "architecture" by the Open Group [49], whereas the first one of the Open Group's definition resembles Zachman's definition largely. The definition of architecture by the Open Group reads:

> 1. A formal description of a system or a detailed plan of the system at component level to guide its implementation.
> 2. The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

Many more well-known authors could be cited but this does not offer really new insights. So, let us start commenting on the definitions cited above. Firstly, we cogently reject Zachman's definition and the first meaning of the Open Group. Both sources apparently just fail to distinguish between a concept (or better: conceptual system) and the way it is represented, whether in plain text or diagrams or any other language (cf. Figure 2.2). This leaves us with the second meaning by the Open Group and with IEEE's. As said, they are ambiguous: they contain two different definitions in one sentence, something that should be avoided in any definition. Regarding the first part, we note that this idea is already contained (and in a much more effective and precise way) in the notion of system ontology, as discussed in Chapter 3. Obviously, we agree with the cited sources as well as with many other authoritative ones, like [40], that understanding the construction and operation of a system in a fundamental way, abstracted from implementation details, is crucial. However, it has already a name: system ontology. So, let us use the term "architecture" exclusively for the second part, following e.g., [31]. Concluding, we define architecture in the next way:

> Theoretically, architecture is the normative restriction of design freedom.
> Practically, architecture is a consistent and coherent set of design principles.

We have to admit that the application of this notion of architecture in practice is just emerging. Companies that have adopted it are still in the phase of understanding what it means; they are the pioneers. Yet, the potentials are enormous [19]. In fact, it seems to be the only feasible way of 'translating' high-level statements, as can be found in mission and strategy documents, into operationally useful principles for developing systems. As a more or less logical consequence of the previous discussion, architecture holds for all systems of a particular kind, thus for a *class* of systems.

5.2 Architecture in the GSDP

Our notion of architecture appears to fit nicely into the system development process that was presented in Chapter 4. The complete process is called Generic System Development Process (GSDP); it is shown in Figure 5.1. In line with the distinction between function and construction in this conceptual framework, we distinguish between *functional principles* or function architecture, and *constructional principles* or construction architecture. These principles embody experiences as well as otherwise based policies of the supplier of the object system. This is the basic rationale for applying the notion of architecture: *building strategy into design*. Note once more that supplier is an actor role. This role may be fulfilled by another enterprise than the customer, e.g. in case a company buys an ERP system from a vendor for its own internal use but it may as well be fulfilled by a part of the customer organization, e.g. if a major organizational change is brought about in a project performed by one of its own departments.

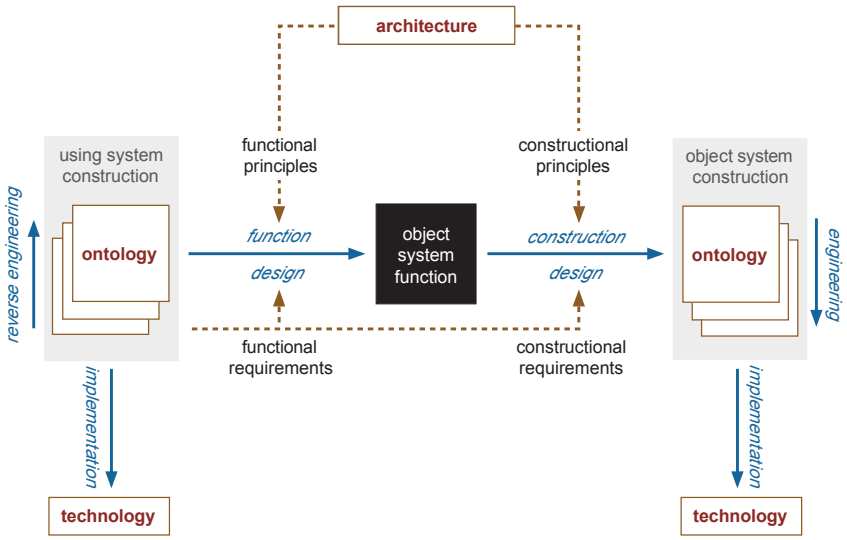


Figure 5.1 The Generic System Development Process

Functional principles guide the function design and thus come in addition to functional requirements as input for the function design phase. Constructional principles guide the construction design (as well as the engineering) and come in addition to constructional requirements as input for the construction design phase (and engineering). Both architectures become 'present' in the resulting object system, however indirectly, as a kind of 'signature'. To exemplify this, everyone recognizes whether a church is a roman church or a gothic one, or whether a temple pillar is an Ionic or a Doric one. Likewise, one recognizes a painting as a Van Gogh or a Rembrandt. This is the right understanding of architecture (as it has always been understood by constructional architects!). To express the idea in one sentence:

> Architecture is not what you see but what shaped what you see.

Through this notion of architecture suppliers (or better: producers) of systems are able to distinguish themselves from others. Among other things, this gives rise to the distinction between 'good' and 'bad' architectures. Good function architecture immediately reveals the function of a product. An example of a product with a 'good' function architecture is the Apple MacOS; an example of a product with a 'bad' function architecture is the (first) video recorder. Good construction architecture immediately reveals the construction of a product. An example of 'good' construction architecture is the modern PC, whereas 'bad' construction architecture can be found in unstructured ('spaghetti') computer programs.

These examples also clarify the difference between system requirements and design principles. The functional requirements for Windows are roughly similar to those for the Mac-OS. Therefore, the differences between these operating systems are largely due to the differences in the applied functional architectures. Likewise, the constructional requirements for a 'spaghetti' computer program are roughly similar to those for a well-structured program. The differences between the two programs are largely due to the differences in the applied constructional architectures.

5.3 Enterprise Architecture

By *enterprise architecture* is understood the whole set of design principles that an enterprise applies in (re-) designing itself, basically in all its aspects. In accordance with the distinction between the three aspect organizations of an enterprise (see Chapter 3), we partition enterprise architecture into three architectures: the business architecture, the application architecture, and the technical architecture[1]. They apply respectively to the (re-) design of the B-organization, the I-organization, and the D-organization. Consequently, the business architecture of an enterprise comprises the functional and the constructional principles that are applicable to the (re-) design of its B-organization. Although this seems contradictory to the fact that an architecture applies to a class of systems, it is not. The class in this case is the class of all possible B-organizations of the same enterprise. At any point in time, there is only one B-organization in place. In the course of time, there may be many, however, because of the re-designing of (the current) B-organization. A similar reasoning holds for the application architecture and the technical architecture. Figure 5.2 exhibits the role of *business architecture* in the process of (re-) designing the B-organization. Since the object system is the B-organization, the using system must be the market to which the enterprise offers its services. In the figure, it is assumed that this market is a business market, so the collection of enterprises with which the object system will interact. The red colored box, labeled "B-organization", represents the ontological model of the B-organization. Its implementation model will commonly consist of the organizational chart or organization chart (HR in Figure 5.2), the administrative organization (AO), etc. The main implementation technology is people, as discussed in the previous chapters.

Examples of functional requirements in Figure 5.2 are the services that have to be delivered to the market and the expected growth figures. Examples of constructional requirements are the volume of these services per day and the maximum delivery times.

Examples of functional principles in Figure 5.2 are a high level of customer intimacy and the embargo on trading with particular nations. Examples of constructional principles are limited size business units and substantively expert managers.

Figure 5.3 exhibits the role of *application architecture* in the process of (re-) designing the I-organization of an enterprise. Now the using system is the B-organization; the red colored box, labeled "B-organization" represents its ontological model. The green colored box, la-

[1] With these rather imprecise names, we try to conform to current terminology.

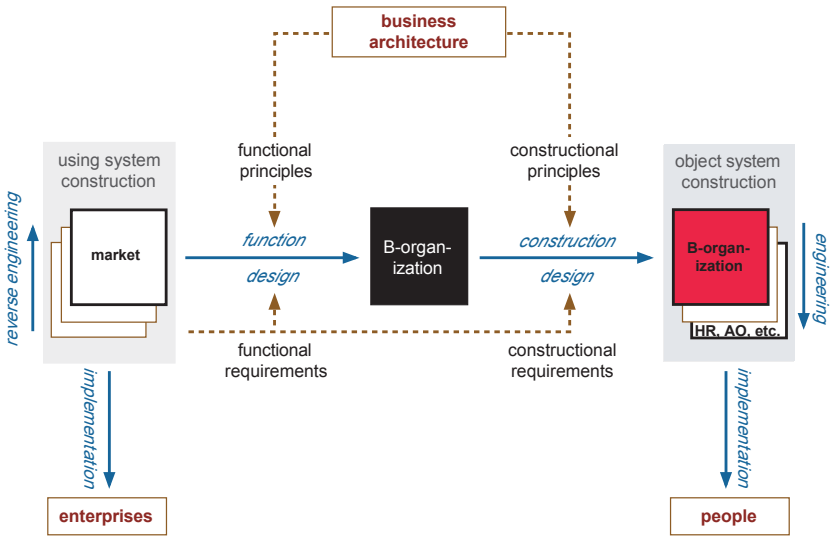


Figure 5.2 Business Architecture

beled "I-organization" represents the ontological model of the I-organization, which is going to support the B-organization. The implementation model of this I-organization could be such that people would be the appropriate technology, like it is for the B-organization. Instead, however, we have chosen for 'automating' the I-organization. Consequently, the implementation model is the source code of the needed I-applications (cf. Figure 4.9). These applications are implemented in software.

Examples of functional requirements in Figure 5.3 are the information services that the I-applications must provide and actor role based access rights. Examples of constructional requirements are low response times and the requirement that the services are web-based.

Examples of functional principles in Figure 5.3 are standardized dialogs and multi-lingual interfaces. Examples of constructional principles are the adoption of Service Oriented Architecture and the adoption of Client-Server Architecture with thin clients.

Figure 5.4 exhibits the role of *technical architecture* in the process of (re-) designing the D-organization of an enterprise. Now the using system is the I-organization; the green colored box, labeled "I-organization" represents its ontological model. The blue colored box, labeled "D-organization" represents the ontological model of the D-organization that is going to support the I-organization. The implementation model of this D-organization could be such that people would be the appropriate technology, like it is for the B-organization. However, since we have decided to 'automate' the I-organization, we consequently decide to 'automate' the D-organization too. The D-applications are implemented in ICT systems (including software and hardware).

Examples of functional requirements in Figure 5.4 are the infrastructural services that the D-applications must provide and the prohibition of duplicate data storage. Examples of constructional requirements are high security levels and high availability of the services.

