H.A. Proper

Grounded Enterprise Modelling

February 6, 2009

Radboud University Nijmegen & Capgemini

Preface

These are the lecture notes for the ‘Modelleren van Organisaties’ course offered in the spring of 2008 at the Radboud University Nijmegen. These lecture notes are still under development, and are expected to evolve considerably over the next years.

In 2008, the course ‘Modelling of Organizations’ will be taught for the fivth time. In the academic year 2007/2008, a fourth incarnation of the lecture notes will be created in a number of incremental steps. Two important evolutions planned for this year will be:

- The integration of the foundations of work-systems mini lecture notes into these lecture notes, involving the re-writing of the formal aspects retargeting it at a 1st year audience.
- Further elaboration of the Grounded Enterprise Modelling language, in particular the different aspects.

It is needless to say that any feedback from either students or colleagues is more than welcome. Special thanks go out to the students attending the ‘Modelling of Organization’ courses taught in 2004 to 2007.

Prof.dr. H.A. (Erik) Proper
Radboud University Nijmegen
The Netherlands

Contents

Part I Grounding layer

Part II Enterprise Layer

Chapter 1
Introduction

These lecture notes are still very much in development. There are several places that need further elaboration. Sometimes we will even hint at the work that needs to be done. Whenever we make such comments, we will use: {*this is an example comment*}. Rather than hiding these notes from you as the reader, we want to openly share these thoughts with you in the hope you may feel free to contribute your thoughts and opinions to the improvement of these lecture notes.

The focus of these lecture notes is on modelling of different aspects of enterprises. In doing so, we will build on top of the general domain modelling background provided in [101]. Let us start by first exploring the concept of enterprise and enter-prise systems.

1.1 Enterprises and enterprise systems

Enterprises are a ubiquitous phenomenon in our modern daysociety. Most of our lives are spent in the context of enterprises. We are born in *hospitals*, we receive an education from *schools* and *universities*. Later on we work for *factories*, *banks*, *government departments*, etc. In our spare-time, we visit *restaurants*, *sport clubs*, etc. These are all examples of enterprises.

The ubiquity of enterprises is not something new. Alsoin the past enterprises have always been dominantly present. Ever since people and/or animals started to band together to jointly achieve some common goal, enterprises were formed.

According to the Webster-Webster dictionary [64], an enterprise is: *a systematic purposeful activity or a unit of economic organization or activity*. In our definition we will make a mix of these two definitions. When stating " *unit of economic organization or activity*", we would like to generalise this slightly. The focus on economic organization or activity is to strict in our opinion. In line with "*a systematic purposeful activity*" we would like to read this as a "*purposeful organization or activity*", where this purpose might be an economical one (for example, in the case of a commercial enterprise such as a business), but might also be a societal one (for example, in the case of Greenpeace), or a governmental one (for example, Governments of nations of groupings of nations).

Since enterprises involve activity, enterprises can be regarded as (active) systems. They are also likely to be composed of several sub-systems, such as departments, information systems, machines and software systems. We take the perspective that enterprises systems, are work systems. In other words, systems in which actors (humans, animals, machines, etc.) perform work. In Chapter 3, we will provide a more precise definition of system and work systems. For now, we can define enterprises as a special kind of work system:

Enterprise – A work system which performs a systematic and *purposeful* activity.

We will use the term enterprise system to refer to an enterprise as a whole and/or one of its sub-systems:

Enterprise system – An enterprise, being a work system, and/or one of its subsystem.

The many sub-systems of an enterprise derive (part of) their purpose from the purpose of the enterprise as a whole. Having a purpose also implies that the design of these systems should follow rationally from its purpose. This implies that the engineering of such systems can, in principle, be approached in a rational and scientific manor.

1.2 Systems

Even though the notion of system is, specificallyan IT context, often equated to 'software system', the original sense of the word is much broader. The notion of system is also not uniquely defined in the literature, but typically, it can be found explained as: "*A collection of interrelated parts characterized by a boundary with respect to its environment*" [43] or just as: "*A set of objects with a set of links*" [58]. In general, humans refer to all sorts of things as 'systems'. The broadness of our understanding of the concept of 'system' comes, for example, to the fore in the definition as found in [64]:

> *A regularly interacting or interdependent group of items forming a unified whole, as:*
>
> 1. *a group of interacting bodies under the influence of related forces,*
> 2. *an assemblage of substances thatis in or tends to equilibrium,*
> 3. *a group of body organs that together perform one or more vital functions,*
> 4. *the body considered as a functional unit,*
> 5. *a group of related natural objects or forces,*
> 6. *a group of devices or artificial objects or an organization forming a network es-pecially for distributing something or serving a common purpose,*
> 7. *a major division of rocks usually larger than a series and including all formed during a period or era,*

> 8. *a form of social, economic, or political organization or practice.*

The IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [42] provides a functionality-oriented perspective on systems:

> *A collection of components organized to accomplish a specific function or a set of functions.*

In practice, most people intuitively agree on such simple definitions of systems. Apparently these definitions are broad enough to cover the meaning of usual linguistic constructs where 'system' is used. But system is a much more difficult concept. If we look at what in practice are considered systems, and if we really think about it, it becomes obvious that some very important aspects of the system concepts are missing in the traditional definitions. In [23] some examples are given of what we would, and would not, observe to be systems in our daily life:

> *One can regard an organization or a bicycleas systems. Also a Hitchcock film re-corded on a video cassette, which is inserted in a video cassette player, which again is connected to a TV-set, could easily be interpreted as a system. Nothing is unusual with such system views, and they are well covered by the definitions. But if you buy some eggs from a farmer and use two of them for breakfast, then the domain of obviously interrelated phenomena: You, the farmer, the farmers hen that laid the eggs, the frying pan you used to prepare the eggs, and the two eggs now in your stomach (and thereby in some transformed form a part of yourself) – this domain might probably not be regarded as a system, because it might be difficult to see a purpose for that. But it fits the definitions.*
>
> *Or consider a single raindrop in an April shower: It consists of a vast number of water molecules, kept together by surface tension and constantly moving around among each other in a complicated manner controlled by a set of (thermo-) dynamic forces. Again according to the simple definitions above, the drop qualifies as a system. But that is strange, because when you on your way back from the farmer, happen to get soaked in the shower, you might feel it is caused by raindrops – not by systems.*
>
> *On the other hand, a meteorologist studying possible weather situations that could cause rain, may see a purpose in regarding a raindrop as a system in interaction with the surrounding atmosphere, but in most other situations a raindrop is just a rain-drop.*

When looking closely at what is regarded as a system in practice, it becomes apparent that some very important aspects of the system concept are missing from the traditional definitions.

Whatever the definition of enterprise, they are first and foremost systems! To be more pre-cise, they are generally systems that, in addition to processing information, are:

- capable of undergoing (state) changes,
- able to perform actions,
- able to respond to external triggers,

in other words, they are so-called [23] open and active systems. The latter three properties hold for numerous other systems as well. Some random examples include:

- The human nervous system.
- An ant colony.

- A train.
- A school of fish.
- A group of people.

1.3 Information systems

In our modern western society, most enterprises use some form of information systems to support the activities of the enterprise. With ‘information system’ we (informally) refer to information processing activities that may be performed by computerized as well as non-computerized actors. Without these information systems, most enterprises would no longer be able to exist.

Even more, some enterprises are actually large information systems themselves. For example, banks, insurance companies, taxation offices, are really ‘just’ very large information systems comprising human, physical (money, bankcards, etc.) and computerised actors.

The concept of information system can roughly be defined as that aspect of an enterprise that provides, uses and distributes information. An information system may contain computerized sub-systems to automate certain elements. Some information system may not even be com-puterized at all. A filing cabinet used to store and retrieve several dossiers is, in essence, an information system. What we may perceive to be an information system, may indeed vary highly in terms of their scope. Some examples would be:

- Personal information appliances, such as electronic agenda’s, telephone registries in mobile phones, etc.
- Specific information processing applications.
- Enterprise wide information processing.
- Value-chain wide information processing.

Some concrete examples are:

- An insurance-policy administration is an information system.
- A bank is (primarily) an information system.
- Clients are actors in that information system.
- The taxation department is an information system.
- The PDA you use as an agenda.
- The phone number collection in your mobile phone.

In practice, the concept of information system is used quite differently by different groups of people. It seems (see e.g., [23]) to be interpreted in at least three different ways:

1. As a technical system, implemented with computer and telecommunications technology.
2. As a social system, such as an organization, in connection with its information processing needs.

3. As a conceptual system (i.e., an abstraction of either of the above).

1.4 Enterprise engineering & system engineering

{*In this Section we regard enterprise engineering as a form of system engineering. This generalized approach should be explained in more detail. Even more, it should be extended with discussions from the 'Architecture-driven Enterprise Engineering' lecture notes and the GSDP from the xAF [105] documents.*}

Most larger enterprise systems do not appearout of the blue. They need to be developed using some system engineering process. Such an engineering process typically comprises of the following sub-processes:

Definition process – A process aiming to identify all requirements that should be met by the system, the system description, and the engineering process.
In literature this process may also be referred to as requirements engineering.
Where **definition** is defined as:

> The description of the requirements that should be met by both the desired information system as well as the documents documenting this information system. In literature this is also referred to as requirements engineering.
>
> With regards to the information system, the resulting descriptions should identify: *what* it should do, *how well* it should do this, and *why* it should do so. With regards to the documentation of the information system, the descriptions should identify *what* should be documented, *how well* it should be documented, and *why*/*what-for* these documents are needed.

Design process – A process aiming to design a system conform stated requirements. The resulting system design may range from high-level designs, such as an strategy or an architecture, to the detailed level of programming statements or specific worker tasks.
Where **design** is defined as:

> The description of the design of system. These descriptions should identify *how* a system will meet the requirements set out in its definition. The resulting design may (depending on the design goals) range from high-level designs to the detailed level of programming statements or specific worker tasks.

Construction process – A process aiming to realise and test a system that is regarded as a (possibly artificial) artifact that is not yet in operation.

Deployment process – A process aiming to make a system operational, i.e. to implement the use of the system by its prospective users.

One could state that a system definition provides an articulation and motivation of the desired qualities, while the design of a system provides a plan to realize these desired qualities. The constructed system is an, as yet unoperational, materialization of this plan in the real world, while the operational system is the final realisation of the plan.

One might argue that an importantsub-process of an (enterprise) system's life-cycle is miss-ing from the above list of aspects, being that of maintenance. We define maintenance as:

Maintenance – A system which is operational in its usage context, does not remain operational by itself. Both technical and non-technical elements of the system need active maintenance to keep the system operational as is.

However, the design of maintenance procedures, mechanisms and techniques is regarded as an integral part of an (enterprise) system's design. Most enterprise systems are designed to be in existence for longer periods of time. This means that these systems should maintain themselves, in other words, maintenance should be designed 'in'.

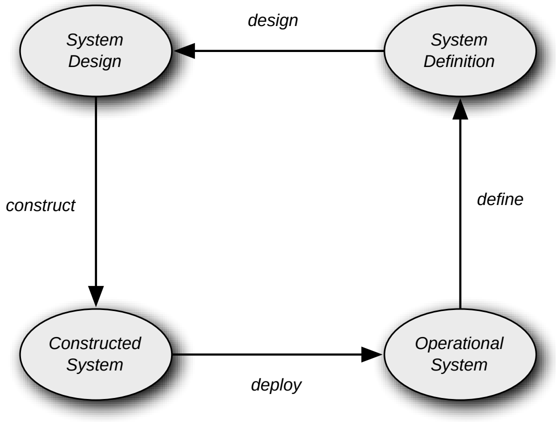


Fig. 1.1 Enterprise engineering

When integrating these processes into a unified process, a first, and naive, way of representing the resulting overall engineering process, would lead to the situation as shown in Figure 1.1. In the situation depicted, it is presumed that some operational work system exists, and that there is a need to change / improve / extend this work system. By means of a definition process, the requirements of this desired work system change can be ascertained. Using these requirements as a starting point, a new system can be designed, leading to a work system design. Based on this design a new system may be constructed, which, after completion, may then be installed as part of the operational work system.

This representation of the development process is, however, naive in two important ways:

1. It suggests a linear flow of activities.

2. It does not distinguish between system domains, conceptions of systems and system descriptions.

The arrows in Figure 1.1 should indeed not be interpreted as putting a requirement on the start of the processes, but rather of the finalization of them. In other words, the processes may quite well run in parallel, however, the definition process should be finalized (if only micro-seconds) before the definition process can be finalized, etc. There are actually three major flavours of development approaches that may be used [25]:

Linear approach – Step by step execution of a (part of a) development process, where a consecutive step is not executed until the preceding step is finished.

Incremental approach – A (part of a) development process is executed on a sub-system by sub-system basis, using some well-defined division of a system into sub-systems.

