1. Introduction

In the discussion of the τ-theory [TEEM-2], we have noticed that human beings make artefacts for the sake of getting desired affordances, called functions, next to using the affordances that they perceive in the (natural or artificial) things that surround them. In this memorandum, we elaborate the design of artefacts, by introducing and discussing the β-theory (β is pronounced as BETA, standing for Binding Essence, Technology, and Architecture). It is a theory about the design of systems (as defined by the τ-theory [TEEM-2]), of any category, which also provides clear, precise and coherent definitions of common terms such as "development", "requirements", "specifications", "architecture", "design", "engineering", and "implementation".

By "*Essence*" is understood both the functional essence and the constructional essence of a system. *Functional essence* is defined as the collective services (affordances) that the system provides to some group of stakeholders, notably its users or using system. The *constructional essence* of a system is defined by the π-theory for technical systems [TEEM-7], and by the ψ-theory for social systems, notably organisations [TEEM-5]. By "*Technology*" is understood the applicable means for implementing the system, starting from its implementation model (section 4). Example technologies for organisations are ICT and human beings. The most troublesome term is "*Architecture*". As was pointed out in e.g. [Hoogervorst, Dietz, 2008] and [Dietz, Hoogervorst, 2011], there seems to be no term in the field of organisation and ICT as vague and as ambiguous as this one. At the same time, it is probably the most frequent and most widely used term, not only in the practice of (re)designing and (re)engineering business processes and information systems, but also in the scientific research concerning these topics. Right now, we like to make crystal clear already that we will forcefully reject the prevalent current meaning of "architecture" as global design, or model, or blueprint, or the like. This notion exists already, under the names mentioned, although it is rarely fully clear wether these names refer to the function or the construction perspective.

At the same time, system designers in the field of organisation and ICT are in great need for help in properly using the design freedom that is left after all specific requirements are satisfied. In the fields of constructional building and industrial design, such guidance is called "architecture". By means of architecture, these designers are enabled to express a personal or collective vision on the artefact, in such a way that the functionality of the artefact is emphasised and made apparent, as well as enriched by current cultural values. Therefore, we will explore and elaborate this notion of architecture for the discipline of Enterprise Engineering, instead of joining the current nonsensical parade of vagueness and ambiguity.

This extended summary presents and discusses the main topics and achievements of the β-theory, including its relationship with the other enterprise engineering theories, as mentioned in the classification scheme in figure 1.1 (in this scheme, the main influences are from bottom to top).

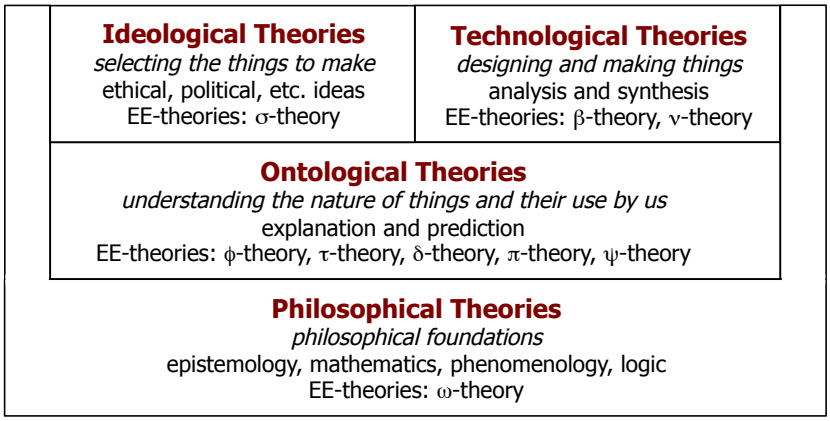

Figure 1.1 Classification scheme for enterprise engineering theories

The β-theory is a *technological* theory, which means that it concerns understanding the designing and making of things. Next to being rooted in the τ-theory, the π-theory, and the ψ-theory, it also builds on [Alexander, 1960] and [Simon, 1996]. Several parts of the β-theory are extensively discussed in [Dietz, 2006], [Dietz, 2008], and [Hoogervorst, 2009], where it is still part of the (former) τ-theory. The β-theory is the main vehicle for properly addressing the full complexity in the (re)design, (re)engineering and (re)implementation of enterprises.