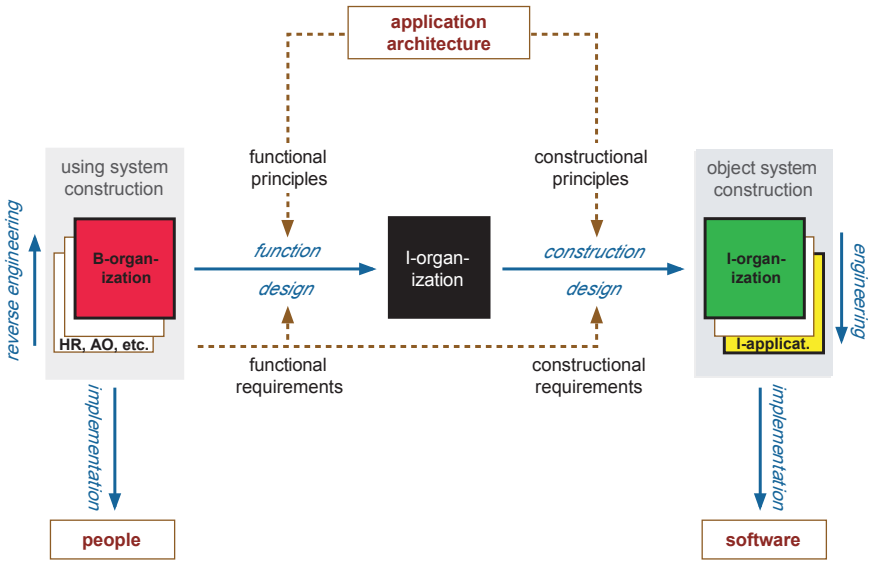


Figure 5.3 Application Architecture

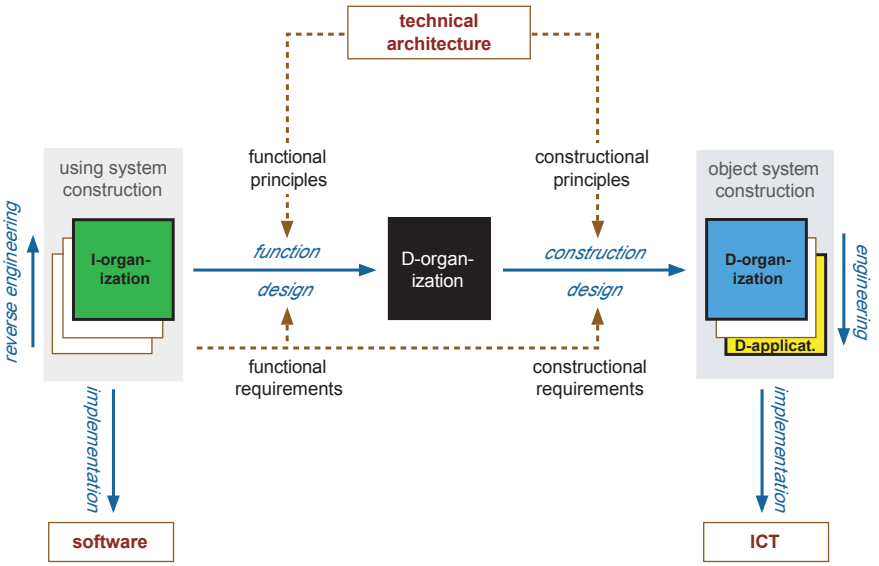


Figure 5.4 Technical Architecture

Examples of functional principles in Figure 5.4 are standard communication protocols and standard document descriptions. Examples of constructional principles are relational DBMSs and the adoption of Service Oriented Architecture.

5.4 Architecturing

Devising architectures can appropriately be labeled *architecturing*, as devising designs is called designing. Consequently, it seems plausible to call the architecturing person an *architect*. Trying to narrow down the current understanding of the word to this one seems to be a job for Don Quichote, however. So, we will not attempt to do it. A warning concerning the term "architecting" seems to be relevant. Apparently, the meaning of this term, as conveyed in the literature [40] has little to do with our notion of architecturing. Actually, it is rather similar to designing; so nothing new.

Because of being fundamentally different, designing and architecturing play a different role in the system development process. Architecturing is an autonomous activity that can be performed independently from developing and implementing systems, like the (operational) management of systems goes independently from system development and implementation. Figure 5.5 tries to illustrate the relationships between the four major activities in a system's life cycle, each of them represented by its own cycle.

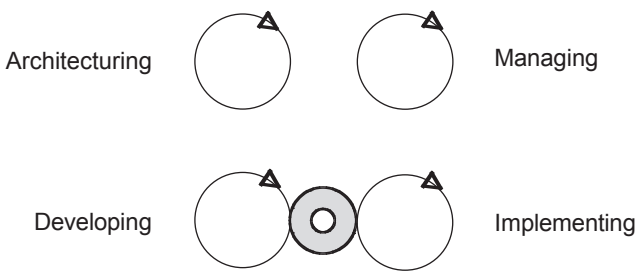


Figure 5.5 Relationships between the activities

Developing and implementing always regard one system, whereas architecturing and managing regard several systems at a time. Developing and implementing also go synchronously, i.e., implementing a system is connected to developing it. The grey 'wheel' between developing and implementing represents this strong coupling. In the traditional system life cycle, implementing comes after developing. In modern life cycle models, like RAD and prototyping, they overlap.

As said, architecturing and managing regard several systems at a time. In addition, they go asynchronously, not only with respect to each other but also with respect to developing and implementing.

In the next chapter, we will discuss the distinct design domains and areas of concern that architecturing deals with, next to the distinct kinds of system, which are for enterprise architecture, as we have seen, the B-organization, the I-organization, and the D-organization, or their 'automated' versions. The remainder of the current chapter will be spent on identifying and specifying design principles.

Before starting architecturing in an enterprise, it is crucial to verify the current level of architecture awareness. By this we mean the degree in which architectural thinking has become common property to the employees (including the managers) of the enterprise.

The minimum level of awareness is that the participants in an architecture project are motivated to do the work, and that the responsible managers are fully supporting it. Unfortunately, the level of architecture awareness anno 2008 in The Netherlands is pretty low, although higher than most other nations of the western world. Put differently, what this book is pleading for is hardly common practice; we cannot point to great successes (neither to great failures). Therefore, we can only present and discuss ideas and approaches that seem to go to work [27].

The best way to create or increase architecture awareness is to illustrate architecturing by simple examples, preferably from outside the business of the enterprise, in order to avoid discussions about the correctness of the example itself. Once that the idea of architecture and of architecturing has come across in the minds of the people can one take examples from within the enterprise in order to let all noses point in the same direction. The hardest part in architecturing appears to be to maintain coherence and consistency in the set of principles that one is devising. Basically, these properties can only be achieved if there is a common root from which all principles stem, and with which they are consistent. This common root is the mission of the enterprise and the strategies that are already formulated on the basis of this mission. One can imagine that it is quite a job to translate mission statements in operational design principles. It is one thing to say that "our enterprise strives to be the best of its sort" but quite something else to achieve it. One thing is sure by now and by itself already of great value: one can only intellectually manage this enormous task by applying the notion of architecture that is presented in this book. There is no other way. At the same time, there is also no escape: future enterprises will be required to be designed 'under architecture'. It is not unlikely that this requirement once becomes a quality standard.

To exemplify the aforementioned let us have a look at two pizzerias which share the same vision but which have formulated different strategies and from there different design principles [27]. One of the pizzerias is owned by Mario, the other one by Luigi. The shared vision reads:

> I want to be the most renowned pizzeria in my region.

Mario has devised the next two strategies for reaching this goal:

> Products must be designed such that they can be customized by the customer.
> The revenue model must be based on lifetime customer value.

Luigi, however, has devised quite different strategies to achieve the same goal:

> Products must be designed such that they can be produced as cheaply as possible.
> The revenue model must be based on maximizing the sales quantity.

For both pizzerias, these strategies already can be considered as business design principles, thus as elements of the business architecture but which are not yet sufficiently specific for being operationally useful. Confronted with this requirement, Mario and Luigi have made their strategies more specific. Mario has done this as follows:

> The customer can choose the ingredients for a pizza himself.
> The more pizzas customers buy, the more discount they will receive.

Contrary to Mario's refinements, Luigi has produced the next more specific principles:

> Only a standard set of the most popular pizzas is offered.
> Pizzas are offered at the lowest possible price.

The difference between Mario and Luigi is quite clear. Mario considers customer intimacy (and seemingly product innovation as well) of paramount importance, whereas Luigi strives for the highest turnover at the lowest costs, to be achieved by operational excellence.
All of the design principles mentioned are functional principles within the business architecture of the pizzerias. The constructional principles have to be added yet, as well as the application architecture and the technical architecture. However, we will not elaborate on them.
The principles above are indeed much more specific than the strategies they are derived from but they still leave room for questions. The ultimate test for determining that the specificity of the principles is sufficient is whether they can easily be translated to unambiguous *operational rules*. Note that operational rules, including *business rules*, are not similar to design principles, the fundamental difference being that design principles guide the design of a system (enterprise), whereas operational rules guide the system's (enterprise's) operations [20]. Nevertheless, they are related since architectural principles are input for the process of designing and engineering a system, next to the requirements (cf. Figure 5.1). Operational rules, and thus business rules, need to be unambiguous because they define the state space and the process space of a system's world (cf. Chapter 2 and Chapter 3). Our hypothesis is that the validity of operational rules can only be determined by demonstrating that they make one or more lowest level, thus most specific, design principles (or requirements) operational. At the same time, this hypothesis could serve as a criterion for the specificity of design principles at the lowest level. With these considerations, we end the chapter on the notion of architecture.

6 The notion of Architecture Framework

In the previous chapter, the notion of architecture has been discussed, to the extent that is considered sensible, given the current immature state of the art. As a support for devising architectures, it is convenient to have a guiding framework or structured checklist of issues that must be taken into account. Such a structured checklist is called an *architecture framework*. This notion is the subject of the current chapter. After having introduced the dimensions that an architecture framework needs to include, we present the idea of the Extensible Architecture Framework (xAF). Next, this xAF will be compared to a well-known framework in practice, namely the Integrated Architecture Framework (IAF), as described in [26].

6.1 Architectural dimensions

There exist many architecture frameworks, or at least things that are called by that name, of which the most well known are listed in [46]. From this statement of affairs, it becomes immediately clear that the label "architecture framework" is used for rather different things. The most striking observation one has to make, unavoidably, is that most of these frameworks are development methodologies 'in disguise'. This is rather unfortunate of course but a fact of life. It is all caused by the architecture hype that started around 1990. As we have disassociated ourselves from the current, ambiguous and unnecessary, notion of architecture being a kind of global design or blueprint, we will likewise disassociate ourselves from this current notion of architecture framework. And for similar reasons: they are ambiguous, incoherent, inconsistent, and above all unnecessary. They offer nothing new; as said before, these architecture frameworks are refurbished development methodologies. What we will do instead is think up what an architecture framework should be like, in order to achieve the objective of supporting the process of architecturing, thus the devising and formulating of design principles. It seems that three dimensions are necessary and sufficient to consider. The first one is the *system kind* that a design principle applies to. Examples of system kinds in our main area of interest are organizations, information systems, and software systems. The second dimension is the *design domain* to which a de-

sign principle belongs. Building on the systemic foundations, as discussed in Chapter 2, we distinguish between two domains: function and construction. The third dimension we distinguish is the *area of concern* to which a design principle belongs. Whereas the first two dimensions are timeless, the third one is not. It reflects the, generally time-dependent, focal interests of stakeholders. Examples of areas of concern are security, compliance, privacy, agility, and user-friendliness. They serve to organize the total set of design principles of an architecture into subsets that correspond to the interests of the distinct stakeholders. These subsets need not be disjoint. Put differently, areas of concern may overlap. It means that a particular design principle regards several interests and therefore belongs to several areas of concern. As an example, taken from Mario's pizzeria (cf. Chapter 5), the design principle

> The customer can choose the ingredients for a pizza himself

may belong to the area of concern "customer intimacy" as well as to "operational flexibility".