Evolutionary approach – A (part of a) development process is executed completely in several iterations, leading to several consecutive versions of the set of deliverables.

These flavours may actually be mixed/matched for different parts/stages of the development process. For example, the following recipe may be used for the execution of a project:

Do linearly:

1. Do evolutionary:
   - Definition
   - Design
2. Do incrementally for all top-level component systems:
   - Do linearly:
     a. Construction
     b. Deployment

In general, system engineering can now be defined as:

System engineering – A process involving aimed at producing a changed system, involving the execution of four sub-processes: definition, design, construction and installation of the system. Processes that may be executed sequentially, incrementally, interleaved, or in parallel.

Leading to the following definition of enterprise engineering:

Enterprise engineering – A system engineering process aimed at the creation of an enterprise system.

Two important sub-processes of system engineering are actually still left out of this definition:

Domain modelling – Modelling of the domains that are relevant to the information system being developed. The resulting models will typically correspond to *ontologies* of the domains. These domains can pertain to the information that will be processed by the information system, the processes in which the information system will play a role, the processing as it will occur inside the information system, etc. Understanding (and modelling) these domains is fundamental to the other activities in information system engineering.

Architecting – The processes which tie definition, design and deployment to the explicit and implicit needs, desires and requirements of the usage context. Issues such as: business/IT alignment, stakeholders, limiting design freedom, negotiation between stakeholders, enterprise architectures, stakeholder communication, and outsourcing, typify these processes.

The domain modelling sub-process usually remains "hidden" underneath the other four processes. Since these lecture notes focus on enterprise modelling, we will also explicitly pay attention to the underlying domain modelling process. The architecting (sub-)process is not treated in these lecture notes. This process is the focus of another lecture notes in de DaVinci series [79].

When combined, the six sub-process of enterprise and system engineering lead to the situation as depicted in Figure 1.2.

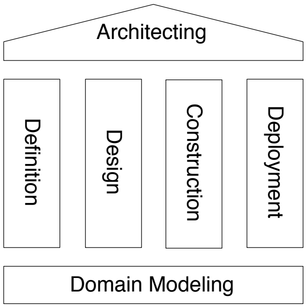


Fig. 1.2 Aspects of Enterprise Engineering

1.5 Enterprise modelling and its role in enterprise engineeering

We consider modelling to be at the very heart of the field of enterprise engineering, informa-tion systems engineering as well as software engineering. Any course on the enterprise modelling should therefore also provide a fundamental understanding of modelling. As we will see in the next Chapter, when two people model the same domain, they are likely to produce quite different models. Even when they use the same information (informants, documents, etc.) to produce the models, the models are still likely to differ considerably. This also means that if two people communicate about the same enterprise system, they are likely to do so with different models of this system in their mind. *Why do these differences occur? What are the origins of these different models? What happens when people produce models?* Questions that beg for a fundamental answer. These lecture notes try to provide some of the answers.

During enterprise engineering several models are created and manipulated. Models may be used in motivating the need for a system (or change of a pre-existing system), ex-press/underpin requirements on a future system, represent design of a system, etcetera. We take the perspective that the prime role of a model, in an enterprise engineering project, is a means of communication about a system being engineered. We take the stance that if some model created during enterprise engineering is not used as a means of communication in some shape or form then this model should not have been created in the first place. Whatever the role of a model, it always involves some communicative aspect.

{*The discussion below should be extended with some examples and illustrations.*}

A model can be used to represent different views on a system. For example:

1. A model may represent the function, construction and foundation of a system.

   a. A model focusing on the function of an enterprise would treat a system as a black-box hiding implementation details of how the function of the system is realized. In doing so, we would focus on the system's behaviour as a mathematical function mapping different inputs in the course of time to some output domain.
   b. A construction model would focus on the way the function of the system is indeed realized, treating the enterprise as a white-box.
   c. Models focusing on the foundation of an enterprise are concerned with those systems and objects which are used as building blocks for the construction.

2. A model may represent the definition (*why* and *what*) of a system or the design (*how* and *what with*) of a system.
3. Models may focus on different types of information about the enterprise system under consideration. For example (based on [85, 17]):

   a. The *essential view* of an enterprise system focusing on services offered to clients, as well as the essential roles and processes needed to realize these services.
   b. The *informational view* of an enterprise system focusing on the processes needed to exchange relevant information and knowledge supporting the essential processes of the organization.
   c. The *technical view* of an enterprise system focusing on the (information) technological implementation of the essential and informational processes of an enterprise system.

4. Enterprise systems involve many aspects, such as: such as the actors involved in the system, the tasks performed by these actors, the objects that are manipulated, the value added by the enterprise to its environment, etcetera. Each of these aspects can be represented as models.

In terms of the above example views on an enterprise system, these lecture notes will cover:

1. both function, construction and foundation views,
2. the design of an enterprise system,
3. the essential and informational perspective,
4. several inter-related aspects of an enterprise system:

    a. *Work flow*: The flow of work from in the system.
    b. *Work roles*: The assignment of work to roles identified in the system, and the (de)composition of these roles.
    c. *Work objects*: The objects which are acted upon by the work performed in the system.
    d. *Work distribution*: The distribution of work over multiple roles in terms of services offered by one role to another, governed by transactions
    e. *Work value*: The value exchanged between roles engaged in services and transactions.
    f. *Work rules*: The rules governing the work performed by/in the enterprise.
    g. *Work agents*: The agents playing active roles in the system, their priorities, competencies, etc.

For each of the above listed aspects, we will introduce one or more modelling techniques. In doing so, we aim to integrate a number of pre-existing modelling techniques covering several (sub) aspects of systems: ArchiMate [59], TestBed [18], YAWL [2] and DEMO [85]. In integrating these techniques we also aim to create a uniform 'look and feel'. In other words, we will not just 'gather' pre-existing modelling techniques but rather integrate them into a coherent whole. The notation we will use is based mainly on the ArchiMate language.

Note that even though we focus on the essential and informational views, we take the position that the identified aspects would apply to the technological view as well. The work agents would in this case, obviously, be technological in nature.

1.6 Grounded enterprise modelling

Since models are a means of communication, it is of the utmost importance that the meaning of these models is properly shared between all parties involved in the communication. What makes this even more challenging is the fact that this communication is quite often of an asynchronous nature. This means that there will be a timelapse between the moment the model is put down on 'paper' and the moment it is 'read'. This timelapse can be from minutes to years.

These lecture notes that the stance that enterprise models should be grounded in a thorough understanding of the concepts of the underlying domain. To this end, we will treat enterprise models as refinements of more general underlying domain models, such as discussed in the domain modelling course [101], expressed in terms of ORM [32] models.

The resulting, integrated, set of modelling techniques is referred to as the Grounded Enter-prise Modelling (GEM) approach. In this modelling technique we identify two layers:

1. A *grounding layer* comprising an ORM-based modelling technique intended to create a domain model of the concepts of the enterprise being modelled.
2. An *enterprise layer* comprising a number of modelling techniques to represent the different aspects (flow, roles, objects, distribution, value, agents and rules) of an enterprise system.

1.7 Structure of this textbook

These lecture notes comprise two parts. The first part is concerned with the grounding layer, where we focus on the foundations of modelling and its role in enterprise engineering, as well as present an ORM-based modelling technique for the creation of domain models for active domains (such as enterprises). The second part is concerned with the enterprise layer, and deals with the aforementioned viewpoints and associated modelling techniques covering different aspects of enterprises.

However, before starting on the first part, we need some more preparations. In discussing the different modelling techniques, we will use meta-models to express syntax and semantics. To this end, however, we need a meta-modelling language. In other words a language in which to express the syntax and semantics of a modelling language.

1.8 Questions

1. What is an enterprise? Give some examples of groupings of people which is not an enterprise.
2. What are the four key sub-processes of enterprise engineering?
3. Produce a model of the hierarchical structure of a university (faculties, departments, schools, etc). Why is the model organized this way? Did you use a specific modelling language to denote this model? If so, why this language?
4. Produce a model of the educational process of attending a course at a university. What are the contributions of the different elements in this process?
5. If two people were to produce a model of the same enterprise. Would you expect them to produce the same model? If not, why do you think these models would differ?

Chapter 2
A meta-modelling language

The aim of this Chapter is to bootstrap the rest of this book. In Chapter 5 we discuss the modelling language ORM [32] as a general domain modelling language. However, in doing so we will already use ORM to precisely define and describe the notions involved. In other words, we will use ORM as a *meta-modelling language* using our understanding of modelling as the domain to be modelled.

{*This Chapter should eventually evolve into a summary of those elements of the ORM and SBVR notation used for meta-modelling purposes in these lecture notes. Currently such an overview is not needed since the "Modelleren van Organisaties" course has domain modelling as a pre-requisite. The use of ORM for meta-modelling purposes is not new. An early extension of ORM, called PSM [37], was actually specifically designed for such a purpose. The associated language LISA-D [36] was used as a rule language. In future versions of the "Modelleren van Organisaties" lecture notes, this will evolve to the SBVR standard [88].*}

2.1 Objectification as abbreviation

Based on [33], we will treat ORM's objectification as an abbreviation. In other words, we will use the abbreviation as suggested in Figure 2.1.

2.2 Questions

1. What is a meta-model?
2. Produce an ORM (meta-)model of the ORM modelling technique as presented in the domain modelling course [101].

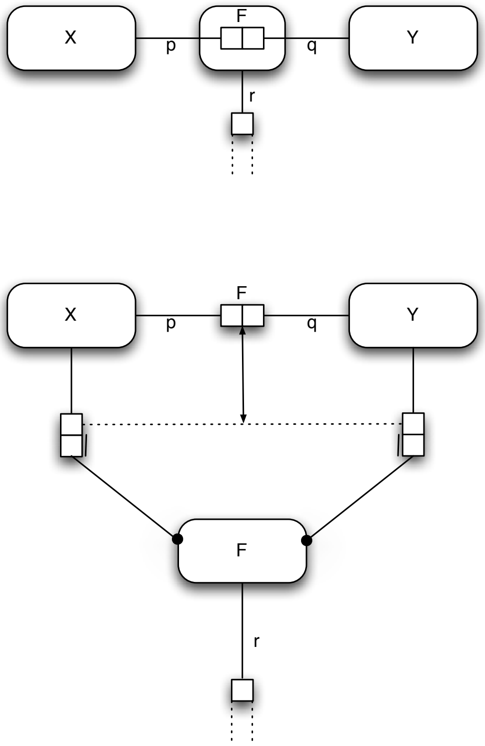


Fig. 2.1 Objectification as abbreviation

Part I
Grounding layer

Chapter 3
Model-driven enterprise engineering

{*In this Chapter we, again, initially regard enterprise engineering as a form of system engineering.*}

3.1 The role of models and modelling in enterprise engineering

As discussed in the introduction, during enterprise engineering several models are created and manipulated. We take the perspective that the prime role of a model, in an enterprise engineering project, is a means of communication about a system being engineered. We take the stance that if some model created during enterprise engineering is not used as a means of communication in some shape or form then this model should not have been created in the first place. Whatever the role of a model, it always involves some communicative aspect. In Chapter 8 we will elaborate further on the plethora of models that may be used during enterprise engineering.

In an engineering system, several actors play a role. These actors are likely to have some stake with regards to the system being developed. Examples of such actors are: problem owners, prospective actors in the future system (such as the future users of the system), domain experts, sponsors, architects, engineers, business analysts, etc.

These actors, however, are not the only "objects" playing an important role in system development. Another important class of are the many different documents, models, forms, etc., that represent bits and pieces of knowledge pertaining to the system that is being developed. This entire group of objects, and the different roles they can play, is what we shall refer to as a *system engineering community*:

System engineering community – A group of objects, such as actors and representations, which are involved in a system engineering process.

3.2 Stakeholder subjectivity in the engineering community

When two people discuss a system, do they really mean the same system? One serious cause for confusion in our professional domain is, that people, usually, think about a system as something that can be objectively determined, for example by a specification of its parts and their relationships, as the above quoted definitions may indicate. But even then, the problem remains. Are both people indeed discussing the same system?

As an example take the simple domain of acar and its driver in the traffic of a city. One person may see it as a useful transport system in action, which is able to move large objects from one location to another in a convenient way. The driver alone cannot, nor can the car, but in combination they can. However, a policeman on his job will regard the same domain differently – as a controllable system which behaviour can be directed by road regulations, traffic lights, arm signals and by certain traffic rules. Again, an environmental activist would probably regard the car as a dangerous polluting system, which is a potential cause of injury or death to persons in the traffic.

Here we have three views of the same domain, but with quite different sets of properties. All three persons could in fact be the same viewer of the same system, e.g. a transport conscious public servant caring about the conditions for people in the city, who just conceives different properties by regarding the same system from different points of view.