2. The Generic System Development Process

From the ψ-theory [TEEM-5] we know that every product is produced in a transaction process between an initiator (the actor who orders the product) and an executor (the actor who is responsible for delivering the product to the initiator). This process is represented in a comprised way by the organisational building block, as exhibited in Figure 2.1. Let us analyse the transaction process of a transaction Tn for the case that the product must be made to order, and that, in addition, it has to be developed from scratch. This holds a.o. for the production of most information systems, both enterprise information systems (EISs) and information systems that are embedded in technical systems.

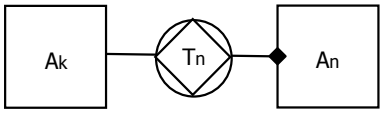

Figure 2.1 Organisational building block

During the proposition phase of a transaction of the kind Tn, the initiator (fulfilling actor role Ak) and the executor (fulfilling actor role An) come to agreement about the product to be developed. As we know from the τ-theory [TEEM-2], the product that is delivered by An is taken by Ak as the function (affordance) it has asked for. Because the product doesn't exist yet, the agreement is necessarily based on specifications of the needed function. In accordance with the τ-theory, these specifications are for the largest part functional specifications, but they may also include constructional ones. Mostly, and certainly if information systems are concerned, the product is not meant to be a personal affordance for the initiator but it is intended to serve the needs of another system (an organisation in case of an EIS and a technical system in case of embedded software). This is illustrated by the generic system development process (GSDP) [Dietz, 2008], exhibited in Figure 2.2. In this figure, the object system could be the EIS to be developed, and the using system would then be the organisation that is going to use the services that the EIS can provide, collectively called its functionality.

The complete design process is divided into two, logically consecutive, phases: function design and construction design. *Function design* consists of selecting a subset from the functional requirements (and the functional architecture, which will be discussed in Section 4) on which the customer (Ak) and the supplier (An) agree, and of transforming them into detailed and precise *functional specifications*. These constitute collectively a black-box model of the object system, referred to as the object system function in the GSDP. Note that the functional specifications exclusively regard the functions that the using system expects from the object system once it is installed and put into operation. The construction of the object system is largely indifferent to the using system. By definition, functional specifications are expressed in 'the language' of the construction of the using system. To give an example, if the using system is a sales department, a possible affordance expression could be "daily turnover". Although one may have an idea about how the product would look like, that is going to be perceived by the accounting people as the needed daily turnover, the specification of the function doesn't tell anything about the construction of the product.

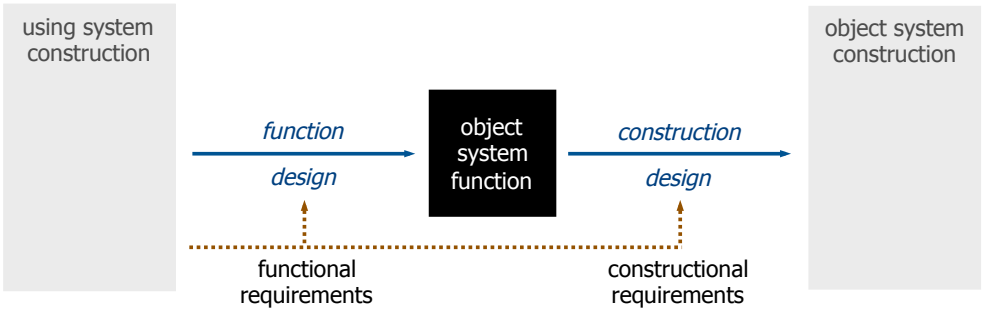

Figure 2.2 Generic System Development Process (1)

The real creative design phase in the GSDP is *construction design*, because now the designer has to throw a mental bridge between two different 'realms': the 'realm' of the using system, in which the function of the object system is specified, and the 'realm' of the object system, in which some construction must be devised that, once installed and put into operation, is perceived by the using system as the specified functionality. To follow up the example we used before, one of the products that the sales information system must be able to deliver is the result of some calculation, that could serve as the daily turnover for the sales department.