We are now able to provide a more precise definition of the notions of design principle and of architecture. A *design principle* regards one system kind and one design domain. It addresses one or more areas of concern. An *architecture* is a set of design principles. It may regard several system kinds as well as several design domains. A 'minimal' architecture regards one system kind and one design domain. When considering an entire enterprise, and with reference to Figure 5.2, Figure 5.3 and Figure 5.4, one may talk for example of business function architecture, of application construction architecture, and of technical function architecture.

Formally, an *architecture framework* can be defined as a tuple <S,D,A> where:

- S is a set of *system* kinds.
- D is a set of design *domains*.
- A is a set of *areas* of concern.

As said earlier, current architecture frameworks do not satisfy the definition above. More fundamentally, one must observe that current architecture frameworks do not apply our normative notion of architecture. The absence of a formal, normative approach also implies the absence of formal design guidance. So, the important objectives that the normative approach tries to establish are not addressed. This is not only detrimental to the design process itself but also to professionalizing the architecturing competence in an enterprise and the architect's profession.

Finally, as we have observed, development, implementation and project related (planning) issues are frequently part of architecture frameworks. Evidently, these aspects have to be addressed professionally but they fall outside the scope of architecture and architecturing, hence should not be part of an architecture framework. Much of what is positioned as an architecture framework is in fact a concealed development and implementation methodology.

6.2 The Extensible Architecture Framework

The formal definition of an architecture framework, as provided above, suggests that it must be feasible to define an architecture framework as an extension of one or more existing architecture frameworks, while being also extensible itself. This would offer the desirable option to compare and evaluate existing frameworks. To realize this idea of an *Extensible Architecture Framework* (xAF), two things are needed. Firstly one must have a *generic architecture framework* that can serve as the *universal root xAF* of all other xAFs. Let us call it xAF_0. Secondly, one needs *extension rules* for defining xAFs as extensions of one or more existing xAFs, and thus ultimately as extensions of the xAF_0. Two extension rules seem to be sufficient for defining a new xAF on the basis of one or more other xAFs: the *specialization* rule and the *integration* rule. These rules guarantee compliance of a new architecture with its direct parent(s) and thus with the xAF_0. In the next subsections, formal definitions of the rules are provided.

6.2.1 The specialization rule

An xAF_j, defined as $<S_j,D_j,A_j>$ is a *specialization* of an xAF_i, defined as $<S_i,D_i,A_i>$ if and only if:

- Every system kind $s \in Sj$ is an exclusive subtype of some $s' \in Si$.
- Dj is a superset of Di.
- Aj is a superset of Ai.

The following explanation of this formal definition might be helpful. Firstly although S represents a set of system kinds, in most cases this set will be a singleton, so containing only one member. Secondly, by "exclusive subtype" is meant that the extension of the system kind *s* is a proper subset (symbol $\subset$) of the extension of *s'*. Put differently, *s* cannot be identical to *s'*. Thirdly, the notion of superset as used above is the inverse of the common notion of subset (symbol $\subseteq$), not the strict or proper one (symbol $\subset$). In other words, D_j may be larger than but it may also be identical to D_i, and A_j may be larger than but it may also be identical to A_i.

As an example for illustration, let us consider three extensible architecture frameworks: xAF_1, with the system kind *chair*, xAF_2, with the system kind *circus chair*, and xAF_3, with the system kind *TV chair* [5]. It looks as if xAF_2 and xAF_3 are specializations of xAF_1. Let us verify this, based on the formal specifications of the frameworks. For the sake of simplicity, we assume that xAF_1 has no areas of concern. Then the formal specification of xAF_1 would read:

$xAF_1 = <$ {chair}, {function, construction}, $\varnothing >$

Thus, S_1 contains only one system kind (chair), D_1 comprises two design domains (function and construction), and A_1 is empty. On the basis of this architecture framework one can devise architectures that regard chairs and either their function (the function architecture of chairs) or their construction (the construction architecture of chairs). A possible design principle in the function architecture could be:

> sitting on the chair should be comfortable for persons with a weight between 20 and 100 kg

whereas a possible design principle in the construction architecture is:

> only wood and synthetic materials may be used

One of the possible specializations of the system kind chair is the circus chair, of which Figure 6.1 exhibits an example. The architectural framework for circus chairs, which we called xAF_2, should cover the specific properties that these chairs should possess. Let us consider two areas of concern that seem relevant for circus chairs. One is *versatility* and the other one is *grip*. The formal specification of xAF_2 then would be:

> xAF_2 = < {circus chair}, {function, construction}, {grip, versatility} >

Figure 6.1 Picture of a circus chair

The framework xAF_2 appears to be a specialization of the framework xAF_1 because the following properties hold:

- *Circus chair* is an exclusive subtype of *chair*, because the extension of circus chair is a proper subset of the extension of chair (there are also chairs that are not a circus chair).
- D2 = D1, thus D2 is a superset of D1.
- A1 is a proper subset of A2 (recall that A1 is empty and A2 is not), thus A2 is a superset of A1.

A possible design principle in the function architecture of circus chairs, within the area of versatility, could be:

> it must be possible to perform all chair acts that are listed in the Acrobat Handbook 2008

whereas a possible design principle in the construction architecture is:

> a firm grip must be assured under normal humidity conditions (40% to 70%)

The other specialization of chair that we considered is the TV chair, of which Figure 6.2 exhibits an example. The architectural framework for TV chairs, which we called xAF_3, should cover the specific properties that these chairs should possess. Let us consider two areas of concern that seem relevant for TV chairs. One is *relaxation* and the other one is *styling*. The formal specification of xAF_3 then would be:

> xAF_3 = < {TV chair}, {function, construction}, {relaxation, styling} >

Figure 6.2 Picture of a TV chair

The framework xAF_3 appears to be a specialization of the framework xAF_1 because the following properties hold:

- *TV chair* is an exclusive subtype of *chair*, because the extension of circus chair is a proper subset of the extension of chair (there are also chairs that are not a TV chair).
- D3 = D1, thus D3 is a superset of D1.
- A1 is a proper subset of A3 (recall that A1 is empty and A3 is not), thus A3 is a superset of A1.

A possible design principle in the function architecture of TV chairs, within the area of relaxation, could be:

> sitting in the chair should relax all muscles in the lower part of the body

and another design principle in the function architecture, within the area of styling, could be:

> the appearance of the chair should be modern

The chair in Figure 6.2 looks like being in full conformity with this function architecture. At the same time, one can imagine that there are TV chairs with a classical styling instead of a modern one. These chairs would conform to a different architecture, i.e., at least the last design principle should be replaced by:

> the appearance of the chair should be classical

6.2.2 The integration rule

An xAF_n, defined as <S,D,A> is an *integration* of a set of xAFs, defined as $\{<S_1,D_1,A_1>,<S_2,D_2,A_2>, \ldots ,<S_k,D_k,A_k>\}$, if and only if:

- S is the integral union of S1, S2, ... , Sk.
- D is the integral union of D1, D2, ... , Dk.
- A is the integral union of A1, A2, ... , Ak.

By *integral union* is meant that the elements of the united set are rigorously related to each other, not just 'talked together'. Regarding the integration of systems, so the integral union of S_1, S_2, ... , S_k, these systems should constitute a layered nesting, as discussed in Chapter 3. An example of integration is devising an enterprise architecture framework as

the integration of several component frameworks, like a framework for the *organization*, a framework for the *information systems* and a framework for the *ICT infrastructure*. In the next section, we will elaborate on architecture frameworks that are constructed by means of integration.

6.2.3 The xAF lattice

Extending the xAF_0 yields a top-down directed *lattice* of xAFs. Figure 6.3 shows a possible lattice. A straight line represents *specialization*. For example, xAF_j is a specialization of xAF_i. A bundle of dotted lines represents *integration*. For example, xAF_n is the integration of xAF_1, xAF_2 and xAF_3.

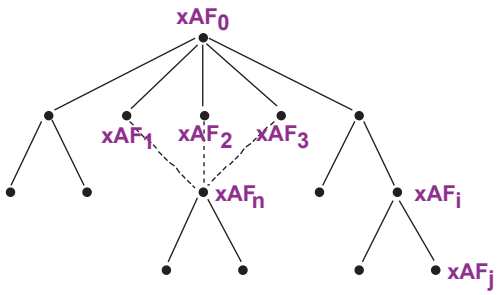


Figure 6.3 The lattice of architecture frameworks

The *system kind* in the root architecture framework, thus the xAF_0, is the general homogeneous system. Two *design domains* are identified in the xAF_0: the function domain and the construction domain. The set A of *areas of concern* in the xAF_0 is empty. This only means that there are no universal areas of concern. This is rather obvious since the xAF_0 has no stakeholders. It is the theoretical root of all architecture frameworks, not a practical one.

6.3 Comparing Architecture Frameworks

One of the motivations for the development of the xAF was to provide a basis for comparing architecture frameworks. This can be done as soon as the frameworks to be compared have got a place in the xAF lattice (see Section 6.2.3). The purpose of the current section is to show how this is done, taking the well-known IAF (Integrated Architecture Framework) as the exemplary framework. The source for our knowledge about the IAF is [26].

6.3.1 Mapping the IAF on the xAF

As the name suggests, the IAF seems to be constructed from several other frameworks by means of integration. So, one of our objectives will be to find out what these frameworks are and whether the construction satisfies the requirement of integral union (cf. Section 6.2.2). Figure 6.4 exhibits the main picture regarding the IAF; it shows the three dimen-

sions that are covered. They are called “Main Architecture Area”, “Design Approach”, and “Architectural Viewpoint”.