Let us elaborate this car example a little further in order to illustrate the difficulties we face when we regard something as a system. Consider for example the question about which parts and which activities are involved in the possible system view: Are the driver and the car two interacting sub-systems – one with the property of being able to observe the traffic and to control the car, and the other with the property of being able to transform chemical energy into movement in a controlled manner. Or is the car to be regarded as a single system with the driver, motor, gear, and steering devices as sub-systems each with their own properties? Is the motor the active part and the chassis a passive component, or is it the other way around – the car as a device transporting among other things the motor. Quite another view – but still one from the same domain – could be to regard the car as a moving cage of Faraday protecting the driver from certain kinds of dangerous electrical fields. There are many possible system views, and still the domain is extremely simple compared with the organizational domains usually considered as systems.

If we regard an enterprise as a system, we have a domain which is much more complicated than a car and driver. Furthermore, the number of possible views of an enterprise is most often enormous.

Key to understanding the system concept, and ultimately organizations is therefore to realize that a system is a subjective phenomenon. In other words, it is not an absolute or objective thing. Systems are not a priori given. As Checkland [15] expresses it, there must be a describer / observer who conceives or thinks about a part of the world as a system. In other words, it is important, that there is a viewer who can see a purpose in regarding some 'set of elements' as a system.

Viewers may also be regarded at an aggregated level. For example, a single business manager, observing an organization, is indeed a viewer, but the collective business management can be seen as a viewer of the organization as well.

The purpose in regarding some set of elements as a system should be expressed in terms of at least one meaningful links between the set of elements and its environment. Such a link is called a systemic property. It is a property the viewer associates with the set of elements they experience as a system. One viewer may regard the set of elements as a system having one set of systemic properties, while another viewer may see other systemic properties concerning the same set of elements.

Most often, the systemic properties of a system cannot be attributed exclusively to any of its constituent components. For example, none of the constituent parts of a train has the exclusive 'train property'. A separate carriage is not a train. A locomotive on its own is (from the perspective of a passenger) also not a train. Together, however, the parts do have the 'train property'. In other words, the whole is more than just the collection of its parts. The farmer-you-frying-pan-eggs-hen situation as discussed in the above example, is a situation which may not constitute a 'whole' with any sensible systemic property. In which case we will not consider it to be a system. In the case of the train, we have an interesting situation if the train consists of two connected train-sets. In this case, each of the individual train-sets still has the 'train property'.

In order for us to gain a fundamental understanding of systems, and ultimately the kind of systems we refer to as organizations, we first need to introduce some core concepts, most of which are based on the ones found in [23] and [42].

When two actors in an engineering community communicate about aspects of an enterprise being engineered, then they are likely to do so from their own personal perspective. These actors are, for instance, likely to have some stake with regards to the system being developed. An example stake of a business manager would be:

I want to be able to rapidly introduce new product to our market.

Stereotypical examples of such actors are: problem owners, prospective actors in the future system (such as the future users of the system), domain experts, sponsors, architects, engineers, business analysts, etc. The actors in an engineering community who have a stakes with regards to subject system are referred to as *stakeholders*:

Stakeholder – Some actor in a system engineering community with a specific stake pertaining to a system's development, its operation or any other aspects that are critical or otherwise important.
Examples are: Users, operators, owners, architects, engineers, testers, project managers, business management, ...

Such a stake typically comes forward from an underlying goal of the actor. These goals are the stakeholder goals:

Stakeholder goal – The end toward which effort is directed by a stakeholder, in which the system (of which the stakeholder is indeed a stakeholder) plays a role.

This may pertain to strategic, tactical or operational end. The role of the system may range from passive to active. For example, a financial controller's goal with regards to a future/changed system may be to control engineering costs, while the goal of users of the system may be to get their job done more efficiently.

The stakeholder goal underlying "*I want to be able to rapidly introduce new product to our market*" might for instance be:

> *I want my business to be able to survive, and preferably utilize, changes in its socio-economic environment.*

The stake of a stakeholder is typically formulated in terms of the objectives of the stakeholder (*I want to*) in which the stakeholder expects an influence (positive or negative) by enterprise being engineered/changed. When re-formulated in terms desirable properties of the enterprise (*Will it be/do/have ...?*) we end up with *concerns* of these stakeholders with regards to the enterprise:

Concern – A stakeholder's interest in properties of a system, relative to their stakeholder goals and the potential role played by the system in achieving these goals. This usually pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

{ *The above discussion should really be supported by some diagram, and maybe even a small ORM model positioning the concepts.* }

3.3 Consequences of subjectivity

{ *The discussion in this Section certainly is relevant, but needs better embedding in the line of the Chapter.* }

The important role of communication and models in system engineering demands a further exploration of the issue of subjectivity in communication. In doing so, we will need to introduce a framework describing the essential processes that take place when a *viewer* (such as a stakeholder) observes a *domain* (such as a system being developed).

The subjective nature of systems is probably the main cause of the serious problems recognized in practice in work systems engineering. Conceiving something as a system may appear as relatively simple for a single person as long as the system will be kept only in his mind. However, it is absolutely not a trivial task for the system viewer to be explicit about it, such that other persons can understand it. And exactly that is the core of the problem, because the process of *developing a system* nearly always involves the expertise of several – often many – persons who may have quite different organizational and professional background and who not necessarily will be fully aware of all the concepts relevant in system engineering. Thereby the practical problem of establishing inter-subjectivity among the involved persons about the system may easily become quite overwhelming.

The main reason is that for a very long period each of the involved persons will have their own personal view of what the system in question is, and these views may be different. But adding to the problem is that people are not always aware of the need of a long and intense communication period as a part of the project. Most often without reflecting about any uncertainty, each person will simply assume that the system they collectively aim at is exactly the one covered by the person's own view.

Even if everybody is conscious of the diversity of system views, in the system engineering process they must still collectively overcome three kinds of uncertainties:

- What are the elements of the system domain?
- What are the systemic properties?
- In which way do the elements contribute to the systemic properties?

Ideally, in order to co-ordinate their efforts such that everybody finally works in the same direction – on the same system – each of the system viewers must express themselves unambiguously about their own individual system, and each one must try to understand the other viewers' systems. Furthermore, as the mutual understanding of the different systems (hopefully) grows, they must aim at and finally agree on a synthesis – a single common view – the system.

In the long and enduring communication process that is necessary to achieve this goal, the only means of the involved partners is mutually to produce, distribute, read and understand (partial) system representations expressed in some commonly accepted and usually system-type-dependent system language. Regardless of whether the representations are expressed orally, on paper or on a screen by means of some tool, and independent of whether it is expressed as text or by graphical means – being explicit about the systems by means of system representations is the only way the involved system viewers can unite their views.

To succeed in the exchange of system views and thereby ending with a single inter-subjectively shared system requires knowledge on two abstraction levels and the corresponding linguistic means for expressing it:

- Subject system (type) specific knowledge.
- Engineering system specific knowledge.

The knowledge specific for the subject system type comprises insight with all the kinds of things and corresponding concepts, which are relevant to consider for all potential instances of that system type, and an according terminology necessary to express oneself properly about the assumed system. This kind of knowledge is a part of the qualifications required for professional systems. It serves as a kind of template for the system engineering work, and it constitutes the basis for the professional language that is used by systems.

The knowledge specific for the engineering system is knowledge of the concepts and kinds of things and of the (local) professional terminology that is relevant to, or used in, the particular engineering system

3.4 What is modelling?

Given the above discussions, we are now able to more precisely define what we mean by modelling.

3.4.1 Observing a universe

Let us first consider what happens if some viewer observes ‘the universe’. It is our assump-tion, based on the work of C.S. Peirce [73, 74, 75, 76], that viewers perceive a uni-verse and then produce a *conception* of that part they deem relevant. The conceptions harboured by a viewer by nature cannot be communicated about or discussed with other viewers unless they are articulated somehow (the need for this ability in the context of system development is evident). In other words, a *conception* needs to be *repre-sented*. Peirce argues that both the perception and conception of a viewer are strongly influenced by their interest in the observed universe. This leads to the following (necessarily cyclic, yet irreflexive) set of definitions:

Universe – The ‘world’ under consideration.
Domain – Any ‘part’ or ‘aspect’ of the universe a viewer may have an interest in.
Viewer – An actor perceiving and conceiving (part of) a domain.
Conception – That what results, in the mind of a viewer, when they interpret a perception of a domain.
Description – The result of a viewerviewer denoting a conceptionconception, using some language to express themselves.

{ *Note: in the next Section the notion of domain is discussed in more detail. This needs to move here, and the domain should be drawn inside the universe part of the diagram in Figure 3.2.* }

The underlying relationships between viewers, universe, conceptions and descriptions can depicted as shown in Figure 3.1.

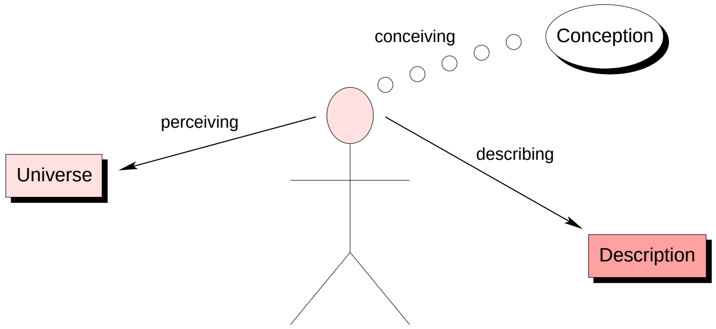


Fig. 3.1 A viewer having a conception of the universe

These distinctions are based on the so-called FRISCO tetrahedron [23], as depicted in Figure 3.2. The FRISCO tetrahedron is based on the work by C.S. Pierce, who made a distinction between an *interpretant*, *representamen* and *object*. This work can, on its turn, be traced back to Ogden's Semiotic Triangle (aka Semantic Triangle). See a see also Figure 3.3, taken from [66].

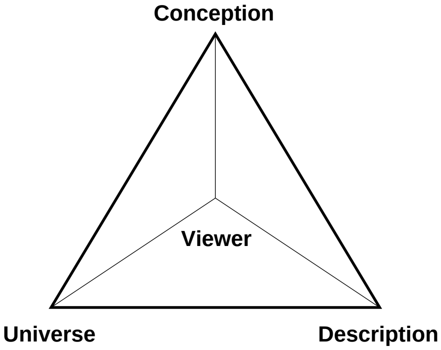


Fig. 3.2 The FRISCO tetrahedron

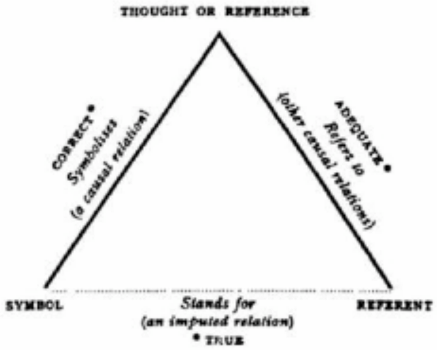


Fig. 3.3 Semantic triangle

One of the key innovations of the Meaning of Meaning [66] was the differentiation between three separate dimensions:

1. The conceptual domain – thoughts that are in our minds.
2. The symbolic domain – words and symbols that we use to communicate with others.
3. The real world – things in the real world that we refer to in our thoughts and with symbols.

The semantictriangle prompted Jan Dietz in his book [17] to create a triangle of systems as illustrated in Figure 3.4.

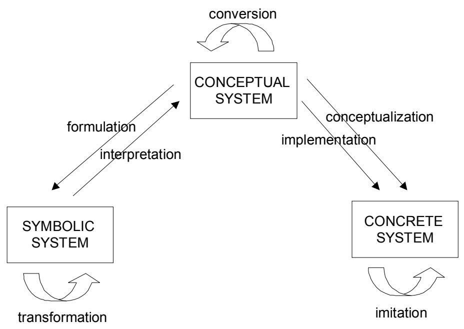


Fig. 3.4 Triangle of systems

3.4.2 Observing under the influence of concerns

In conceiving a part of the universe, viewers will be influenced by their particular interest in the observed universe. In the context of system development, this corresponds to the above discussed notion of a *concern*. Note that viewers, as well as their concerns, may be regarded at an aggregated as well as at an individual level. For example, a *single* business manager conceiving an information system is a viewer. The *collective* business management, however, can also be seen as a viewer of the information system.

Yet concerns are not the only factors that influence a viewer's conception of a domain. Another important factor are the pre-conceptions a viewer may harbour as they are brought forward by her social, cultural, educational and professional background. More specifically, in the context of system development, viewers will approach a domain with the aim of expressing the domain in terms of some set of meta-concepts, such as classes, activities, constraints, etc. The set of meta-concepts a viewer is used to using (or trained to use) when modelling some (part of a) domain, will strongly influence the conception of the viewer. This can be likened to the typical situation of having a 'hammer' and considering all pointy objects to be 'nails'. We therefore presume that when viewers model a domain, they do so from a certain perspective; their *weltanschauung* (German for "view of the world") [104].