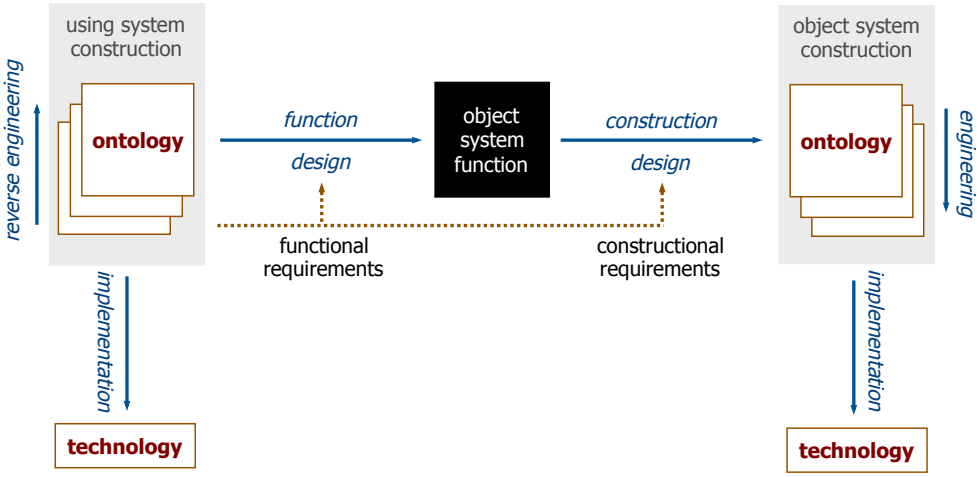

Figure 2.3 Generic System Development Process (2)

The construction design phase actually consists of two sub phases: ontological design and implementation design, which is called "*engineering*" in most engineering disciplines. This is shown in Figure 2.3. *Ontological design* consists of devising the ontological model of the object system, starting from the functional specifications. The ontological model is the highest level construction model of the object system, fully independent of all implementation details. *Implementation design* consists of developing the lowest level construction model, called the *implementation model*, which is directly implementable given some appropriate technology. This sub phase starts from the ontological model, and generally comprises the development of a number of intermediate models, as shown in the picture. The *implementation* phase of the complete development process consists of assigning technological means to the elements of the implementation model. In general, a number of alternative technologies will be available, from which one may choose. As an illustration, Figure 2.4 presents two alternative technologies to implement the single original transaction of withdrawing money from one's bank account (Cf. [TEEM-5]).

Figure 2.4 Alternative implementation technologies of a withdrawal transaction

The left side of Figure 2.3 shows that the using system is also implemented by applying suitable technologies and that it also has its implementation model and its ontological model. The functional and constructional requirements for the object system are preferably based on the ontological model of the using system, in order to keep the design of the object system independent of the using system's implementation. However, this ontological model may not be available, in which case it has to be reconstructed from its implementation model or even from its implementation. This reconstruction process is called *reverse engineering*.

3. Developing enterprise information systems

Instead of trying to explain the GSDP for developing any kind of systems, we choose to take a specific class of object systems, namely enterprise information systems. The ψ-theory [TEEM-5] tells us that the organisation of any enterprise consists of three aspect organisations: the B-organisation, the I-organisation, and the D-organisation. From section 3.2 of TEEM-5 we quote the next definition of an *enterprise information system* (EIS): "... it is some implementation of some realisation of the I-organisation of (a part of) the enterprise, and its supporting D-organisation. This definition shows at once the intrinsic and intensive interweaving of an EIS with the organisation supports. Therefore, a proper metaphor for an EIS is the nervous system of the human body: it is intrinsically and intensively interwoven with the body, and it cannot easily be replaced by another one."

Let us have a closer look at this relationship between an EIS and the organisation it supports, and in particular at the development of an EIS for a given enterprise, as a special application of the GSDP. To this end, let us first discuss how the I-organisation of an enterprise can be designed, based on its essential model (a topic that is thoroughly dealt with in [Jong, 2013]). From section 3.2 of TEEM-5 we also quote the next definition of essential model: "The ontological model of the B-organisation, in which the information sharing services by the supporting I-organisation are modelled as *information links* between actor roles and transaction kinds (now interpreted as conceptual containers of C-facts and P-facts in the B-organisation), is called the *essential model* of (the complete organisation of) the enterprise". Put differently, and with further reference to the ψ-theory, the I-organisation is the *realisation* of all informational actions: the remembering of C-facts (including agenda) and P-facts, the recalling of C-facts and P-facts, and the computation of derived facts. All these actions take place in informational transactions between B-actors (in their informational shape) and I-actors, and among I-actors. In order to illustrate this at a concrete example, we take again the sales department (Figure 3.1).