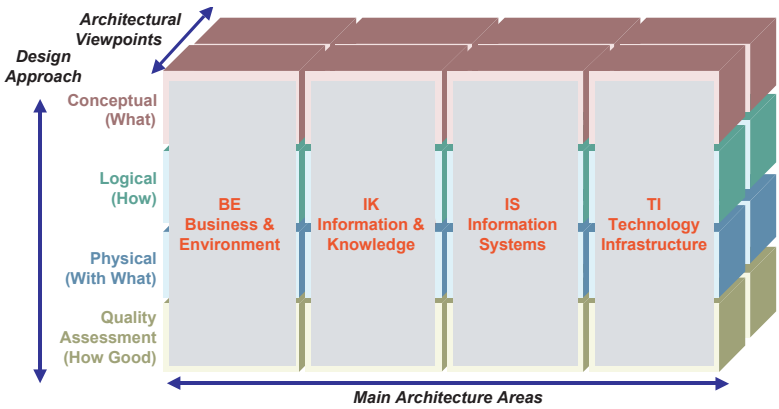


Figure 6.4 The dimensions of the IAF

Comparing these dimensions with the three dimensions of the xAF, it looks on first sight that they match to some extent. So, let us formulate these preliminary hypotheses:

> The dimension Architecture Area of the IAF corresponds to the dimension System Kind of the xAF.
>
> The dimension Architectural Viewpoint of the IAF corresponds to the dimension Areas of Concern of the xAF.
>
> The dimension Design Approach of the IAF corresponds to nothing in the xAF.

Let us investigate the first hypothesis first. The question thus is whether the distinguished Architecture Areas of the IAF may count as system kinds in the xAF. The distinct areas are: Business and Environment (BE), Information & Knowledge (IK), Information Systems (IS), and Technology Infrastructure (TI). Of these areas, the second one does not seem to be a system kind, or the name is not chosen appropriately. The other three could very well be system kinds. For answering the question, another picture from [26] might be helpful; it is shown in Figure 6.5. It presents all four areas as networks and it groups these networks into two system kinds: Business System and ICT System. The Business System is said to consist of the BE-network and the IK-network. In a similar way, the ICT System is said to consist of the IS-network and the TI-network. In addition, it is mentioned that the ICT system supports the Business System.

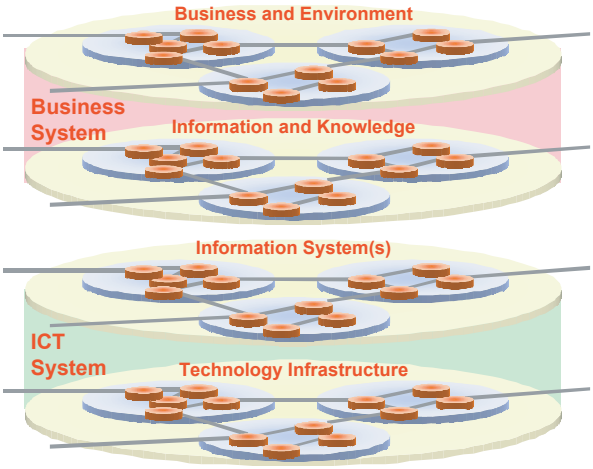


Figure 6.5 Main Architecture Areas in the IAF

Since the notion of network is not defined, we take an intuitive one, and proceed with an attempt to map the four Architecture Areas on the three aspect organizations, as discussed in Chapter 3, while taking into account the grouping of these areas in the two system kinds mentioned. The correspondences as shown in Figure 6.6 then force upon the mind.

Business & Environment	B-organization
Information & Knowledge	I-organization
Information Systems	D-organization
Technology Infrastructure	D-organization

Figure 6.6 First mapping of IAF on xAF

The correspondence between Business & Environment and B-organization is probably the least disputable, regardless the fact that in the IAF no explicit distinction is made between a system's function and its construction. The second line in the figure is more awkward, since we are not sure whether we can Information & Knowledge take as a system. Just doing it for the moment leads to the conclusion that this area then would best correspond to the I-organization of the enterprise under consideration. Concerning the third and the fourth area (Information Systems and Technology Infrastructure), we tend to think that they are respectively the function perspective and the construction perspective on the D-organization.

At the same time however, we are puzzled about the fact that [26] uses rather dominantly technical terms in discussing the areas of Information Systems and Technology Infrastructure. Let us therefore include in our investigation the implementation of the three aspect organization, as discussed in Chapter 4. Figure 6.7 summarizes the relationships between the three aspect organizations and the corresponding three kinds of ICT applications.

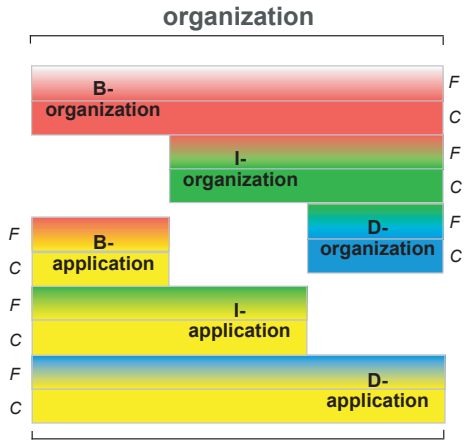


Figure 6.7 Organization and ICT application in xAF

While reconsidering the mapping that is presented in Figure 6.6, the mappings in the third and the fourth line appear to be wrong. It becomes quite clear from the explanations in [26] that the areas Information Systems and Technology Infrastructure concern only computer software and hardware, not people. This leads us to the improvement that is shown in Figure 6.8. These conclusions are reinforced by the enabling relationships that [26] discusses about and that are shown in Figure 6.9.

Business & Environment	B-organization
Information & Knowledge	I-organization
Information Systems	B-applications + I-applications
Technology Infrastructure	D-applications + hardware

Figure 6.8 Second mapping of IAF on xAF

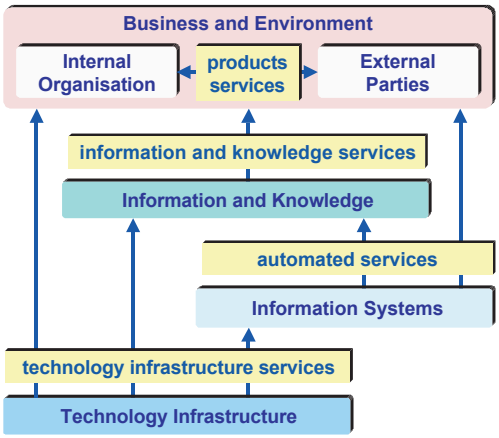


Figure 6.9 Enabling relationships between Architecture Areas

The second preliminary hypothesis we have formulated was that the Architectural Viewpoints of the IAF match the Areas of Concern of the xAF. The rationale for having this dimension in the architecture framework, as provided in [26], is that there are quality attributes, which surpass the border of one Architecture Area. Examples of these attributes are management and security. Since dealing with them adequately requires a broader scope than only one Architecture Area, the dimension Architectural Viewpoint is added to the IAF. Apparently, all quality attributes that can be fully covered in one Architecture Area, are not included in the Architectural Viewpoints.

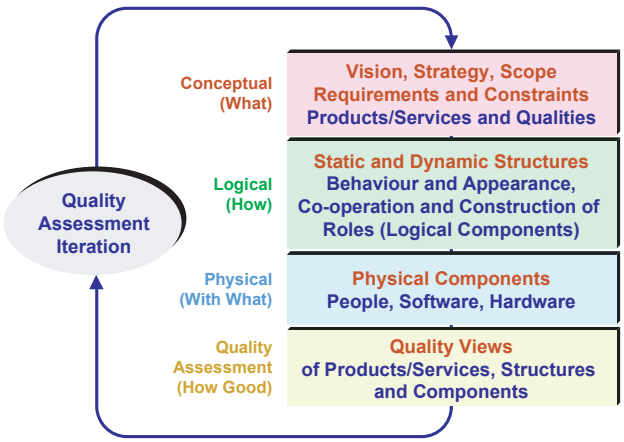


Figure 6.10 The design phases in the IAF

Leaving these minor points of discussion aside, it appears that the Architectural Viewpoints do match our areas of concern very well. That is to say, considering the intention to have relevant issues included in a structured way in the framework. As [26] does not speak of design principles, we are not allowed to conclude also that Architectural Viewpoints are groupings of design principles, like the areas of concern in the xAF are.
Lastly, it is hard to determine whether the integration of the four 'system kinds' (the Architecture Areas) of the IAF satisfies the integration rule, because of lack of information. Anyhow, a strong suspicion remains that the areas Information & Knowledge and Information Systems are not really different system kinds.

6.3.2 Mapping the xAF on the IAF

A fair comparison of the IAF and the xAF requires also that we try to map the xAF on the IAF in order to discover elements in the IAF that are missing in the xAF. A most prominent missing element is of course the dimension Design Approach of the IAF, as we found already in the first analysis. From the pertinent explanations in [26] it appears that this dimension is fully concerned with the phases in a development process. The four phases that are distinguished (cf. Figure 6.4) are: Conceptual (what), Logical (how), Physical (with what), and Quality Assessment (how good). The main questions addressed in each of the

phases are respectively "What ICT-enabled enterprise must be realized?", "How is the ICT-enabled enterprise realized?", "With what is the ICT-enabled enterprise realized?", and "How good is the ICT-enabled enterprise realized?". Firstly, this explanation contributes to the evidence that the IAF is pre-occupied with applying ICT to the implementation of enterprises. Secondly, and more important for our main discussion, this dimension Design Approach is absent in the xAF for very good reasons. It is not about architecture (as we conceive it) but about other parts of the complete system development process that has been extensively discussed in Chapter 4.

Therefore, our conclusion is that the IAF is one of the polished up development methodologies that we referred too already in Chapter 1. Next, the IAF is about a different notion of architecture, namely the notion of model or blueprint, like all current architecture frameworks do.

7 Application: Case Educational Administration

In order to illustrate the core notions of enterprise ontology and enterprise architecture, as discussed in this book, this chapter contains the description and analysis of a case that is somewhat larger than the examples we used in the previous chapters. It has been published earlier in [19, 32]. Still, it is a 'toy' example compared to the real life problems of course, but that is unavoidable.
Subsection 7.1 contains the full description of the case, as well as the ontological analysis; these analysis parts are all in italic, interspersed in the text of the description. Subsection 7.2 contains the Interaction Model of the case. In subsection 7.3 an enterprise architecture is presented and discussed, which could be applicable to the case.

7.1 Description and ontological analysis

The HIT (Holland Institute of Technology) is a university of technology in the Netherlands, comprising fifteen schools, in seven faculties, which offer Bachelor and Master programs in a variety of technological areas. One of them is the school for Information Architecture (IA). This school offers a two-year Master Program with the same name. General information about the program can be found on the (public) website of the school. Although most students who do the IA program have either a B.Sc. in Computer Science (CS) or a B.Sc. in Systems Engineering, Policy Analysis and Management (SEPAM) from HIT, also students from other universities, in principle from all over the world, follow the Master program IA. In the school is a department of Educational Administration, which deals with all administrative matters concerning the Master Program. The operational activities of this department, for which we will use from now on the shorthand name EdA, are described hereafter.