In general, people tend to think of the universe (the "world around us") as consisting of related *elements*. In our view, however, presuming that the universe consists of a set of elements already constitutes a subjective choice, made (consciously or not) by the viewer observing the universe. The choice being made is that "elements" (or "thing") and "relations" are the most basic concept for modelling the universe; the most basic weltanschauung. In this book, we will indeed make this assumption, and presume that a viewer's *conception* of the universe consists of elements. The identification of elements in the universe remains relative to viewers and their *own* conception.

3.4.3 Modelling domains

Viewers may decide to zoom in on a particular part of the universe they observe, or to state it more precisely, they may zoom in on a particular part of *their* conception of the universe. This allows us to define the notion of a domain as:

Domain – Any 'part' or 'aspect' of the universe a viewer may have an interest in.

In the context of (information) system development, we have a particular interest in unambiguous abstractions from domains. This is what we refer to as a *model*:

Model – A purposely abstracted domain (possibly in conjunction with its environment) of some 'part' or 'aspect' of the universe a viewer may have an interest in.
For practical reasons, a model will typically be consistent and unambiguous with regards to some underlying semantic domain, such as logic.

For practical reasons, a model will typically be consistent and unambiguous with regards to some underlying semantic domain, such as logic. Note that both the domain and its model are *conceptions* harboured by the same viewer. We are now also in a position to define more precisely what we mean by modelling:

Modelling – The act of purposely abstracting a model from (what is conceived to be) a part of the universe.

For practical reasons, we will understand the act of *modelling* to also include the activities involved in the *representation* of the model by means of some language and medium. Note that the above definition of modelling is broader than the colloquial use of the word in the context of system development.

3.4.4 The role of meta-models

We presume a viewer not only to be able torepresent (parts of) their conceptions of the universe, but also to be able to represent (parts of) the perspectives they use in producing their conception of the universe. This requires viewers to be able to perform some kind of self-reflection. When modelling a domain in terms of, say, UML class diagrams [13], the viewer/modeller is presumed to be able to express the fact that they are using classes, aggregations, associations, etc., to view the domain being modelled. In doing so, the viewer essentially needs to construct a conception of her perspective on the world; i.e., a meta model. This meta-model comprises the meta-concepts and modelling approach used by the viewer when modelling a domain; it is a model of the viewer's perspective. Such a meta-model can in essence be regarded as a 'high level ontology' [53].

In Figure 3.5 we depict a situation where a viewer is confronted with a number of domains $(W_1,...,W_n)$. Each of these domains may be modelled from the perspective of the viewer's concern $\underline{C}$ and meta model $\underline{M}$, leading to as many domain-models $(D_1,...,D_n)$. The concern, the meta-model, and the domain models can be represented using some language and medium, leading to representations $C,M,D_1,...,D_n$.

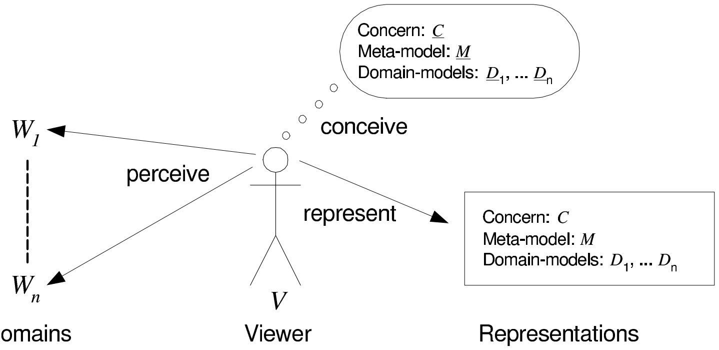


Fig. 3.5 A viewer viewing domains from a particular concern and meta-model

A viewer may also consider a specific domain W from the perspective of some concern C, using two different meta models M_1 M_2. This situation is illustrated in Figure 3.6 where a viewer models a domain D from the perspective of meta models M_1 and M_2, leading to domain models D_1 and D_2 respectively. For example, when viewing a domain from the perspective of UML class diagrams, this is bound to lead to a different domain model than when the same domain is viewed from the perspective of UML sequence diagrams.

If a viewer observes a domain D on the basis of the same meta-model M, but from the perspective of different concerns C_1 and C_2, it is also quite likely that the viewer will produce different domain models, each catering to the specificities of the

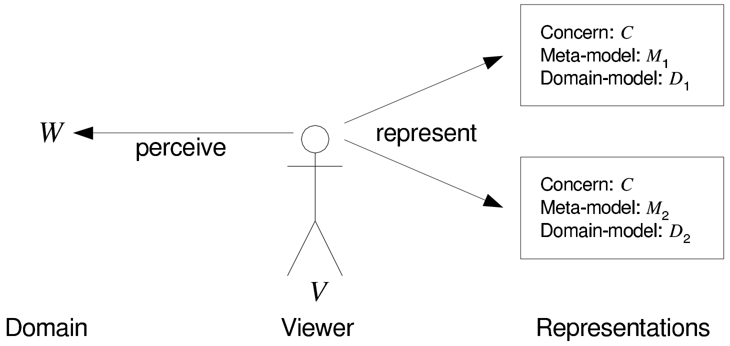


Fig. 3.6 A viewer viewing a domain from the perspective of two different meta-models

respective concerns. Consider, for example, a concern focusing on the functionality offered by a system to its users, versus a concern focusing on the impact of the system on the efficiency of business processes. Given two different concerns, it is also likely tha t questions underlying these concerns cannot be met by using a one-size-fits-all meta model. For example, the operators who will be required to maintain a planned information system, will regard this system in terms of costs of keeping the system up and running, costs and efforts involved in implementing the system, etc. Future users of the same planned system, however, will be more interested in the impact the system is likely to have on their work related tasks, or support provided by it. This implies that when modelling a system (being designed/developed), different meta models need to be used to address different concerns.

3.5 The process of modelling

In this Section we take a first look at the process of modelling. In Chapter 7 we take a closer look at modelling processes.

If we take the perspective that modelling is the act of purposely abstracting a model from a part of the universe, then this *purpose* really forms the start of the modelling process. This purpose provides the *why* of a modelling process. From this why and the context in which the modelling will take place, the requirements, the *what* of the modelling process, can be derived. These requirements should answer such questions as:

- What should the model be about?
- What is the intended audience of the model?
- What level of precision/formality is required?

To better understand and compare methods of information systems development, in [89, 102] a framework was proposed to dissect such a method into a number of aspects. Based on this framework we propose the following framework for system engineering methods. A system engineering method is regarded as comprising the following six aspects:

Way of thinking – Articulates the assumptions on the kinds of problem domains, solutions, engineers, analysts, etc. This notion is also referred to as *die Weltanschauung* [91, 104], *underlying perspective* [62] or *philosophy* [7].

Way of working – Structures (parts of) the way in which a system is engineered. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

Way of delivering – The languages, conventions and documentation standards used in producing deliverables during system engineering.

Way of controlling – The managerial aspects of system engineering. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [52, 92]).

Way of learning – The process and measures that enable continuous improvement of consecutive executions of the method. It should provide answers to questions such as: *How can we learn from past experiences? How can the method be refined to reflect new experiences?*

Way of supporting – The support to system development that is offered by (possibly automated) tools. In general, a way of supporting is supplied in the form of some computerised tool (see for instance [63]).

{ *The way of supporting reference to McClure is indeed still valid, but should be updated. The advent of MDA/MDSD (Model-Driven Architecture / Model-Driven System Development) sheds new light on these tools.* }

The way of learning was not present in the original framework. However, it only makes sense for organizations engaged in system engineering to evaluate their experiences and improve their work practices, i.e. improve their engineering methods. In the field of software engineering this has led to the so-called capability and maturity model (CMM) [72], where the so-called "*optimising*'' level is regarded as the highest level of maturity. At this level organisations engaging in software engineering are expected to continually improve their engineering processes enabled by quantitative feedback from the process and from piloting innovative ideas and technologies. The resulting framework is illustrated in Figure 3.7.

Let us now return to (enterprise) modelling processes. We regard a modelling method essentially as a specialized system engineering method, in which the *way of delivering* is specialised to a *way of modelling* consisting of a *way of describing* and a *way of conceiving*:

Way of modelling – Identifies the *core concepts* of the language that may be used to denote, analyze, visualize and/or animate system descriptions.

Way of conceiving – A set of modelling concepts by which viewers are to observe domains. This usually takes the form of a meta models.

Way of describing – The medium and 'notations' used to represent the concepts as identified in a way of conceiving. It describes how the abstract concepts from

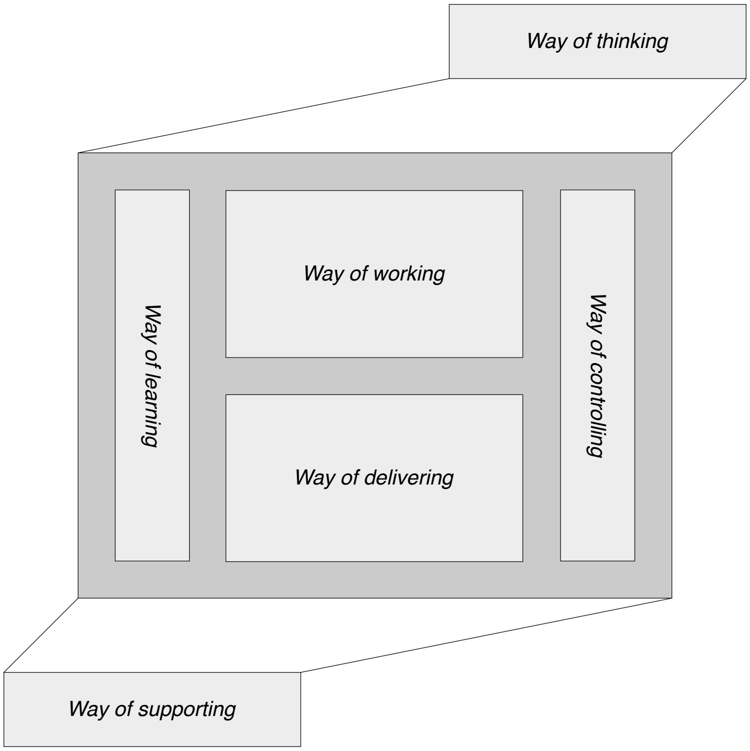


Fig. 3.7 Aspects of a system engineering method

the way of conceiving are communicated to human beings, for example in terms of a textual or a graphical notation.
Note that it may very well be the case that different modelling techniques are based on the same way of conceiving, yet use different notations.

The resulting framework is shown in Figure 3.8. As synonyms, one may refer to a way of working as a *(modelling) approach* and to a way of modelling as a *(modelling) technique*.

In these lecture notes, the primary focus will be on a fundamentally underpinned *way of thinking*, *way of conceiving* and a uniform *way of describing* for several aspects of enterprise systems: work flow, work roles, work objects, work distribution, work value and work agents.

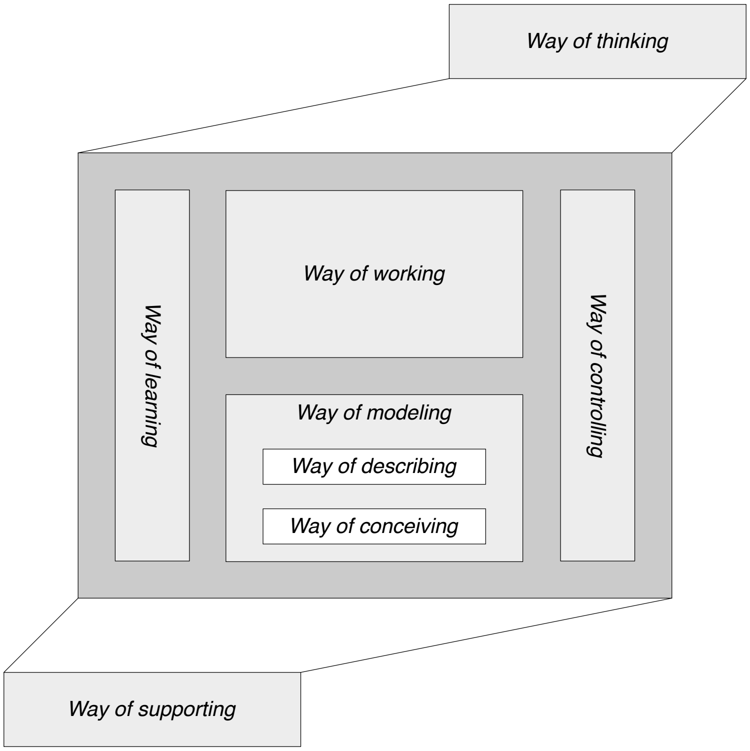


Fig. 3.8 Aspects of a modelling method

3.6 Questions

1. Could engineering projects take place without the use of models? How would people communicate about the requirements and the design of the artefact being designed?
2. Why is it sensible to make a distinction between a way of describing and a way of conceiving? Give examples.
3. Why does it make sense to compare methods based on frameworks as discussed above?
4. Produce an ORM model positioning the concepts of viewer, stakeholder, concern, conception, universe, domain and description.