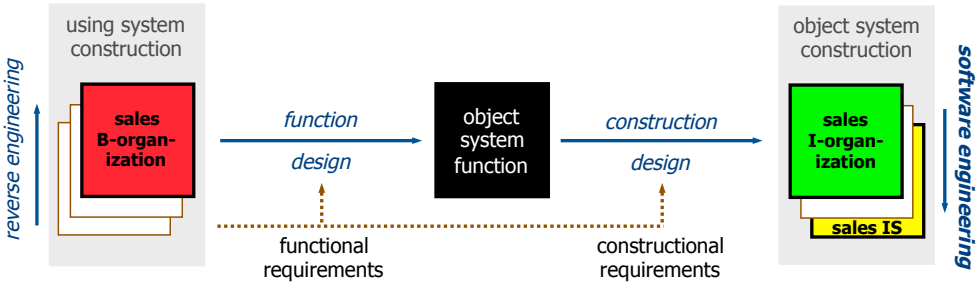

Figure 3.1 Designing an enterprise information system

The red colored box at the left side represents the ontological model of the B-organisation of the sales department (the using system). The object system function consists of the specifications of the informational services that the B-actors need. As we have seen, they comprise remembering, recalling and computing C-facts and P-facts from the sales B-organisation. The green colored box at the right side represents the ontological model of the I-organisation of the sales department. This sales I-organisation is a network of informational actor roles and transaction kinds that, once realised, implemented and put into operation, delivers informational products that are perceived by the B-actors as the informational services they need. Let us call the implementation model of the sales I-organisation the sales information system (sales IS). Regardless the implementation technology that is or will be applied, Figure 3.1 emphasises the intrinsic and intense interweaving of the sales IS with the sales B-organisation that it supports. Conversely, it means that any sales IS, e.g. one bought 'off the shelve' or one that is a configured ERP-system, by definition imposes some ontological model of the B-organisation on the enterprise

(although it is probably inadequate and incomplete). This theoretical insight has important practical implications. One of them is that the developers of EISs, once more, regardless whether these systems are fully customer made or customised 'standard' solutions (like ERP-systems), must be aware that their solutions will only be fully satisfactory if their functional specifications are based on the essential model of the using system. In current practice, it is highly improbable that they are. The consequence in current practice is that organisations feel 'girded' by their ERP systems. Ironically, neither the customer nor the ERP supplier understands why. Another implication is that apparently the customers of ERP systems need to be competent in determining and approving requirements.

4. Architecture

In explaining the name "BETA" in the Introduction, we have called "architecture" the most troublesome term because it is currently surrounded by so much vagueness and ambiguity in the state of the art in the field that is commonly labeled as 'organisation and ICT".

For all engineering disciplines, it holds that during a design process, the designer is left with some 'amount' of design freedom after all specific requirements for the artefact are satisfied. This 'amount' may be quite substantial for disciplines where the implementation technology is hardly or not at all constrained by physical laws. Among these disciplines are Information Systems Engineering and Enterprise Engineering. Consequently, designers of enterprises and information systems are faced with the question how to use this design freedom. Throughout the history of mankind, man has answered this question in much the same way, namely by using this freedom for expressing a personal or collective vision on the artefact to be designed. As an example, designers of wooden walking sticks, often use their design freedom to let the stick evoke the experience of nature, e.g. by gouging on the stick pictures of nature (trees, flowers, mountains, trails, etc.). More generally, designers of many artefacts use their design freedom to appeal to aesthetic experiences or to impose their design style. As another example, designers of churches, presumably being religious themselves, mostly use their design freedom to appeal to religion related concepts, like devotion, hospitality, and sense of safety.

Before presenting a thorough and appropriate definition of architecture for the discipline of EE, let us consider some examples from the building industry, namely these three opera houses: The Sydney Opera House, The Scala in Milan, and the Metropolitan Opera in New York (Figure 4.1). Their functional requirements were roughly the same, and neither of them had serious constructional constraints. Then, why are they so different?

Figure 4.1 Examples of architecture in the building industry

Clearly and indisputably, the difference between the three opera houses is in the way in which the design freedom was used by their (teams of) architects. We leave it to the reader's knowledge or imagination what the architects wanted to express in the way they used their design freedom. It is also clear and indisputable that this is what these architects call "architecture". Therefore, instead of attempting to invent something new, let us try to apply this notion of architecture to the field of EE. To this end, we extend the GSDP picture in Figure 2.3 to the one that is exhibited in Figure 4.2.