The first thing students have to do if they want to follow the IA Program is to apply for admission. To this end, they have to fill out the so-called Program Admission Form, of which a PDF can be downloaded from the school's website. One has to add photocopies (Xerox) of the Bachelor diploma one possesses, or of a diploma that one considers as an alternative

(e.g. a Master diploma). The applications for admission are collected and processed by the Admission Office of EdA. The rules for admission are as follows:

1. Students with a BSc in CS from HIT or from some other Dutch university or from a European university in the so-called IDEA-league[1] are directly admitted.
2. Students with a BSc in SEPAM from HIT or with a BSc in Business Administration from a Dutch university are directly admitted.
3. Students with a BSc from a university on the 'black list', as well as students with no BSc or equivalent education at all, are not admitted.
4. In case of doubt, an application is passed through to the Program Director of the IA Program for approval.

So, the Program Director decides on all exceptional cases. In the long run, this may lead to adaptation of the admission rules that are applied by the Admission Office.

As the first step in developing the ontological model of EdA, we produce the global Interaction Model, in which all actor roles of EdA are represented as one composite actor role.
The first interface transaction type (T01) that can be identified from the text above is "program admission", with the result type "program admission A has been started". Regarding the State Model we can already identify the presence of a category "ADMISSION" and the existentially dependent binary fact type "Person P is the applicant of admission A". In order to make the ontological model of EdA generic we add to it the existentially dependent binary fact type "admission A applies to Program P". The initiator of T01 is the aspirant student and the executor is EdA.
The second interface transaction type (T02) that can be identified is "admission approval", with the result type "program admission A has been approved". The initiator of T02 is EdA. We will call the executor role "Admission Approver", while knowing that this role is now fulfilled by the organizational function "Program Director".

The Admission Rule Base is an external Production Fact Bank. The adding of new rules is not part of the responsibility of the EdA and therefore will not be contained in the ontology of the EdA.

Allowed admissions are entered into MASH (MSc Administration System HIT). A confirmation of the admission is sent to the student by e-mail, as well as by a written letter, which is delivered by the public postal mail service.

---

[1] The IDEA league is the next collection of cooperating universities: **I**mperial College of Science and Technology (London), **D**elft University of Technology, **E**idgenössische Technische Hochschule Zürich, and Rheinisch-Westfälische Technische Hochschule, **A**achen.

After having been admitted, students have to enroll in the courses they want to follow by means of the WhiteBoard system. The complete program and schedule of the courses can be found on the school's website. Successful enrolments are confirmed to the students by a corresponding e-mail message. The WhiteBoard system is also used for all kinds of communication between the lecturers of the courses and the students.

> We identify the third interface transaction type (T03), called "course enrollment" with the result type "course enrollment E has been started". For the State Model we identify the category "ENROLLMENT" and the existentially dependent binary fact types "Student S is the student in enrollment E" and "enrollment E regards course C". It is worthwhile now, if not necessary, to define precisely the concept course. The most convenient way is to conceive a course as being of a particular course type (e.g. IN4148) and given in a particular course period. This definition makes courses uniquely identifiable. Consequently, we assume that there is only one course of any course type per course period. The initiator of T03 is the student and the executor is EdA.

The scheduling of the courses is also a task of the EdA. Scheduling a course means that one has to find a free lecture time slot and a free lecture hall for a course, such that there are no conflicts of coinciding lectures within the IA program. Put differently, students can always take the courses they want. A lecture time slot consists of two consecutive lecture hours. A lecture hour effectively is 45 minutes, since by tradition it starts with 15 minutes spare time. A weekday has eight lecture hours, usually identified by the numbers 1 through 8. Therefore, an example of a lecture time slot is "Friday, 5th and 6th hour". The course year is divided into four periods of 9 weeks. Lectures are given in the first seven weeks. The eighth week is free for study and in the ninth week are the exams. The availability of the lecturers is not taken into account while scheduling the courses, which means that they are considered to be available. Of course, time conflicts for the lecturers in the schedule are avoided.

> We identify at least one internal transaction type within EdA, concerning the scheduling of courses. Further analysis will be taken up after the global IAM has been produced.

Exams are scheduled either in the morning (9-12 hrs.) or in the afternoon (14-17 hrs.) of the weekdays in the ninth week of a period. This is also a task of the EdA. Students have to register explicitly for the exams, via EARS. The system produces a list of all registered students for an exam, which is sent to the examiner (who is usually the same professor as the lecturer of the course). After having valuated the answers of the students, the examiner

writes the grades on the list and sends it back to the EdA. These data then are processed in EARS. For security reasons, students are not able to inspect their grades online. Instead they can apply to the students' desk of the EdA and ask for a printout of all their grades. The grades per course are also published on the WhiteBoard system. They can be accessed by the students on the basis of their personal user code and password.

> We identify at least one internal transaction type within EdA, concerning the scheduling of exams. Further analysis will be taken up after the global IAM has been produced.
>
> Next to that, we identify the fourth interface transaction type (T04), called "exam_registration" with the result type "student S has registered for exam E". For the State Model we identify the category "EXAM". Analogous to what we did for the concept course, we conceive an exam as being of a particular course type (e.g. IN4148) and held in a particular exam slot. This definition makes exams uniquely identifiable. The initiator of T04 is the student and the executor is EdA.
>
> Apparently, the valuation of exams is not a responsibility of EdA. The initiator of the valuation transaction type clearly is the student and the executor clearly is the examiner.

If a student has passed all exams of the IA Master Program, he or she is entitled to receive the diploma. In order to get a diploma, one has to fill out the so-called Diploma Application Form, which can be downloaded from the school's website. These forms have to be handed in or sent to the EdA who passes them to the Examination Board. This board formally decides about the issuing of diplomas. The EdA records these decisions.

> From this text, we identify the diploma awarding transaction type. Clearly, the initiator of this transaction type is the student and the executor is the Examination Board. The involvement of the EdA is mainly on the datalogical and infological level. At the ontological level, it seems that the EdA has the delegated authority for performing the promise act.

7.2 *The Enterprise Ontology of EdA*

Based on the analysis above, we produce the Interaction Model (cf. Section 3.5) of the EdA, expressed in an Actor Transaction Diagram (Figure 7.1) and the corresponding Trans-

action Result Table (Table 7.1). For the explanation of the Actor Transaction Diagram, the reader is referred to Section 3.5 or to [17]. Only the transactions T06 and T08 need further explanation. Their executors, A06 and A08 respectively, appear to be also the initiator of the transaction. Such actor roles are called self-activating. Self-activation is a convenient way to model periodic activities.

From the analysis in Section 7.1 it appears that there are five core object or entity types in the case EdA: admission, enrollment, student, exam, and period. Their instances are represented by a variable (capital letter) in the result specification in Table 7.1. Note that these variables are local; the scope is the result type in which they occur. Therefore, the letter "E" is can be used to denote enrollments in R03 and exams in R04. A similar reasoning holds for the letter "P" in R05, R06, R07, and R08.

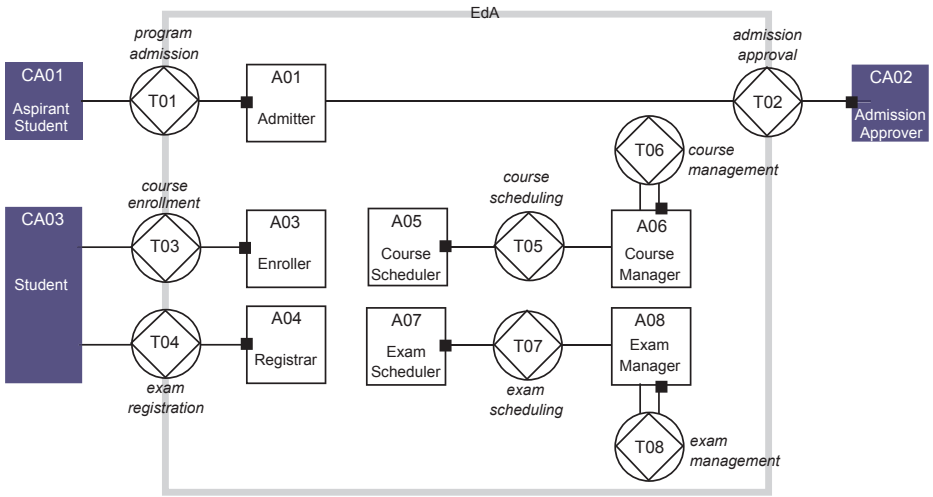


Figure 7.1 Actor Transaction Diagram of EdA

transaction type	*result type*
T01 program_admission	R01 *program admission A has been started*
T02 admission_approval	R02 *program admission A has been approved*
T03 course_enrollment	R03 *course enrollment E has been started*
T04 exam_registration	R04 *student S is registered for exam E*
T05 course_scheduling	R05 *the courses for course period P have been scheduled*
T06 course_management	R06 *the course management for course period P has been done*
T07 exam_scheduling	R07 *the exams for exam period P have been scheduled*
T08 exam_management	R08 *the exam management for exam period P has been done*

Table 7.1 Transaction Result Table of EdA

7.3 The Enterprise Architecture of EdA

With reference to the Generic System Development Process (Figure 5.1) it is clear that the ontology and the architecture of an enterprise are largely independent from each other, since many design principles do not regard the design of the ontological model but the engineering of the subsequent construction models down to the implementation model. Still it is possible, although not apparent from the case description, to conceive and formulate design principles that could have been applied or that could be applied in future changes to the EdA. Therefore, the way in which we will proceed, is to make assumptions about how the devising of the enterprise architecture could have been or could be.

Suppose that in the main strategic document of the HIT, the next three areas of concern are identified: *student satisfaction*, *flexibility*, and *compliance*. The first area of concern regards the ability to attract and retain students, whereas the second area addresses HIT's ability to adapt quickly to changing internal and external conditions. Furthermore, the area of concern "compliance" has to do with satisfying regulatory requirements; both national and European (like the Bologna treaty). Subsequent analysis by the enterprise architects that are hired from a well-known consultancy company concludes that the area of concern *student satisfaction* is addressed by the function of the B-organization and the construction of the B-organization. The architects formulate three design principles that should guarantee the desired level of student satisfaction. They are listed in the first three rows of Table 7.2. We consider them sufficiently self-explaining.

Further investigation by the enterprise architects reveals that the drive for student satisfaction necessitates easy access by students to the University network. An additional area of concern *security* is thereby identified. It is considered to be addressed by the construction of the I-organization and the function of the D-organization. Next, the architects propose to have the corresponding parts of these aspect organizations be automated. As a result, the system kinds I-application and D-application are added. The applicable design principles are listed in row 4 through 6 of Table 7.2.