Chapter 4
Foundations of domain modelling

The aim of this Chapter is to fundamentally look at modelling as the creation of a conception by a viewer. We start by building a meta-model of the core concepts which were discussed in the previous Chapter. In doing so, we aim to make explicit the assumptions one makes about the world in terms of one's meta-model. Furthermore, we take the stance that any refinement to one's meta-model of the world should have a clear utility [83]. In other words, it should be motivated in terms of its (potential) use to us in an enterprise engineering context.

4.1 From conceptions to systems

4.1.1 Conceptions revisited

In the previous Chapter we already defined:

Universe – The 'world' under consideration.
Domain – Any 'part' or 'aspect' of the universe a viewer may have an interest in.
Viewer – An actor perceiving and conceiving (part of) a domain.
Conception – That what results, in the mind of a viewer, when they interpret a perception of a domain.
Description – The result of a viewerviewer denoting a conceptionconception, using some language to express themselves.

In Figure 4.1 we have depicted the meta-model of what happens if some viewer observes a domain. What is missing from this diagram are the notions of domain and description. These will surface later in this Chapter. Most importantly, one should realise that stating that a universe consists of domains, in which a viewer may have an interest, already constitutes an assumption about the universe and conceptions that may be harboured by a viewer.

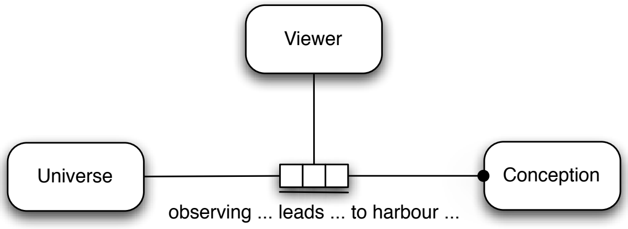


Fig. 4.1 A viewer observing a universe, leading to a conception

A first refinement to the situation depicted in Figure 4.1 is the assumption that a conception consists of elements. In other words, we assume that conceptions harboured by viewers comprise of sets of elements:

Element – The elementary parts of a viewer's conception.

Without this assumption, conceptions remain black boxes which can hardly be discussed with other viewers, other than stating that "it exists". In the context of enterprise engineering this would be highly impractical. The resulting refinement is shown in Figure 4.2.

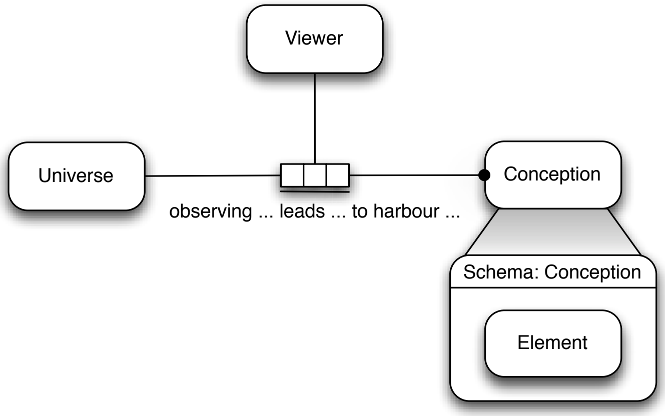


Fig. 4.2 Conceptions consist of elements

This situation is depicted more graphically in Figure 4.3. A viewer, when observing a domain, draws a picture of the observed universe (their conception). In painting this picture of the world they will use certain 'constructs'; the elements.

{*One should actually regard the viewer as a viewer in a particular state. A viewer may for example have, in differing states, different interests with which they con-*

ceive the universe. Since we do not do make this distinction, viewers can have multiple conceptions for the same universe according to our meta-model. Even more, the fact that viewers can change their conception over time is also ignored for the moment.}

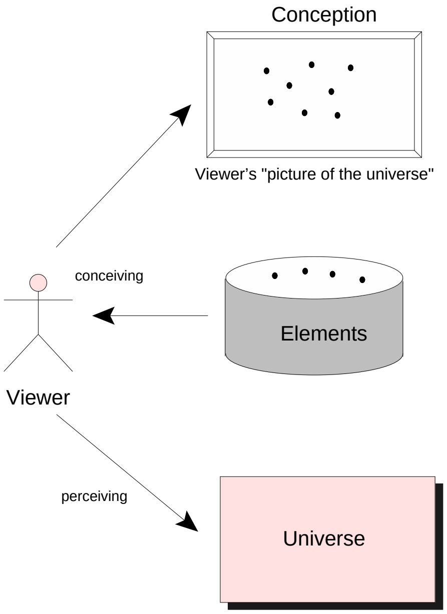


Fig. 4.3 Painting a picture of the universe

As discussed before, a viewer may zoom in on a particular part of the universe they observe, or to state it more precisely, they may zoom in on a particular part of their conception of the universe:

Domain – Any ‘part’ or ‘aspect’ of the universe a viewer may have an interest in.

When reasoning about systems, which we will regard as a particular class of domains, it is commonplace to also identify their environments [11]. Even more, the very definition of a system depends on our ability to distinguish it from its environment.

Environment – The environment of a domain is that part of a viewer’s conception of a universe, which has a direct link to the domain.

This resulting situation is illustrated in Figure 4.4.

To be able to define the environment of a domain in general and a system in particular, however, we must first be able to define the direct environment of a domain. A domain, and its environment, can best be regarded as subsets of the elements in a conception. This leads to the refinement of our meta-model as depicted in Figure 4.5.

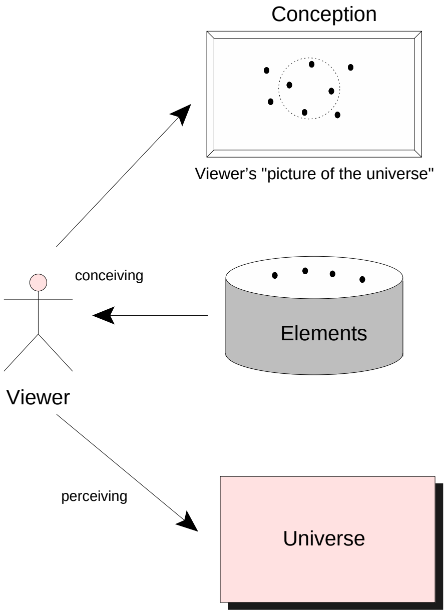


Fig. 4.4 Identifying a domain and its environment

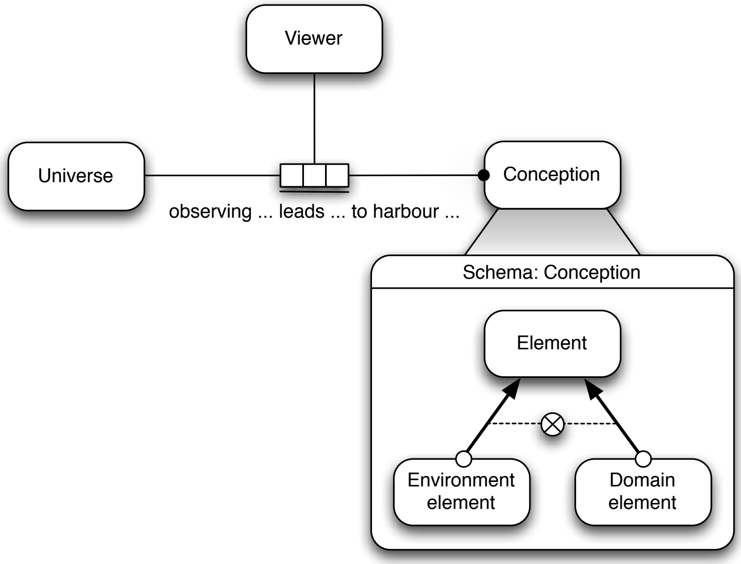


Fig. 4.5 Identifying environment and domain elements

Note that there can be elements, which are neither in the environment nor in the domain. However, for an element in the environment of a domain, it must be somehow linked to the domain. Otherwise, we cannot really reason about the relation between a domain and its environment. In order to more precisely define the notions of domain and environment, we should therefore first refine our notion of elements in a conception. There are really two types of elements: concepts and links connecting the concepts. We will define these notions as follows:

Concept – Any element from a conception that is not a link.
Link – Any element from a conception that relates two concepts.

The distinction between a link and a concept for the elements of a given conception, may not always be that clear, as the distinction is rather *subjective*. It all depends, to no surprise, on the viewer of a domain. In terms of Figure 4.4, our viewer can now select from two classes of elements: concepts and links. This is depicted in Figure 4.6. In the remainder of these lecture notes, we will provide an even more refined view on the classes of elements we identify.

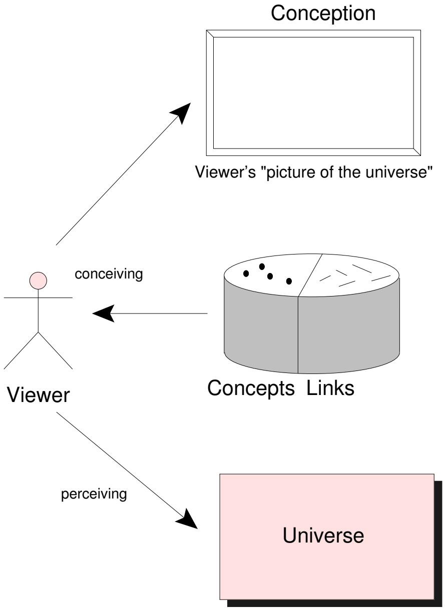


Fig. 4.6 Painting a more refined picture of the observed domain

The resulting meta-model after this latest extension is depicted in Figure 4.7. The derived fact types (marked by a $*$) are defined as follows:

an Element is reachable from **another** Element **iff**
  the latter Element is reachable from **the former** Element **or**
  the former Element is reachable from **some** Element
    which is reachable from **the latter** Element **or**

```
  **the former** Element **is a** Link from **the latter** Element **or**
  **the former** Element **is a** Link to **the latter** Element

**a** Domain element is reachable from **another** Domain element **iff**
  **the latter** Domain element is reachable from **the former** Domain element **or**
  **the former** Domain element is reachable from **some** Domain element
    **which** is reachable from **the latter** Domain element **or**
  **the former** Domain element **is a** Link from **the latter** Domain element **or**
  **the former** Domain element **is a** Link to **the latter** Domain element
```

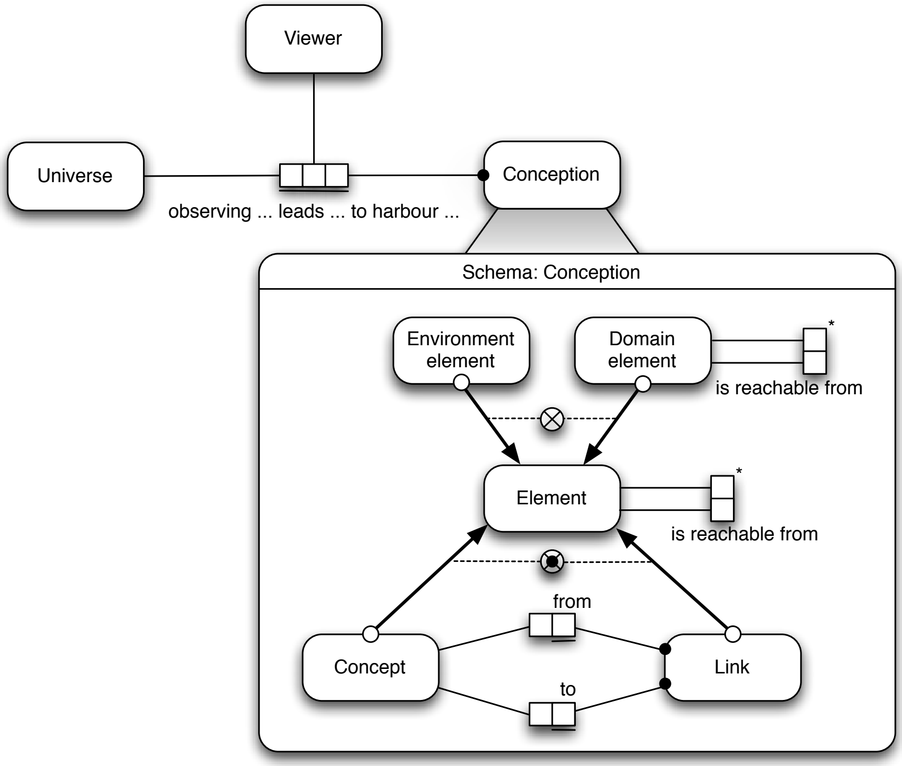


Fig. 4.7 Identifying concepts and links

A conception is required to be a connected graph:

```
**within** a Conception:
  **for each** Element **pair**: **the first** Element **is reachable from** **the second** Element
```

In our view, allowing conceptions to be disconnected graphs would entail allowing one conception to deal with multiple domains. Note: if a conception would not be a connected graph, it would be a conception of multiple universes.