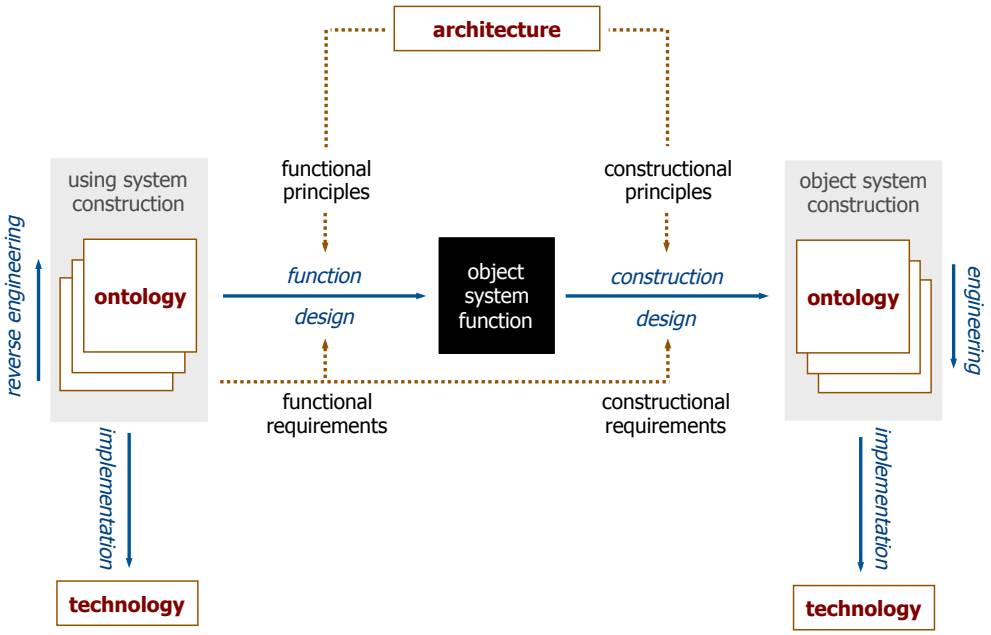

Figure 4.2 Generic System Development Process (3)

The figure shows that "architecture" is the collective name for functional and constructional principles. The functional principles are an additional 'input' for the function design phase, next to the functional requirements, whereas the constructional principles are an additional 'input' for the construction design phase, next to the constructional requirements. Consequently, principles must be understood as generic requirements, i.e. requirements that are not specific for the object system to be designed, but that apply also to other object systems. This is an important observation. Among other things, it means that one cannot tell from the formulation of a requirement that it is a 'real' (i.e. specific) requirement or a design principle of the applicable architecture. Here are some examples from the automotive industry:

braking distance within 12 m at 50 km/h
acceleration to 100 km/h within 8 seconds
voice controlled music selection
traffic information must override music

These can be formulations of functional requirements but also of functional principles (but both regard the stakeholder group "drivers"). The difference between (specific) requirements and (generic) principles is in the way they are determined. Requirements are determined during the development process of a specific object system. In the automotive industry, the second requirement (*braking distance within 12 m at 50 km/h*) could be a specific requirement for a new car model. Principles are determined independent of the design of a particular object system. Instead, they are determined in a separate and permanent process, which we propose to call "*architecturing*". The mentioned requirement (*braking distance within 12 m at 50 km/h*) could as well be a principle that holds for all car models, e.g. because of corporate or national safety rules. This brings us to the major sources of design principles. One of them is the mission and strategy of the enterprise at hand, for example a car manufacturer. This enterprise may have chosen the safety of cars as a key strategic issue, because it has been found that most customers give a high value to the experience of safety. The second major source of design prin-

ciples consists of existing and applicable (mandatory or recommended) laws, norms and standards, which come from outside the enterprise. The car manufacturer has to decide which laws, norms and standards it wants to adopt, and consequently have to be translated into design principles. Similarly, the enterprise has to translate the internal mission and strategy into design principles, so that the car designers can apply them as design principles during the car development process. Like it is the case for the building industry, aesthetic and cultural considerations may also lead to design principles for car manufacturing. This is the third source of design principles. The collective design principles, both functional and constructional, that are applied in developing a car model, are called the *architecture* of the car model. As said before, we will use the same notion of architecture in developing enterprises and EISs. The only difference may be that in the field of EE aesthetic and cultural considerations play a lesser role.