Design principle	*Area of concern*	*System kind*	*Design domain*
Products and services must allow personalization by students	Student satisfaction	B-org	F
Management must enable employee self-management	Student satisfaction	B-org	C
Student administration must be locally present, under unified, central governance	Student satisfaction Flexibility	B-org	C
Complete and up-to-date student information must be available to students	Student satisfaction	I-org	F
Student information must be available from one unified source	Security	I-org / I-app	C

Network access must be based on authentication and role-based authorization	Security	D-org / D-app	F
Workflow control logic must be separated from process execution logic	Flexibility	D-org / D-app	C
Employee decision making must take place at the lowest possible level	Flexibility	B-org	C
Admission rules must be in accordance with the Bologna treaty	Compliance	B-org	F

Table 7.2 Examples of design principles in EdA

The area of concern *flexibility* appears to be addressed by the construction of the B-organization and the construction of the D-organization (and likewise of the D-applications). Two additional design principles are formulated (row 7 and 8), while the third principle is considered also to contribute to the desired level of flexibility. Finally, the area of concern *compliance* is addressed by the function of the B-organization. A possible design principle pertaining to compliance is listed in the last row of Table 7.2. This concludes our 'hypothetical' architecturing for the HIT.

Conclusions

Architecture and architecture framework are not what authoritative institutions, as well as most professionals in the area of business and ICT, consider it to be. Naming something "architecture", which was until then known as global model or fundamental organization or blueprint, is not a substantial contribution to the progress of the profession. Similarly, referring to existing system development methodologies by the name "architecture framework" is not an impressive act of innovation. Names are never important, only their designation counts, i.e., the concepts they represent. Substantial progress and innovation can only be achieved if new ideas emerge, ideas that appear to be much more effective in dealing with the problems at hand than the existing ones.

From the fundamental analysis of the system development process, it has become evident that there are three central notions involved. One is *technology*. Whatever kind of system one wants to develop, without thorough knowledge of the technological means with which the system could be built and the technological skills to actually do it, one will never build anything. Because the generic system development process (GSDP, cf. Figure 5.1) concerns every kind of system, this notion of technology includes 'human' technology. Consequently, human resource management turns out to be an engineering activity.

Next, before arriving at the implementation phase, a coherent, consistent, comprehensive, and yet concise conceptual model of the designed system is needed which is truly and fully independent of any implementation, including the technology to be selected. It is needed because only from such an understanding does one have the right freedom for engineering the system; only in this way could one arrive at innovative implementations. We have called this notion of implementation independent understanding the *ontology* of a system. It constitutes the fundamental organization of a system in the most essential way. Every system category has its own ontological meta model. For enterprises, the Ψ-theory and the DEMO methodology have proven to be most appropriate. In addition, it appears that many so-called 'technical' systems, including all computer applications, can be conceived very well as social systems. The 'technical' part is only in the implementation.

While designing and engineering a system, constrained by the provided functional and constructional requirements, one is still left with far too much freedom. It does make a lot of sense to formulate the design decisions one has to make explicitly, such that other people

can take notice of it. It does still make a lot more sense to restrict this freedom beforehand, and in a normative way, such that other, more general, objectives can be achieved. We have called this normative restriction of design freedom *architecture*. It is the third central notion in system development. Operationally, architecture is a coherent and consistent set of design principles, which guide the designer through the development process, in addition to the requirements. By applying this notion of architecture, one achieves various beneficial properties. The real challenge, however, is to get the design principles aligned with the strategic statements in an enterprise.

The discussion of the notions of Enterprise Ontology and of Enterprise Architecture has demonstrated that without them the complexity that professionals in the area of business and ICT face can hardly be mastered. We have also shown that both notions can be defined very precisely and very consistently. Enterprise Ontology and Enterprise Architecture are taken as the basic components of the discipline of Enterprise Engineering, which is currently emerging from the convergence of the traditional organizational sciences and the information sciences. The paradigm of Enterprise Engineering is that an enterprise is a designed system instead of an organically growing entity.

At the same time, we have noticed that the practical application of Enterprise Architecture, contrary to Enterprise Ontology, is still in its infancy. One can imagine that it is quite a job to translate mission statements in operational design principles. It is one thing to say that "our enterprise strives to be the best of its sort" but quite something else to achieve it. However, what is sure by now and by itself already of great value: one can only intellectually manage this enormous task by applying the notion of architecture as presented in this book. There is no other way. At the same time, there is also no escape: future enterprises will be required to be designed 'under architecture'. It is not unlikely that this requirement will become a quality standard one day.

Bibliography

1. Ackoff, R.L., Ackoff's Best: His Classic Writings on Management, New York, Wiley 1999.
2. Alexander, C.: *Notes on the synthesis of form* (Ablex Publishing Company, NJ, USA 1960).
3. Apostel, L.: Towards the formal study of models in the non-formal sciences. Synthese 12, (1960).
4. Austin, J.L.: *How to do things with words* (Harvard University Press, Cambridge MA 1962).
5. Baldinger, F., Dietz, J.L.G., Op 't Land, M.: Een generiek en uitbreidbaar Raamwerk voor (IT)-Architectuur: xAF, *Informatie & Architectuur* vol 3 en vol 4, 2005 (in Dutch).
6. Beer, M., Eisenbach, R.A., Spector, B., Why Change Programs Don't Produce Change, *Harvard Business Review*, November/December 1990, pp. 158-166.
7. Bertalanffy, L. von: *General Systems Theory* (Braziller, New York 1968).
8. Bertels, K., Nauta, D.: *Inleiding tot het modelbegrip* (Wetenschappelijke Uitgeverij, Amsterdam 1974).
9. Bunge, M.A.: *Treatise on Basic Philosophy, vol.3, The Furniture of the World* (D. Reidel Publishing Company, Dordrecht, The Netherlands 1977).
10. Bunge, M.A.: *Treatise on Basic Philosophy, vol.4, A World of Systems* (D. Reidel Publishing Company, Dordrecht, The Netherlands 1979).
11. Burlton, R.T., *Business Process Management*, Indianapolis, Sams Publishing 2001.
12. Dietz, J.L.G.: *Denkwijzen, methoden en technieken, Handboek Informatica*, B2000, Samsom Bedrijfsinformatie, 1996 (in Dutch).
13. Dietz, J.L.G., Mulder, J.B.F.: Integrating the Strategic and Technical Approach to Business Process Engineering. In: Scholz-Reiter, B., Stickel, E. (eds): *Business Process Modeling* (Springer-Verlag, Berlin Heidelberg 1996).
14. Dietz, J.L.G.: The Atoms, Molecules and Fibers of Organizations. *Data and Knowledge Engineering* 47, 301–325 (2003).
15. Dietz, J.L.G., Albani, A.: Basic notions regarding business processes and supporting information systems. *Requirement Engineering Journal* (fall 2005).

16. Dietz, J.L.G.: System Ontology and its role in Software Development. Proc. CAiSE'05 workshops, Lecture Notes in Computer Science, Springer 2005.
17. Dietz, J.L.G., *Enterprise Ontology – theory and methodology*, Springer-Verlag Heidelberg, Berlin, New York 2006.
18. Dietz, J.L.G.: The Deep Structure of Business Processes, in: *Communications of the ACM*, May 2006, vol. 49, no. 5, pp 59-64.
19. Dietz, J.L.G., Hoogervorst, J.A.P.: Enterprise Ontology and Enterprise Architecture – how to let them evolve into effective complementary notions, *GEAO Journal of Enterprise Architecture*, vol. 2, nr. 1, March 2007.
20. Dietz, J.L.G.: On the nature of Business Rules, in: Proceedings of the CIAO-EOMAS workshops at CAiSE 2008, Springer LNBIP10.
21. Dijkstra, E.W., A Discipline of Programming, Prentice-Hall series in Automatic Computation, New Jersey, 1976.
22. Drobik, A., *Enterprise Architecture: The Business Issues and Drivers*, http://www.gartner.com/DisplayDocument?id=366199.
23. Eckes, G., *The Six-Sigma Revolution*, New York, Wiley 2001.
24. Galliers, R,D., Baets, W.R., *Information Technology and Organizational Transformation*, Chichester, Wiley 1998.
25. Gharajedaghi, J., *Systems Thinking*, Boston, Butterworth Heinemann 1999.
26. Goedvolk, J.G., H de Bruin and D.B.B. Rijsenbrij: Integrated Architectural Design of Business and Information Systems; Proceedings of the Second Nordic Workshop on Software Architecture (NOSA'99), 1999.
27. Go, A., Lee, C., Dietz, J.L.G.: Awaraness bepaalt succes of falen van een EA-traject (in Dutch), *Tiem* 21, September 2007, pp 4-10.
28. Goldkuhl, G., Lyytinen, K., A language action view of information systems, in Ginzberg, M., Ross, C.A. (Eds.), Proceedings of the 3rd international conference on information systems, TIMS/SMIS/ACM, 1982.
29. Hammer, M., 1990. Reengineering Work: Don't Automate, Obliterate. *Harvard Business Review*. July-August, pp. 104-112.
30. Hoogervorst, J.A.P., *Quality and Customer Oriented Behavior. Towards a Coherent Approach for Improvement*, Delft, Eburon 1998.
31. Hoogervorst, J.A.P.: Enterprise Architecture: enabling Integration, Agility, and Change. *International Journal of Cooperative Information Systems*, Vol. 13 No. 3, September 2003, pp 213–233.
32. Hoogervorst, J.A.P., Dietz, J.L.G.: Enterprise Architecture in Enterprise Engineering, in: *Enterprise Modelling and Information Systems Architecture*, Vol. 3, No. 1, March 2008.
33. Jackson, M.C.: *Systems Thinking* (Chichester Wiley 2003).
34. Kalakota, R., Robinson, M., E-Business. Roadmap for Success, Reading MA, Addison-Wesley 1999.
35. Kaplan, R.S., Norton, D.P., *Strategy Maps*, Boston, Harvard Business School Press 2004.

36. Kirby, J., *Implementing CRM: Business Change Program, Not Project*, Gartner Research, July 2001.
37. Knox, S., Maklan, S., Payne, A., Peppard, J., Ryals, L., *Customer Relationship Management*, Oxford, Butterworth Heinemann 2003.
38. Langefors, B.: Information System Theory. *Information Systems* 2, 207–219 (1977).
39. Maier, M.W. Emery, D. Hilliard, R., Software Architecture: Introducing IEEE Standard 1471, *IEEE Computer*, April 2001, Vol. 34-4, pp 107-109.
40. Maier, M.W., Rechtin, E., *The Art of Systems Architecting*, Boca Raton, CRC Press 2002.
41. Miles, R.E., Snow, C.C., Fit, Failure and the Hall of Fame, *California Management Review*, Vol. 26, No. 3, 1984, pp. 128-145.
42. Oakland, J.S., Porter, L.J., *Cases in Total Quality Management*, Oxford, Butterworth-Heinemann 1994.
43. Op 't Land, M.: *Instrument for fast and effective splitting of organizations*, www.demo.nl/publicaties.
44. Rechtin, E. *Systems Architecting of Organizations*, Boca Raton, CRC Press 2000.
45. Reijswoud, V.E. van, J.B.F. Mulder, J.L.G. Dietz, Speech Act Based Business Process and Information Modeling with DEMO, *Information Systems Journal*, 1999.
46. Rico, D.: *IDEF0 methodology*, J. Ross Publishing, 2004.
47. Schekkerman, J.: *How to survive in the jungle of Enterprise Architecture Frameworks*, Trafford Publishing Ltd, Canada, Third Edition, July 2006.
48. Smith, H., Fingar, P., *Business Process Management: The Third Wave*, Tampa Fl., Meghan-Kiffer Press 2003.
49. TOGAF – The Open Group Architecture Framework (2003) http://www.opengroup.org/bookstore/catalog/g051.htm.
50. Weinberg, G.M., *An Introduction to General Systems Thinking*, John Wiley & Sons, 1975.
51. Winograd, T, F. Flores, 1986. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood NJ.
52. Woolridge, L., Hayden, F., How to get value from mergers, acquisitions and divestments, *CSC Research Journal*, November 2002.
53. Yourdon, E.: *Modern Structured Analysis*, Prentice-Hall, Inc., 1989.
54. Zachman,J., *Concepts of the Framework for Enterprise Architecture*, Zachman International Inc. 1997.