A domain should be *closed*. With *closed* we refer to the fact that each source and destination of a link in a domain should also be in the domain. In other words:

```
within a Conception:
  if a Domain element is a Link ( to or from ) some Concept
  then that Concept is a Domain element as well
```

The same holds for the environment:

```
within a Conception:
  if an Environment element is a Link ( to or from ) some Concept
  then that Concept is an Environment element as well
```

A domain, on its own, should be a connected graph as well:

```
within a Conception:
  for each Domain element pair: the first Domain element is reachable from the second Domain element
```

The environment, on its own, does *not* have to be a connected graph! Since a conception as a whole is required to be connected, disparate "pockets" in the environment of a domain will be connected to each other by way of their connections to the domain.

{*Some explicit examples are needed. Also involving a domain, its environment, and the bridges.*}

The authors of [23] also define the notions of domain and environment. However, they do not take the subjectivity with regards to viewing the universe as a set of elements into consideration. As a result, they define domain and environment as being parts of the *universe* as opposed to being parts of a viewer's conception of the universe.

4.1.2 Decomposition of conceptions

When a viewer conceives a domain, we presume there to be an concept in their conception representing the *whole* of the domain as well as one representing the *whole* of the environment. The same applies to the universe. In other words, the concepts in the domain and the environment can be regarded as decompositions of entities representing the whole of the domain and environment, while these latter concepts are decompositions of another concept representing the universe as a whole. This is illustrated in Figure 4.8.

This 'decomposition game' can be played repeatedly. When viewing a domain a viewer may decide to zoom in further into a specific part of this domain. For example, when observing an insurance claim-handling process, involving amongst

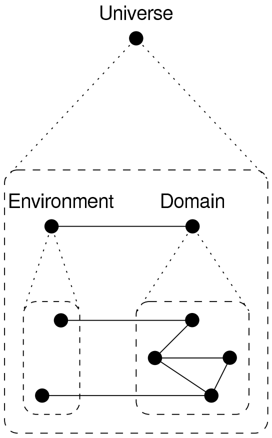


Fig. 4.8 Decomposition of the universe

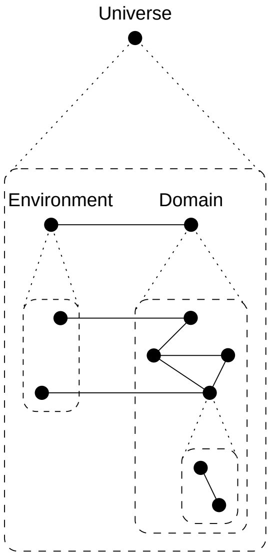


Fig. 4.9 Decomposition of a part of a domain

other things an evaluation of the claim, one may decide to zoom in closer into the actual evaluation process. This has been illustrated in Figure 4.9.

The fact that one concept is in the 'decomposition' of another concept really means that there is a link between them in the viewer's conception. This has been illustrated in Figure 4.10.

To be able to decompose conceptions, we need to identify a specific class of links called decomposers. This leads to the meta-model refinement as depicted in Figure 4.11. To enforce well-formedness of conceptions, we need some extra derived

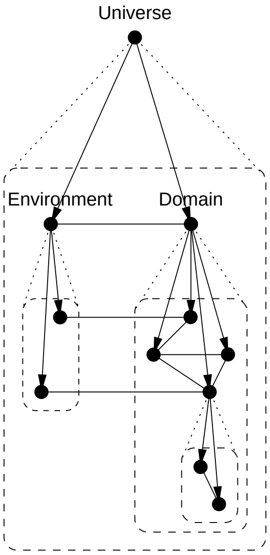


Fig. 4.10 Decomposer relationships

fact types in the meta-model. The needed extensions are provided in Figure 4.12. The derived fact types (the ones marked with a *) are defined as follows:

a Concept is an immediate composite of **some** Concept **iff**
  some Decomposer **exists** [to **the first** Concept] from **the second** Concept

a Concept is a composite concept **iff**
  some Concept **exists which**
    is an immediate composite of **the first** Concept

a Concept is in the composition group of **some** Concept **iff**
  the first Concept is an immediate composite of **the second** Concept **or**
  the first Concept **is the same as the second** Concept

a Concept is a composite of **some** Concept **iff**
  the first Concept is an immediate composite of **the second** Concept **or**
  the first Concept is a composite of **some** Concept
    which is a composite of **the second** Concept

a Concept is at the top of the conception **iff**
  each other Concept is a composite of **that** Concept

a Concept is at the top of the environment **iff**
  it is an Environment element **and**
  each other Environment element is a composite of **that** Concept

a Concept is at the top of the domain **iff**
  it is a Domain element **and**
  each other Domain element is a composite of **that** Concept

a Link is non decomposing **iff**
  it is a Link **which is not** a Decomposer

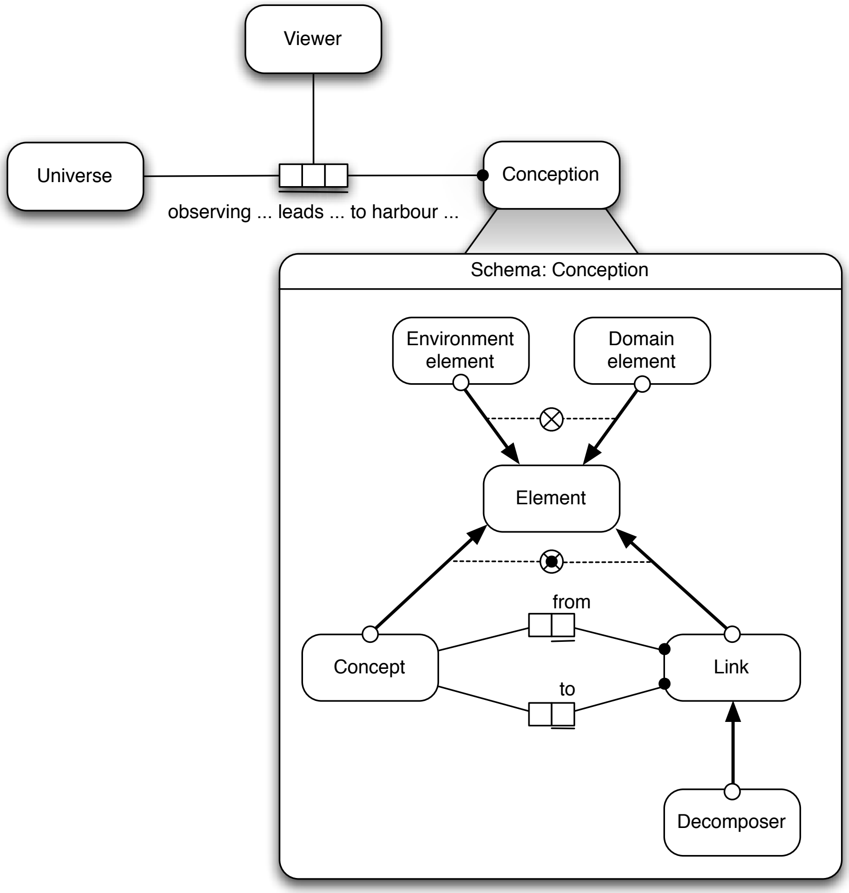


Fig. 4.11 Adding decomposers

a Concept is level linked to **some** Concept **iff**
  some Link **exists** [from **the first** Concept] [to **the second** Concept]
  which is non decomposing

Using these derived fact types, we can now more easily denote properties of decompositions. Decompositions should be acyclic. In other words:

within a Conception:
  if a Concept is a composite of **some** Concept
  then the former Concept **is not equal to the latter** Concept

Decompositions should not cross the borderline between the environment and the domain:

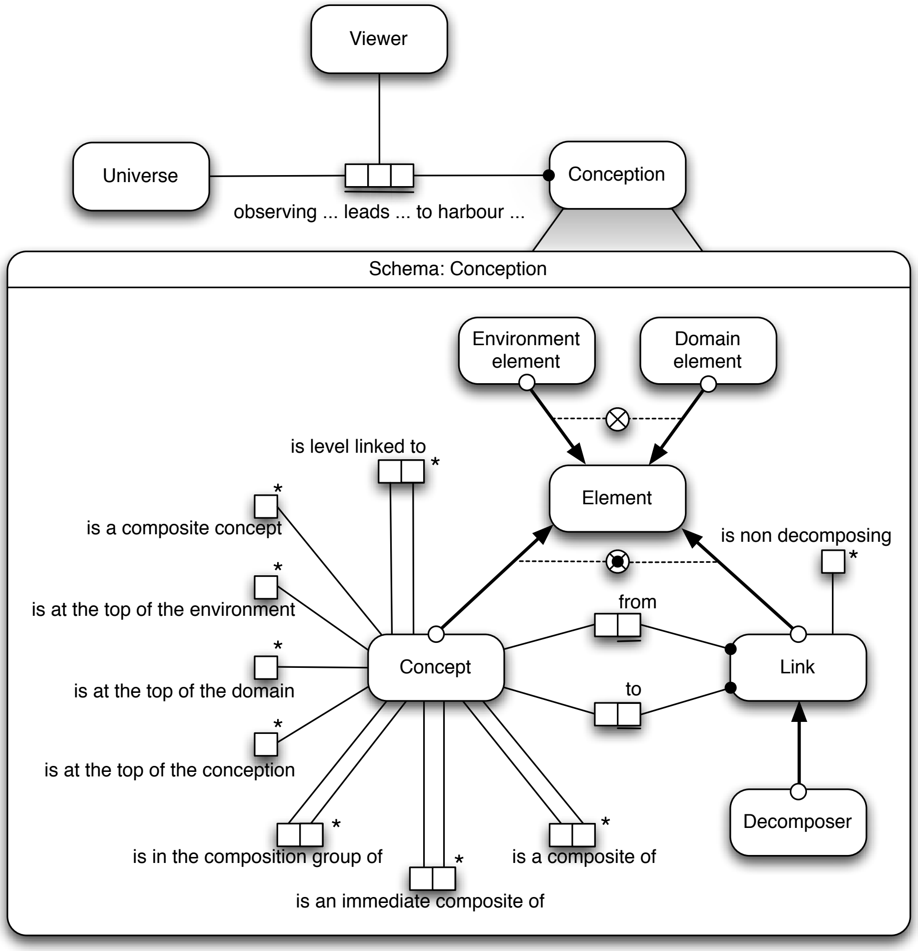

Viewer
Universe
Conception
observing ... leads ... to harbour ...
Schema: Conception
Environment element
Domain element
Element
is level linked to
is a composite concept
is at the top of the environment
is at the top of the domain
is at the top of the conception
is non decomposing
from
to
Concept
Link
Decomposer
is in the composition group of
is a composite of
is an immediate composite of
*

Fig. 4.12 Meta-model with derived fact types

within a Conception:
  if a Concept is a composite of **some** Environment element
  then that Concept **is an** Environment element

within a Conception:
  if a Concept is a composite of **some** Domain element
  then that Concept **is** a Domain element

A viewer's conception of a universe consists of one 'top' concept representing the universe as a whole. To enforce this we require:

```
within a Conception:
  there exists a Concept that is at the top of the conception
```

Even more, that concept is unique:

Corollary 4.1 (Unique concepts).

```
within a Conception:
  there exists exactly one Concept that is at the top of the conception
```

Proof. Left as an exercise to the reader.

Another consequence of the previous rule is that it forces a conception to be connected. In other words, the requirement of a conception being a connected graph, as put forward in the previous Section becomes redundant by the previous rule.

The only concept which is neither a domain element nor an environment element, is the top of the conception:

```
within a Conception:
  each Concept which is not a Domain element or an Environment element
  must be a Concept which is at the top of the environment
```

For domains and environments we also require unique tops to exist:

```
within a Conception:
  there exists a Domain element that is at the top of the domain
```

```
within a Conception:
  there exists an Environment element that is at the top of the environment
```

Corollary 4.1 applies to each of these as well.

For these unique tops we require:

```
within a Conception:
  the Concept [ which is at the top of the environment ]
  is an immediate composite of the Concept which
    is at the top of the conception
```

```
within a Conception:
  the Concept [ which is at the top of the domain ]
  is an immediate composite of the Concept which
    is at the top of the conception
```

```
within a Conception:
  the Concept [ which is at is at the top of the environment ]
  is level linked to the Concept which is at the top of the domain
```

Which implies that the top of a conception is a triangle as depicted in Figure 4.10.