5. The Generic Requirements and Architecture Framework

In the previous sections, the GSDP has been presented and discussed, including the role of architecture in the system development process. These discussions have brought insight in the nature of system development, regardless the system category, although our main interest has been in enterprise information systems. But it has left open the question how the determination of specific requirements and generic requirements (design principles) can be structured and organised in such a way that it stays manageable. To answer this question, we introduce the Generic Requirements and Architecture Framework (GRAF). The GRAF considers three dimensions for managing the determination of requirements. The first one is the *system kind* that a requirement applies to. Examples of system kinds in EE are B-organisations, I-organisations, and D-organisations. The second dimension is the *design domain* to which a requirement belongs. Building on the systemic foundations, as discussed in [TEEM-2], we distinguish between two main domains: function and construction. Systems can be decomposed in each of these domains. The third dimension we distinguish is the *area of concern* that is addressed by a requirement. Whereas the first two dimensions are quite stable, the third one reflects the, generally time-dependent, focal interests of stakeholders. Examples of areas of concern are security, compliance, privacy, agility, and user-friendliness. They divide the total set of requirements into subsets that correspond to the interests of distinct stakeholders. Note that these subsets need not be disjoint; put differently, areas of concern may overlap. As an example from the automotive industry, the requirement

traffic information must override music

may address the areas of concern “safety” and “user-friendliness”.

We are now able to provide a more precise definition of the notion of requirement: *requirement* is a limitation of the design freedom that regards one system kind and one design domain, while it may address several areas of concern. This definition holds both for specific and for generic requirements (design principles).

Formally, the GRAF can be defined as a tuple <S,D,A> where:

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

Examples of system kinds for the discipline of EE are B-organisations, I-organisations, and D-organisations. So, in a system development process, one would be concerned with an instance of one of these kinds. Regarding the dimension ‘design domain’, the top division consists of function and construction. However, each of these domains may be further divided until the level of detail is reached that one considers to be adequate. A practical guideline for such a subdivision is respectively the functional decomposition and the constructional decomposition that has been made or can be made. Taking the car again as the example system, we have shown a functional decomposition in Figure 3.2 of [TEEM-2] and a constructional one in Figure 3.1 of the same document. In Figures 5.1 and 5.2, these decompositions are repeated. The components are now viewed as functional design domains and constructional design domains respectively.

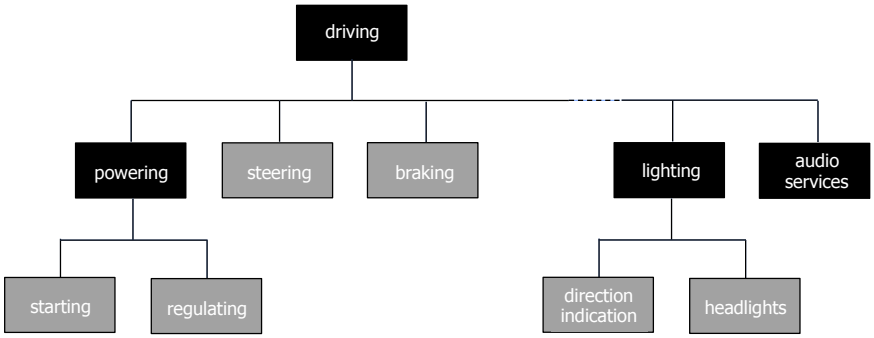

Figure 5.1 Functional design domains of a car

Below, we repeat the example functional requirements for a car from section 4. After every requirement, the system kind (always: car) and the design domain is mentioned:

braking distance within 12 m at 50 km/h	(S = car, D = braking)
acceleration to 100 km/h within 8 seconds	(S = car, D = powering)
day/night controlled headlight switching	(S = car, D = headlights)
traffic information must override music	(S = car, D = audio services)