Glossary

Act An atomic unit of activity, of which the effect is the occurrence of an *event*. Two kinds of acts are distinguished: *coordination acts* and *production acts*. An act is performed by an *actor*.

Action rule An instruction or guideline for an *actor* in order to deal with a particular *type* of *agendum*. [see also *imperative business rule*]

Actor A *subject* in his or her fulfillment of an (elementary) *actor role*.

Actor cycle The cyclic state of activity of an *actor*, in which he or she deals with one *agendum*.

Actor role The unit of *authority*, *responsibility*, and *competence* in the Ψ-theory. [see also *elementary actor role*]

Agenda The plural form of *agendum*. The word is used also as a singular term, then referring to a set of agendums.

Agendum A *coordination event* that has to be dealt with by the *responsible actor*.

Application Software supporting *actors* in doing their work. Three gross kinds of applications are distinguished: *B-applications*, *I-applications*, and *D-applications*.

Application architecture The part of an *enterprise architecture* that regards the *development* of the *I-organization* and the supporting *I-applications*.

Architecture Conceptually, architecture is the normative restriction of design freedom. Operationally, an architecture is a coherent and consistent set of *design principles*. It regards one or more *system kinds* and one or more *design domains*, and it addresses one or more *areas of concern*.

Architecture framework Conceptually, an architecture framework is a structured checklist of issues that must be taken into account during *architecturing*. Operationally, it is a triple <S,D,A> where S is a set of *system kinds*, D is a set of *design domains*, and A is a set of *areas of concern*.

Architecturing The activity of devising an *architecture*. This comes down to formulating *design principles*. Ideally, design principles are aligned with the strategies of the supplier of the *object system*.

Area of concern One of the dimensions in an *architecture framework*. Areas of concern serve as classifiers for *design principles*. They represent collections of stakes of stake-

holders in the *development* process and/or in the resulting *object system*. Examples are security, compliance, and agility.

Authority The condition of being allowed to act. It can be assigned to a subject through *authorization* and *delegation*. [see also *competence*]

Authorization The way of assigning *authority* to a subject by which the subject becomes exclusively *responsible* for performing the *acts* that the authority allows for.

B-application (B from business) An *application* which supports the *construction* of the *B-organization* of an *enterprise* (see Figure 4.8).

B-organization (B from business) The aspect organization of an *enterprise* consisting exclusively of *ontological transactions*. The *actors* involved in these transactions are called B-actors.

Black-box model A conceptual *model* of the *function* of a *concrete system*. Synonym: *functional model*. [see also *white-box model*]

Business architecture The *architecture* that pertains to the *development* of the *B-organization* of an *enterprise*.

Business process A tree structure of *transaction types*.

Business rule A unit of restricting the operational freedom of *actors*. It represents a limitation of the *state space* or the *process space* of either the *coordination world* or the *production world* of an organization. Depending on the way of formulation, one speaks of *declarative business rule* or *imperative business rule*.

Cancellation pattern [see *transaction pattern*]

Competence The ability of a *subject* to perform a particular *type* of *production act* as well as the corresponding *coordination acts*. Put differently, it is the ability to be the *executor* of a *transaction type*.

Composite actor role A non-*elementary actor role*. It comprises a number of elementary actor roles plus their *interaction* and *interstriction* relationships.

Composition [see *system*]

Concrete system A system that exists in the objective world, as opposed to a conceptual system (see Figure 2.2).

Construction A notion related to the notion of *system*. The composition, the environment, and the structure of a system are collectively called the construction of the system (see Figure 2.1).

Construction architecture The *architecture* which his applicable to the *construction design* and the *engineering* of a *system*. It consists of *constructional principles*.

Construction design The design phase that starts from the *functional model* of the *object system*, and ends with the *ontological model* of the object system (see Figure 5.1). It has two inputs: *constructional requirements* and *constructional principles* [see also *engineering*]

Constructional (de)composition A technique for mastering the complexity of the *construction* of a *system*. The effect of applying the technique is that a *constructional model* of a *(sub)system* is replaced by a set of constructional models, representing its *constructional subsystems*, and vice versa (see Figure 2.3). [see also *functional (de)composition*]

Constructional model A *white-box model* of a *system*. There is an ordered set of constructional models for every *designed* and *engineered* system. The 'highest' one is the *ontological model*; the 'lowest' one is the *implementation model*.

Constructional principle *Design principle* that regards the *construction design* and/or the *engineering* of a *system*.

Constructional requirements The requirements that the customer in a *development* project provides with respect to the *construction* of the *object system*. They are also called non-functional requirements.

Constructional specifications The constituent elements of the *constructional models* of an *object system*, from the *ontological model* down to the *implementation model*.

Constructional subsystem Synonym for *subsystem*.

Coordination act An act by which a *coordination event* is caused. The actor who performs the act is called the performer and the one to which it is addressed is called the addressee. A coordination act has two components: the intention and the proposition. In order to perform a coordination act, a number of communicative acts have to be performed (see Figure 3.5).

Coordination event The becoming existent of a *coordination fact*.

Coordination fact An elementary state of affairs in the *coordination world*.

Coordination world One of the two *worlds* in which the *actors* in an organization cause *events*. A state of a coordination world is a set of *coordination facts*.

D-application (D from data and document) An *application* which supports the *construction* of the *D-organization* of an *enterprise* (see Figure 4.8).

D-organization (D from data and document) The aspect organization of an *enterprise* consisting exclusively of *datalogical transactions*. The *actors* involved in these transactions are called D-actors.

Datalogical A notion related to *production acts*. Production acts that concern the storing, transmitting, copying, destroying etc, of data or documents are called datalogical production acts.

Datalogical transaction A *transaction* of which the *production act* is *datalogical*.

Declarative business rule The formulation of an *existence law* or an *occurrence law* in a declarative way.

Delegation The *subject* who is authorized for performing an *act*, may delegate the *authority* to someone else. If a subject A delegates authority to a subject B, subject A remains (also) *responsible* for the acts of subject B. [see also *authorization*]

Design A notion related to *systems*. It is the process of producing conceptual *models* of an *object system*, which is divided in two consecutive phases: *function design* and *construction design*. It is normally performed iteratively and incrementally (see Figure 5.1).

Design domain One of the dimensions in an *architecture framework*. The two basic design domains are *function* and *construction*. Each of these may be subdivided however.

Design principle A guideline applying to the *design* of a *system*. We distinguish between *functional principles* and *constructional principles* (see Figure 5.1). A design principle regards one *system kind* and one *design domain*; it may address several *areas of concern*.

Development The whole process of bringing about an *object system*. It comprises all activities that have to be performed in order to arrive at an implemented system. The main phases are *function design*, *construction design*, *engineering*, and *implementation*.

Elementary actor role The *authority*, *responsibility*, and *competence* for being the *executor* of exactly one *transaction type*.

Enterprise An organized human undertaking, like a company, a governmental agency, and a non-profit institution. An enterprise may be rather permanent, like a railway company, or temporary, like the project of building a railway section.

Essential A notion related to systems. The essence of a *system* is the understanding of its construction, as represented by its *ontological model*.

Engineering A notion applied in a broad and in a narrow sense. In the broad sense, it is a discipline. Examples are mechanical engineering and enterprise engineering. In the narrow sense, it is the phase in system development that starts from the *ontological model* of *the object system* and ends with its *implementation model* (see Figure 5.1).

Enterprise Architecture (operationally) The whole set of *design principles* that are applicable to the (re-) designing and (re-) engineering of an *enterprise*. Three partial architectures are distinguished: the *business architecture*, the *application architecture*, and the *technical architecture*.

Enterprise Engineering The emerging discipline concerning the (re-) *design* and the (re-) *engineering* (in the narrow sense) of *enterprises*, in all aspects. Two main pillars of the discipline are *Enterprise Architecture* and *Enterprise Ontology*.

Enterprise Ontology (operationally) The whole of the three distinct ontological models of an *enterprise*, regarding respectively its *B-organization*, its *I-organization*, and its *D-organization*.

Environment [see *system*]

Event The becoming existent of a *fact* as the result of an *act*. In more general terms: a *transition* at a particular point in time. Events regarding the same transition are said to be of the same event type.

Executor A role of an *actor* in a *transaction*. An actor who is *authorized* to be the executor of a transaction, is *responsible* for performing the *production act* and the corresponding *coordination acts*, according to the *transaction pattern*.

Execution phase The *transaction* phase in which the *executor* performs the *production act*. It starts from the transaction status promised and it ends with the transaction status stated (see Figure 3.2).

Existence law A law that requires or prohibits the coexistence of *facts* in a *state* of a *world*. Existence laws are expressed in *business rules*.

Extensible Architecture Framework [see *xAF*].

Fact An elementary state of affairs in a *world*.

Forma The ability of a subject to perform *formative acts* as well as *datalogical production acts*.

Formative act A notion related to *coordination acts*. Uttering data (speaking, writing) and perceiving data (listening, reading) are called formative communicative acts (see Figure 3.5).

Formative exchange A number of to and fro *formative acts* between two *subjects*.

Function A notion related to the notion of *concrete system*. The function of a concrete system is the set of services it is able to provide.

Function architecture The *architecture* that has been or is going to be applied to the *function design* of a *system*. It consists of *functional principles*.

Function design The design phase that starts from the *ontological model* of the *using system*, and ends with the *functional model* of the *object system*. It has two inputs: *functional requirements* and *functional principles*.

Functional (de)composition A technique for mastering the complexity of the *function* of a *system*. The effect of applying the technique is that a *functional model* of a *(sub)system* is replaced by a set of functional models, also called functional subsystems, and vice versa. [see also *constructional (de)composition*]

Functional model A *black-box model* of an *object system*, expressed in terms of the *using system*. It contains all *functional specifications* for the *object system*.