The only links which can neither be domain element nor an environment element are links between the domain and the environment and/or the top of the conception

Corollary 4.2 (Bridges).

```
within a Conception:
  each Link
    which is neither an Environment element nor
    a Domain element nor
    a Link from some Concept which is at the top of the conception
  must be
    a Link from some Environment element and
    a Link to some Domain element
```

Proof. Left as an exercise to the reader.

Connections between decomposed concepts should be exhibited at the composite level as well:

```
within a Conception:
  each Concept [ which is level linked to some Concept
    which is a composite of some higher level Concept ]
  is in the composition group of some Concept
    which is level linked to that higher level Concept
```

The other way around applies as well. If the composite concepts show a linkage, then this is shown at the decomposed level as well:

```
within a Conception:
  each Concept [ which is a composite of a Concept
    which is level linked to some higher level Concept ]
  is level linked to some Concept
    which is in the composition group of that higher level Concept
```

4.1.3 Models

In the context of organisations, we are not interested in all types of conceptions. Our interest is limited to those conceptions, that may be referred to as a *model*:

Model – A purposely abstracted domain (possibly in conjunction with its environment) of some 'part' or 'aspect' of the universe a viewer may have an interest in.
For practical reasons, a model will typically be consistent and unambiguous with regards to some underlying semantic domain, such as logic.

As a model is a conception, it also consists of elements, which can be specialized further into concepts and links:

Model element – An element from a conception which is a model.
Model concept – A concept from a conception which is a model.
Model link – A link from a conception which is a model.

We are now also in a position to define more precisely what we mean by *modeling*:

Modelling – The act of purposely abstracting a model from (what is conceived to be) a part of the universe.

For practical reasons, we will understand the act of *modelling* to also include the activities involved in the *description* of the model by means of some language and medium.

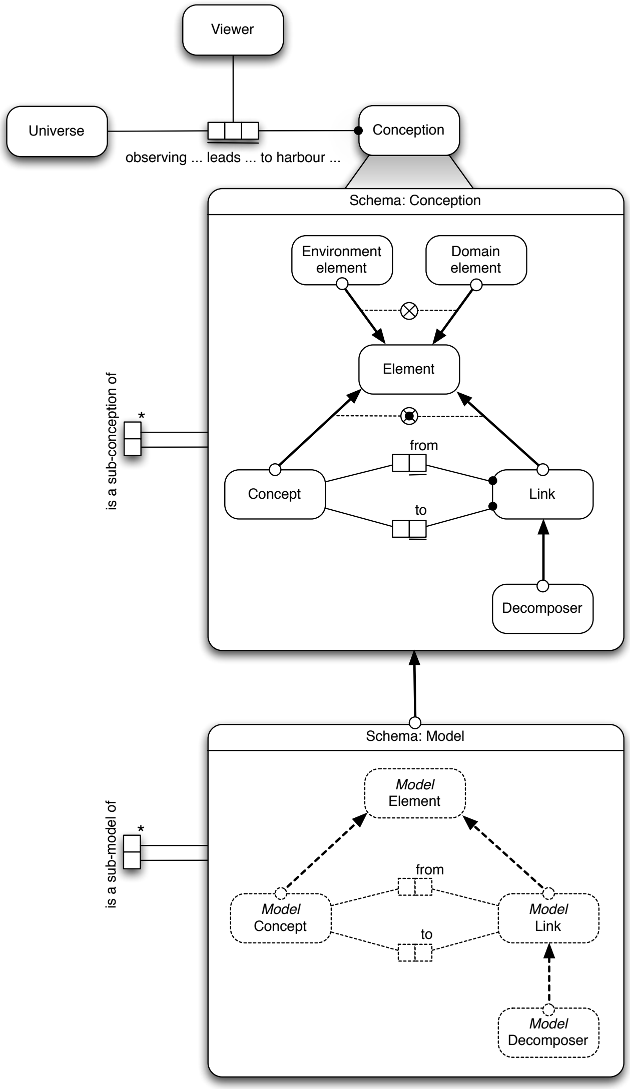


Fig. 4.13 Model as a sub-type of conception

To represent the fact that some viewer produces a model in an environment when they observe some part of the universe, we refine our meta-model to the situation in Figure 4.13. We have represented the class of models by means of a sub-type of conceptions. The fact that we renamed elements, concepts, links and decomposers to *model* elements, *model* concepts, *model* links and *model* decomposers is indicated by means of the dotted schema inside the model schema type.

The meta-model as depicted in Figure 4.13 also includes two new derived fact types:

a Conception is a sub-conception of **some** Conception **iff**
observing **some** Universe leads **some** Viewer to harbour **a** Conception **and**
observing **that** Universe leads **that** Viewer to harbour **another** Conception
where the first Conception **is a subset of the second** Conception

a Model is a sub-model of **some** Model **iff**
the first Model is a sub-conception of **the second** Model

4.1.4 System

Using the above general definitions, we can, in line with [23], more precisely define the way we view systems:

System domain – A domain that is conceived to be a system, by some viewer, by the distinction from its environment, by its coherence, and because of its systemic property.

Systemic property – A meaningful relationship that exists between the domain of elements considered as a whole, the system domain and its environment.

System viewer – A viewer of a system domain.

System – A special model of a system domain, whereby all the things contained in that model are transitively coherent, i.e. all of them are directly or indirectly related to each other and form a coherent whole.
A system is conceived as having assigned to it, as a whole, a specific characterisation (a non-empty set of systemic property) which, in general, cannot be attributed exclusively to any of its components.

The elements, concepts and links concepts can be further specialized to systems:

System element – Any element from a system.

System concept – Any element from a system that is a concept.

System link – Any element from a system that is a link.

As identified in [23], there is a potential objection against our subjectivity-based definition of system. In daily life, it is quite sensible to talk about "designing, constructing and implementing a system" or "to interact with a system". The use of the terms 'system' gives associations to this term as denoting something that can be interacted with in a rather concrete way and not just as a conception. These associations, however, do not lead to any inconsistencies. These example phrases are simply convenient abbreviations for more elaborate expressions. For instance, “to interact with a system” really means:

> *to interact with phenomena in the system domain that is conceived as a system (because of its systemic properties).*

To “design, construct and implement” a system really means:

> *to bring together and structure phenomena in a particular part of the world (which then becomes the system domain) with the purpose of constructing them such that they together have certain systemic properties.*

To gain a better understanding of complex systems it has proven to be useful to identify smaller-scale systems within a larger system, leading to sub-system. A detailed discussion on dealing with complexity by systems in general, and the role played by hierarchical decomposition, may be found in e.g. [90]. In this textbook, for example, information systems will be positioned as sub-systems of organisational systems.

However, when it comes to the point of being less intuitive and more explicit about the concept, there is little consensus about what really characterizes a sub-system – or rather what should characterize it, if the concept is to be a useful one. The influence from the absoluteness of the ‘classical’ system concept together with some apparent preference to associate the understanding of sub-system with the subset concept seem to be the main cause of the confusion.

The ‘old’, simple interpretation of the concept system as being just ‘a set of interrelated parts’, made it rather obvious to think of sub-system as: A subset of the parts together with an appropriate subset of their mutual relationships. However, with the introduction of the notion that in order for something to be a system, it must have at least one systemic property, the matters became more difficult: Should the definition of sub-system then also involve the specification of a subset of the systemic properties? Intuitively this notion could be reasonable, and it may even work in some cases, but the problem is that this is not always so. Consider, for example, a well-functioning mechanical watch. It can be conceived of as having the systemic property that under certain conditions it ‘shows the time’. A possible sub-system of such a watch is the energy supplying device for the clockwork consisting of the spring, the winding knob, the exchange and click mechanism for tightening the spring, and a part of the frame to support these mechanical parts. The only sensible systemic property of such a sub-system is that it serves as a storage of mechanical energy. But then we have a serious problem with the subset notion applied on the systemic property, because being an energy storage is in no way a subset of the systemic property of showing the time.

The problem of defining a sensible sub-system concept by means of subset relationships becomes even more difficult with the notion of a system as a subjective issue. Apart from the systemic properties not being absolute, but rather depending of the viewer, one element in the system domain may now also potentially be viewed as several different in the system. Consider, for example, an organization that is viewed as an and a person from that organization: Here the person may appear as an actor of the type salesman that is the agent of various sales activities. But independent hereof, the same person may also be conceived as having the type employee relevant in connection with calculations of salaries and the planning of sales campaigns. The person may even be regarded as being of type transportable object in the context of an activity transport by car during sales trips. This causes the following question: Should a possible subset relationship applied in attempts to define a subsystem concept then refer to the domain alone, or to the system alone, or to both? It is certainly difficult to find logical or pragmatic arguments that universally justify any of these choices. (For further aspects of the problems encountered when one is aiming at defining sub-system by means of subsets, see the more comprehensive discussion in [23].)

It is necessary to consider the sub-system concept differently – in fact, in a way that very well is in accordance with the way people intuitively apply it in practice. The 'solution' is to realize that when viewing something as a system then only one system should be considered at a time. Applied here, either one must consider that which is regarded as the system or that which is regarded as the sub-system. The advantage of this sub-system interpretation is exactly what appears to be the main positive feature of the intuitively applied concept: Depending on which level of detail as regard potential components you want to consider, you can use the concept to encapsulate unnecessary details on a chosen level of abstraction. Applied to organizations one obvious way to consider the relationship between an organization and a sub-system of it, is to conceive the sub-system equivalent with what an actor in the organization does (or a part of that). Typically a whole department (a possible system candidate in itself) may be considered a single actor in the organization, and (part of) what is done in that department in respect to other departments (i.e. possible systemic properties of the "department system") may be conceived as a single action at the organisational level. A data-processing system may be conceived as a single (artificial) actor carrying out data-processing actions in the organization, even if we know that it, in fact, is composed of a lot of components.

To represent the fact that some viewer 'sees' a system in an environment when they observe some part of the universe, we further refine the meta-model. Since each system is a model, systems are treated as a subtype of models. This leads to Figure 4.14.

In the refined meta-model, the notion of sub-models has been extended to systems as well:

```
a System is a sub-system of **some** System **iff**
  **the first** System is a sub-conception of **the second** System
```

Two common dimensions along which to define sub-systems are component system and aspect system:

Component system – A component-system S' of a system S, is a sub-system, where the set of model concepts in S' is a proper subset of the set of entities in S.

Aspect system – An aspect-system S' of a system S, is a sub-system, where the set of model links in S' is a proper subset of the set of the links in S.

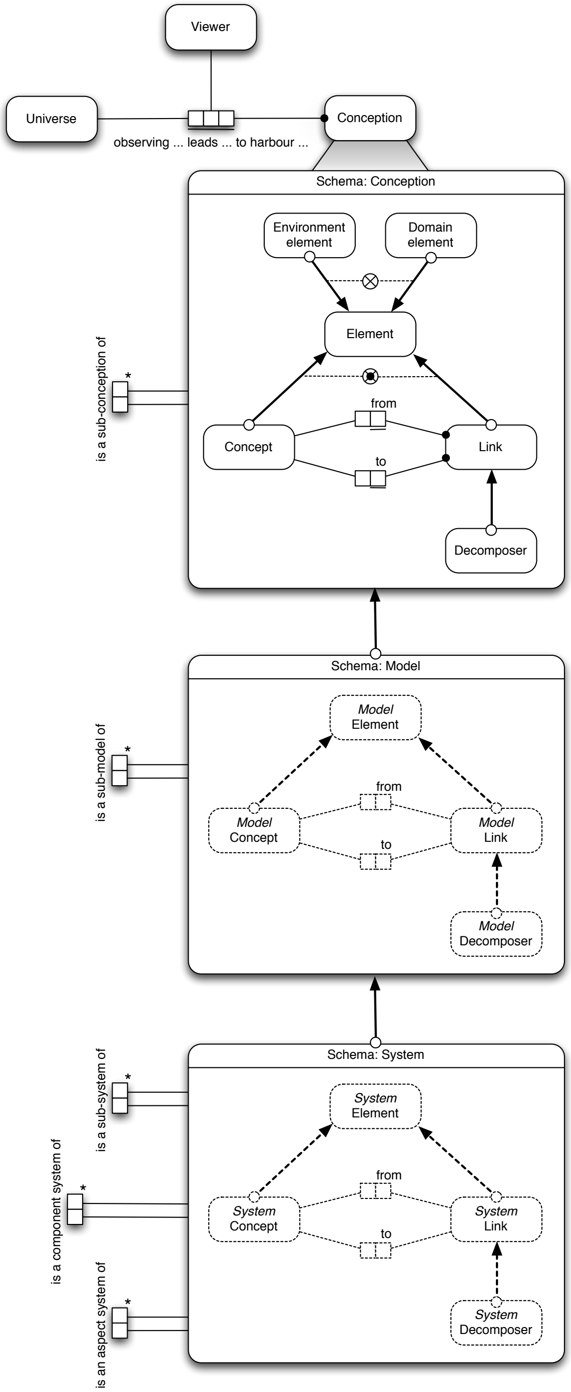


Fig. 4.14 Systems as a sub-type of models

Formally:

a System is a component system of **some** System **iff**
the first System is a sub-system of **the second** System **and**
each Concept [in **the first** System] is a Concept in **the second** System

a System is an aspect system of **some** System **iff**
the first System is a sub-system of **the second** System **and**
each Link [in **the first** System] is a Link in **the second** System

Note that some authors, for example [97, 10], use the term sub-system to refer to the above defined concept of component system. However, we prefer to use the term sub-system as defined above (following the definition in [23]), as it allows us to view it as a generalization of the concepts component system and aspect system.