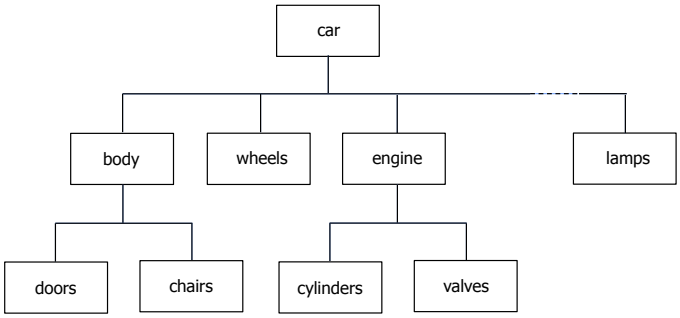

Figure 5.2 Constructional design domains of a car

Below, we present some examples of constructional requirements in the automative industry, also with indications of the system kind and the design domain. Note that one cannot tell from the formulations whether these requirements are specific or generic (so being design principles). For their effect on the construction of the car, the origin doesn't matter.

the total mass of the car must be less than 1500 kg	(S = car, D = car)
side impact protection must be placed in the doors	(S = car, D = body)
grilles must be made of composite matter	(S = car, D = body)
all lamps must be LEDs	(S = car, D = lamps)

A subset of the constructional requirements regard the user-system interface, thus the means for operating the system's 'leaf' functions. In the car example, the *operating requirements* comprise e.g. the accelerator pedal, the brake pedal and the light switches. Here are some examples:

using the brakes must be power assisted	(S = car, D = brake pedal)
all light switches must be in one handle	(S = car, D = light switch)
fuel filling point must be lockable	(S = car, D = body)
steering wheel must be adjustable	(S = car, D = steering wheel)

6. Application of the GRAF to the design of enterprises

In TEEM-2 (the τ-theory) we have discussed the functional decomposition of an enterprise. The example given there (Figure 7.1) is repeated in Figure 6.1 below, but now the components are called functional design domains. For each of them generic requirements (architecture principles) and specific requirements can be formulated.

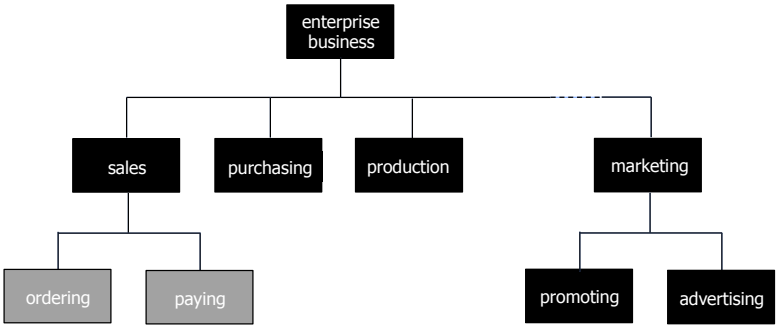

Figure 6.1 Functional design domains of an enterprise

Below, we provide some example functional requirements for an enterprise's business. After every requirement, the system kind (always: enterprise) and the design domain is mentioned:

capability to configure (customised) products	(S = enterprise, D = production)
easy to use ordering and payment services	(S = enterprise, D = sales)
easy to use goods return and refund services	(S = enterprise, D = sales)
low environmental damaging effect of products	(S = enterprise, D = production)

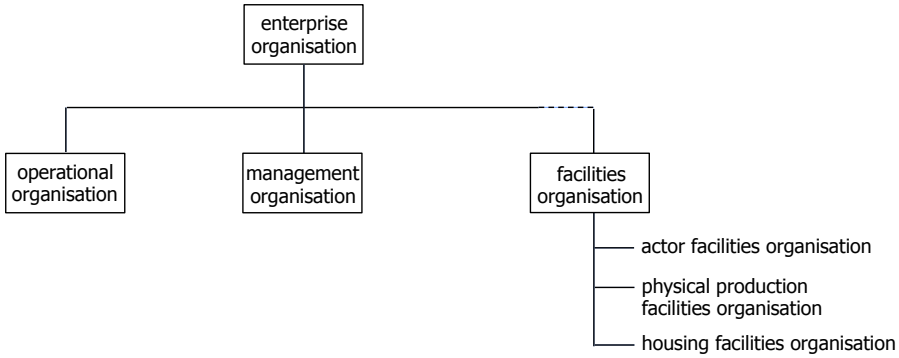

Figure 6.2 Constructional design domains of an enterprise

Below, we present some examples of constructional requirements for organisations, with indications of the system kind and the design domain. Regarding the system kind, we apply the generic partitioning of organisations that is part of the τ-theory [TEEM-2]. It is reproduced in Figure 6.3. Where one of the partitions is mentioned, we mean to indicate the primary partition. Where "organisation" is mentioned without prefix, all partitions (B-, I-, D- and P-organisation) are meant.