Functional principle *Design principle* that regards the *function design* of a *system*.

Functional requirements One of the inputs for the *function design* of an *object system*. They are provided by the *customer* of a *development project*, and expressed in terms of the *using system*.

Functional specifications The constituent elements of the *functional model* of an *object system*.

Functional subsystem [see *functional (de)composition*]

Heterogeneous system A system that is a layered nesting of a set of *homogeneous* systems of different *system categories*.

Homogeneous system A system in one *system category*.

I-application (I from information) An *application* that supports the *construction* of the *I-organization* of an *enterprise* (see Figure 4.8).

I-organization (I from information) The aspect organization of an *enterprise* consisting exclusively of *infological transactions*. The *actors* involved in these transactions are called I-actors.

Imperative business rule The formulation of an *existence law* or an *occurrence law* in an imperative way. It is equivalent to *action rule*.

Implementation As a verb, it is the activity of making a *system* operational by means of appropriate technology. As a noun, it is the state of being implemented. [see also *realization*]

Implementation model The *constructional model* of a (homogeneous) *system* that contains all the details necessary for implementing the system.

Infological A notion related to *production acts*. Production acts that concern reproducing, deducing, computing etc. of information are called infological production acts.

Infological transaction A *transaction* of which the *production act* is *infological*.

Informa The ability of a subject to perform *informative acts* as well as *infological production acts*.

Information bank Either a *coordination bank* or a *production bank*.

Informative act A notion related to *coordination acts*. Expressing thoughts (formulating) and educing thoughts (interpreting) are called informative acts (see Figure 3.5).

Informative exchange A number of to and fro *informative* acts between two *subjects*.

Initiator A role of an *actor* in a *transaction*. An actor who is *authorized* to be the initiator of a transaction, is *responsible* for performing the corresponding *coordination acts*, according to the *transaction pattern*.

Integration rule A construction rule for *extensible architecture frameworks* (xAFs). A new *xAF* is constructed as the integration of several existing xAFs.

Interaction Conceptually, it is the active mutually influencing of *actors*. Operationally, interaction is the specification of the *transaction types* and the *actor roles* involved.

Interstriction Conceptually, it is the passive mutually influencing of *actors*. Operationally, interstriction is the specification of the information links between *actor roles* and *information banks*.

Model A role of a *system*. Three types of models are distinguished: concrete models, conceptual models, and symbolic models (see Figure 2.2).

Object system A *system* that is the object of *designing* and *engineering*. [see also *using system*]

Occurrence law A law that requires or prohibits sequences of *events* in a *world*. Occurrence laws are expressed in *business rules*.

Ontological A notion related to *production acts*. Production acts that concern bringing about original new facts, like deciding, judging, and manufacturing, are called ontological production acts.

Ontological model The *constructional model* of a (homogeneous) *system* that is fully independent of its *implementation*.

Ontological system notion [see *system*] [see also *teleological system notion*]

Ontological transaction A *transaction* of which the *production act* is *ontological*.

Ontology Generally speaking, it is the metaphysical study of the nature of being and existence. Specifically (in this book), the ontology of something is a conceptual *model* that satisfies the next requirements: coherent, comprehensive, consistent, concise, and *essential*.

Order phase The *transaction* phase in which the *initiator* and the *executor* strive to reach consensus about the *transaction result* that the executor has to bring about. It starts from the transaction status requested and ends with the transaction status promised.

Performa The ability of a subject to perform *performative acts* as well as *ontological production acts*.

Performative act A notion related to *coordination acts*. It establishes the appropriate commitment in the performer and evokes the appropriate commitment in the addressee of a coordination act.

Performative exchange A number of to and fro communicative acts between two *subjects*. If successfully, a *performative act* is performed.

Process space A notion related to *worlds*. The process space of a world is the set of lawful sequences of *events* in the world, i.e., sequences that satisfy the applicable *occurrence laws*. [see also *business rule*]

Process step The building block of *transactions*. It consists of either a *coordination act* and the *coordination event* caused by it, or a *production act* and the *production event* (or *transaction result*) caused by it.

Production act The *act* in a *transaction* by which the *executor* causes a *production event* (or *transaction result*).

Production bank A conceptual store of *production facts*. There is a production bank for every *transaction type*.

Production fact An elementary state of affairs in the *production world*.

Production event The becoming existent of a *production fact*.

Production world One of the two *worlds* in which the *actors* in an organization cause events. A *state* of a production world is a set of *production facts*.

Realization As a verb it is the activity of establishing a layered nesting of *homogeneous systems*. As a noun it is the state of being realized. [see also *implementation*]

Responsibility Quality of a *subject* to be committed to the *coordination acts* he or she has performed, as well as to *coordination acts* that are addressed to him or her. [see also *authority*]

Result phase The *transaction* phase in which the *initiator* and the *executor* strive to reach consensus about the *production result* that the executor has brought about. It starts from the transaction status stated and ends with the transaction status accepted.

Reverse engineering The activity of reconstructing the *ontological model* of a *system* from its *implementation model*. [see also *engineering*]

Self-activation A notion related to *actor roles*. An actor role is called self-activating if it is the *initiator* ánd the *executor* of the same *transaction type*.

Social system A system in the *system category* of social systems. The elements are social individuals or *subjects*. The mutual influencing between the elements is of two kinds: *interaction* and *interstriction*.

Specialization rule A construction rule for *extensible architecture frameworks* (xAFs). A new *xAF* is constructed as the specialization of an existing xAF.

State A notion related to *worlds*. At any moment, a world is in some state, defined as the set of *facts* that exist.

State space A notion related to *worlds*. The state space of a world is the set of lawful *states*, i.e., states that satisfy the applicable *existence laws*. [see also *business rule*]

Structure [see *system*]

Subject A human being in his or her quality of social individual, i.e., being able to enter into and comply with commitments. Only subjects can fulfill *actor roles*.

Subsystem A *system* S2 is a subsystem of a system S1 if and only if the *composition* of S2 is a subset of the composition of S1, the *structure* of S2 is a subset of the structure of S1,

and the *environment* of S2 is a subset of the union of the environment of S1 and the set-theoretic difference of the composition of S1 and the composition of S2.

System Something is a (homogeneous) system if and only if it has composition, environment, production, and structure. The composition and the environment of a system are both a set of elements of the *system category*. These sets are disjoint. The structure is the set of influencing bonds among the elements in the composition, and between them and the elements in the environment. By production is meant that the elements in the composition produce services that are delivered to the elements in the environment.

System category Every *homogenous system* belongs to exactly one system category. It determines the kind of elements of a system, as well as the kind of influencing bonds between the elements. Examples of system categories are: physical, chemical, biological, social.

System kind One of the dimensions in an *architecture framework*. Typical system kinds are organizations, information systems, and ICT infrastructure systems. The distinguished system kinds may be of the same *category*, e.g., the *B-organization* and the *I-organization*.

Technical architecture The part of an *enterprise architecture* that regards the *development* of the *D-organization* and the supporting *D-applications*.

Technology The means by which the *implementation model* of a system can be made operational, including the skills that are needed in order to do this.

Teleological system notion *Black-box model* of a *system*. [see also *ontological system notion*]

Transaction A sequence of acts that is a path through the complete *transaction pattern*, minimally comprising the basic pattern. It goes off in three consecutive phases: the *order phase*, the *execution phase* and the *result phase*. There are three gross kinds of transactions: *ontological*, *infological*, and *datalogical transactions*.

Transaction pattern A pattern of *acts* between two *actor roles* for bringing about a *production fact* (or *transaction result*). The basic transaction pattern consists, besides the *production act*, of the *coordination acts* request, promise, state, and accept (see Figure 3.2). The standard transaction pattern includes also the coordination acts decline, quit, reject, and stop. The complete transaction pattern consists of the standard pattern and the four cancellation patterns (see Figure 3.3).

Transaction result The *production fact* that is going to be brought about (or has been brought about) as the effect of executing the *transaction*.

Transition (also called event type) A notion related to *worlds*. A transition is a change of *state* of a world. It is defined as a pair of states: the state before and the state after the change. [see also *event*]

Using system A term related to the layered nesting of systems. A system at level $n + 1$ is said to be the using system of a system at level n ($n > 0$). In the process of *designing* a system at level n, this system is called the *object system*.

White-box model A conceptual *model* of the *construction* of a *concrete system*. Synonym: *constructional model*. [see also *black-box model*]

World A notion for adequately dealing with the effects of the activity of a *system*. Every system has its own world (sometimes divided in one or more partial worlds). The effect of an *act* by the system is an *event* in the system world, resulting in the becoming existent of a *fact*.

xAF An Extensible Architecture Framework (xAF) is an *architecture framework,* which is defined on the basis of one or more other xAFs by (possibly repeatedly) applying a construction rule: the *integration rule* or the *specialization rule*.

xAF lattice All xAFs constitute together a lattice. Every node in the lattice is an xAF. The universal root node regards the generic *homogeneous system* as the *system kind*, and the *design domains* function and construction. The set of *areas of concerns* is empty.

Index

Architecture is omnipresent and natural. Every child applies it, and it will keep doing so as a grown-up. Both Salvador Dali and Vincent van Gogh were guided by functional and constructional principles. Examples of functional principles for them were to shock or to please. Examples of constructional principles were to use oil color or water color. Design principles are the operational shape of architecture. Conceptually, it is normative restriction of design freedom. The rationale for architecture is simply that our design freedom is always too large. The question then is: how are you going to use this freedom? Through the whole history of mankind, people have answered the question in much the same way; they use design freedom for expressing their individual or collective vision.

In this book, architecture is introduced and elaborated for the area of business and ICT (information and communication technology). Fortunately, there is a lot of talk about architecture in this area. Unfortunately, the original notion of architecture has degenerated into something like a blueprint or a global design. This degeneration process must be stopped, the sooner the better. Certainly, there is also a need for global understanding of complex systems, for abstraction from irrelevant details. However, for this goal we have the notion of system ontology. It provides the understanding of a system in a coherent, consistent, complete and concise way, fully abstracted from all implementation aspects.

The aim of the book is to make the notion of architecture crisp and clear and to show how one can benefit from it in (re) designing and (re) engineering systems in the area of business and ICT, ranging from infrastructural networks to enterprises. Having a corporate strategy is good but having the means to make this strategy operational is better. That is what architecture can achieve: to build strategy into design.

This book is the final report of the xAF (Extensible Architecture Frameworks) working group of the NAF (Netherlands Architecture Forum).

About the author

Jan Dietz is professor in Information Systems Design at Delft University of Technology (The Netherlands). He is the spiritual father of DEMO (Design & Engineering Methodology for Organizations), co-founder and chairman of the DEMO Center of Expertise (www.demo.nl), member of the board of the NAF and chairman of the NAF working group xAF (www.xaf.nl).

ISBN 978 90 12 58086 1
NUR 982

ACADEMIC SERVICE

9 789012 580861

www.academicservice.nl