real-time workload distribution	(S = B-organisation, D = operational organisation)
possibility to quickly reconfigure processes	(S = organisation, D = operational organisation)
internet-based communication facilities	(S = P-organisation, D = facilities organisation)
percentage of competent managers ≥ 75	(S = B-organisation, D = mgt organisation)

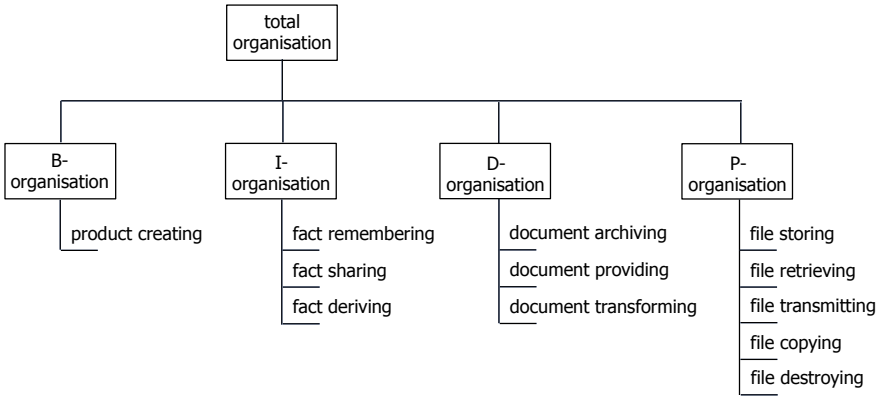

Figure 6.3 Generic partitioning of an organisation in system kinds

As a final remark, we like to emphasize that we consider the elaboration of enterprises in the GRAF as ‘work in progress’. At the same time, we feel confident with the first steps that are made above.

Bibliography

Alexander, C.: *Notes on the synthesis of form* (Ablex Publishing Company, NJ, USA 1960)

Dietz, J.L.G., *Enterprise Ontology*, Berlin, Springer, 2006.

Dietz, J.L.G., *Architecture - building strategy into design*, SDU Publishing, The Hague 2008. Since 2014, it is freely available from www.sapio.nl/publications

Dietz, J.L.G., Hoogervorst, J.A.P.: A critical investigation of TOGAF, in: Advances in Enterprise Engineering V, LNBIP 79, Springer-Verlag 2011. An extended version is freely available from www.ee-institute.org.

Fowler, M., Scott, K., 1999. 'UML Distilled'.

Hoogervorst, J.A.P., *Enterprise Governance and Enterprise Engineering*, Berlin, Springer 2009

Hoogervorst, J.A.P., Dietz, J.L.G.: Enterprise Architecture in Enterprise Engineering, in: Enterprise Modelling and Information Systems Architecture, Vol. 3, No. 1, July 2008, pp 3-11, ISSN 1860-6059

Jacobson, I., Christenson, M., Jonsson, P., Overgaard, G., 1992. 'Object-Oriented Software Engineering: A Use Case Driven Approach'.

Jong, J. de: *A Method for Enterprise Ontology based Design of Enterprise Information Systems*, dissertation TU Delft, 2013

Simon, H.A.: *The Sciences of the Artificial*, Cambridge, 3rd edition, MIT Press 1996

List of TEEMs (Theories in Enterprise Engineering Memorandum)

TEEM-1: the ω-theory (OMEGA: Organisation's Management, Engineering & Governance Appreciation)

TEEM-2: the τ-theory (TAO: Teleology Across Ontology)

TEEM-3: the δ-theory (DELTA: Discrete Event in Linear Time Automaton)

TEEM-4: the ϕ-theory (FI: Fact and Information)

TEEM-5: the ψ-theory (PSI: Performance in Social Interaction)

TEEM-6: the σ-theory (SIGMA: Socially Inspired Governance and Management Advancement)

TEEM-7: the π-theory (PI: Performance in Interaction)

TEEM-8: the β-theory (BETA: Binding Essence to Technology under Architecture)

TEEM-9: the ν-theory (NU: Normalised Unification)