Different viewers may disagree on the fact whether some sub-system is an aspect system or a component system (or a combination thereof). This can be traced back to the subjectivity involved in distinguishing between links and concepts. Whenever there is a 'clear' analogy to physical structures, it will be easier to identify the difference. Consider a freight-train as an example system. Typical component systems of such a system are: the locomotive, the engine-driver, several types of box-cars, etc. An aspect system of a freight-train would be the hydraulic braking system of the train as a whole.

A sub-system is indeed a system. As such, a sub-system S' of a system S will also have its own systemic properties. However, these properties are most likely no subset of the systemic properties of S. For example, the engine-driver's systemic properties are by no-means a clear subset of the systemic properties of a freight-train.

In our informal exploration of the concept of system, we already discussed that there are three major ways of viewing systems [87]: *structural*, *functional* and *hierarchical* (as a specific class of structural). A major difference between a structural and a functional perspective is the distinction between the white-box and black-box approach when regarding systems. In other words, is one looking *inside* the system (white-box) or is one only looking at the *outside* of the system. This does seem to raise the question whether, when viewing a system as a black-box, one can still argue that the system consists of elements? The answer to this question is a resounding *yes*. When a viewer, for some reason, views a domain as a system, and does so using a black-box approach, that what they conceive of as being a system is still a conception consisting of elements. The difference between a white-box and a black-box approach when viewing a system, however, is in the concepts and links one will see. When taking a black-box approach, one will only see the external behaviour of the system, while when taking the white-box approach one will see the internal structure/behaviour of the system as well.

4.2 Describing & studying systems

In order to really to understand the concept system it is necessary to be aware of a number of important aspects:

- that the system domain always comprises several elements,
- that all elements are related to each other such that it constitutes a transitively coherent whole,
- that the whole is conceived to have at least one systemic property,
- that it is only relevant to incorporate a thing as an element of the particular system domain if in the system view it somehow contributes to the systemic property,
- that when viewing a thing as an element of a system domain then only those aspects of the thing that directly or indirectly contributes to the systemic properties are relevant for the system view.

When a system developer in a system viewing or modelling process gradually realizes what (currently) 'is the system', i.e. becomes conscious of all relevant aspects of the involved elements and of each of the systemic property, it is very useful to be aware of the type of system in question and to produce a system exposition in accordance with the system type.

System type – A type that determines the potential kinds of systemic property, elements of the system domain and roles of the elements in achieving the systemic properties.

System exposition – A description of all the elements of the system domain where each element is specified by all its relevant aspects and all the roles it plays, being of importance for the interestinterest of the viewer. (The system viewer may conceive one and the same thing in the system domain to play more than one role in the system.)

A system type can be regarded as a viewing template to be used by a system developer, analyst or modeller in order to decide which kinds of things (and thereby which aspects of the things) to consider relevant in realizing what actually 'is the system'. A system type comprises:

- Properties determining 'the nature' of the systemic properties, for example for open active systems that the system is seen as something that changes things in the domain of the environment and that the environment is seen as changing things in the system domain. This set of properties may be called the *system characteristic*.
- Properties determining the kinds of things which are relevant to incorporate in the exposition of the system domain, and for each kind the kinds of roles they may play in respect to the potential kinds of systemic properties. Examples of such kinds of things are for dynamic systems: states, transitions and transition occurrences, and for open active systems (among other things): actions, subjects, agents, transitions in the domain of the environment caused by actions in the system, etc. This set of properties may be called the *exposition characteristic*.

A more detailed elaboration of concepts related with the system viewing process can be found in [23]. A semi-formal description of it based on an example is presented in [61].

In conceiving a domain as an organization, several classes of elements may be relevant to include in a system exposition of that domain. As part of the domain it may also be relevant to incorporate a number of concepts generally relevant in an organizational context, for example public services, laws or other kinds of constraints imposed by society, or aspects of the particular professional field of the organization. However, for an organization it is generally relevant to consider the following kinds of things as candidates to (at least) be included in a system exposition:

Actors – human actors as well as artificial actors and all kinds of symbiotic compositions of these two kinds.

Actions – (together with the associated goals) such that a (not exclusive) distinction is made between those influenced by impressions from the environment and those either directly constituting expressions of the system or only contributing to (or in some cases even explicitly counteracting) the expressions. Actions that are irrelevant for the expression of the system should be ignored in the exposition.

Co-actions – i.e. co-ordinated actions performed by several actors together.

Knowledge – that is necessary for the actors to know the relevant pre-states of their actions and the respective goals. A goal may be situation dependent.

Triggers – involving internal and/or external dynamic criteria for the initiation of actions (temporal, impressive and actor- or action-caused transitions).

Communication – between actors to ensure that they have the information necessary to perform their actions.

Representation – of the information/knowledge relevant to the organization's activities, in order to enable the preservation or communication of it. That includes all relevant aspects of the use of data technology and/or data-technical sub-systems to accomplish the preservation or communication.

In practice, aspects of organizational culture, social norms, empation (i.e. knowledge that cannot be properly represented), resources in general (energy, skills, intellect, etc.), ecology, economy, etc., may be added to this list.

A work system, a organization, as well as an information systems belong to a system type that primarily is characterized as being open and active (where the latter implies also that it is dynamic). We can define these specific types of systems as:

Active system – A special kind of system that is conceived of as begin able to change parts of the universe.

Dynamic system – A special kind of system that is conceived of as undergoing change in the cause of time.

Open system – A special kind of dynamic system that is conceived as reacting to external triggers, i.e. there may be changes inside the system due to external causes originating from the system's environment.

Open active system – A system that is an open system as well as an active system.

Note that a system may be active and yet be non-dynamic. For example, the mere presence of a dummy speeding camera, i.e. one that is not able to capture speeding vehicles on film, may lead drivers to drive more slowly. The dummy speeding camera may thus be seen as an active, yet non-dynamic, system.

Note that the sub-system of an open active system does not have to be an open active system. In other words, even though our main interest lies with open active system, we may quite well need to consider non-open or non-active sub-systems of these systems.

For open active systems – therefore for organizations too – it is relevant to consider the following. The behaviour of an open active system is generally reflected as:

Internal function – Conceptions of changes in the system domain caused by processes in the domain itself.

External function – Here the following two kinds are distinguished:

- **Impression** – Conceptions of changes in the system as caused by the environment.
- **Expression** – Conceptions of changes in the environment as caused by the system.

The very fact that something is regarded as a system often serve the purpose of hiding the internal function and focus on the external function. (Like the phrase “a black-box system”). The internal function of an open active system is referred to as “the function *in* the system”, while the external function is “the function *of* the system”. The latter is equivalent with the systemic property of an open active system.

One can classify open active systems in several ways according to their behaviour (for details see [4]). Here we shall only distinguish between three kinds of open active system based on the following distinctions. A reaction of an open active system is an expression that is seen as unconditionally caused by an impression. An action of an open active system is an expression that is seen as being completely independent on any kind of impression. Thereby we can define the three additional types of open active systems:

Reactive system – An open active system where each expression of the system is a reaction, and where each impression immediately causes a reaction.

Responsive system – An open active system (possibly also a reactive system) where it holds for at least one expression that a certain impression or a temporal pattern of impressions is a necessary, but not a sufficient dynamic condition for its occurrence. The receipt of an order is a necessary impression to a “sales system”, for the expression “delivery of the ordered goods”, but it is not a sufficient condition.

Autonomous system – An open active system (possibly also a responsive system, but not a reactive system) where at least one expression is an action. A human being and most (if not all) organisation can be regarded as autonomous system.

As mentioned before, in [5, 6] Alter defines a work system as:

> *A work system is a system in which human participants and/or machines perform business processes using information, technologies, and other resources to produce products and/or services for internal or external customers.*

where *information systems* are to be regarded as special classes of work systems. We will therefore operate under the assumption that we have the following hierarchy of systems:

- Systems in general.
- Open active systems: Subclass of systems.
- Work systems: Subclass of open active systems.
- Organisational systems: Subclass of work systems.
- Information systems: Subclass of work systems and a sub-system of organisational systems.
- Computerised information systems: Subclass of work systems and a sub-system of information systems.

Based on [23] and [5], we can provide the following stacked set of definitions:

Work system – An open active system in which actor perform processes using information, technologies, and other resources to produce products and/or services for internal or external actors.

Enterprise system – An enterprise, being a work system, and/or one of its subsystem.

Organisational system – A special kind of system, being normally active and open, and comprising the conception of how an organisation is composed and how it operates (i.e. performing specific actions in pursuit of organizational goals, guided by organizational rules and informed by internal and external communication), where its systemic property are that it responds to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment.

Information system – A sub-system of an organisational system, comprising the conception of how the communication and information-oriented aspects of an organisation are composed and how these operate, thus leading to a description of the (explicit and/or implicit) communication-oriented and information-providing actions and arrangements existing within the organisational system.

Computerised information system – A sub-system of an information system, in which all activities are performed by one or several computer(s).

{*The definition of organisational system should really build on the definition of enterprise system. This is currently not the case.*}

4.3 Evolution of conceptions

Before concluding this Chapter, there is one final issue to deal with. An enterprise system is an open active system. This specifically means that it is a system which changes over time. Thus far we have taken the assumption that the conception of a viewer is a static notion. If we say “observing Universe *U* leads Viewer *V* to harbour Conception *C*”, then this really means that viewer *V* has *at some point in time* the conception *C* when observing (a part of) universe *U*. However, in the course of time this conception will evolve, which raises the question: *How to deal with evolution of conceptions?*.

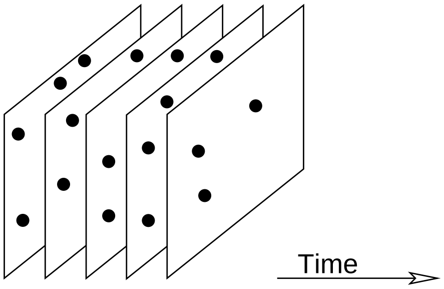


Fig. 4.15 Modelling evolution by snapshots

Several strategies exist to deal with evolution [77]. One strategy to deal with this evolution is to take snapshots, like photographs, of a viewer’s conceptions. This leads to the situation depicted in Figure 4.15. This approach, however, does have as drawback that one cannot ‘trace’ the evolution of a specific element in a viewer’s conception. The approach we take, therefore, is illustrated in Figure 4.16.

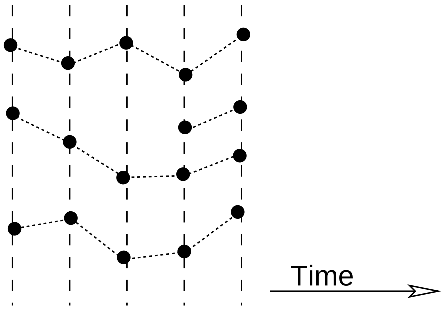


Fig. 4.16 Modelling evolution by functions in time

Based on the approach taken in [77], the evolution of the elements in a viewer’s conception is treated as a set of (partial) functions over time. At each point in time, a specific element (a version) may be associated to such a function. This means (as also illustrated in Figure 4.17, the situation depicted in Figure 4.15 can still be derived. When we know the entire evolution of a nation, we can also provide a detailed descriptions of the state-of-affairs as it holds at any arbitrary point in time.

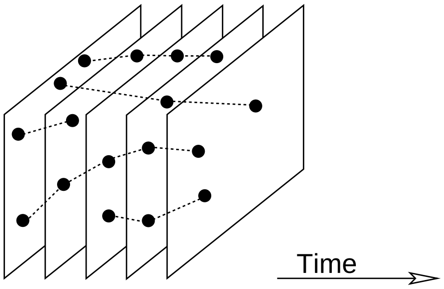


Fig. 4.17 Deriving snapshots

To extend our meta-model, we actually need to refine Figure 4.2, leading to the situation depicted in Figure 4.18.

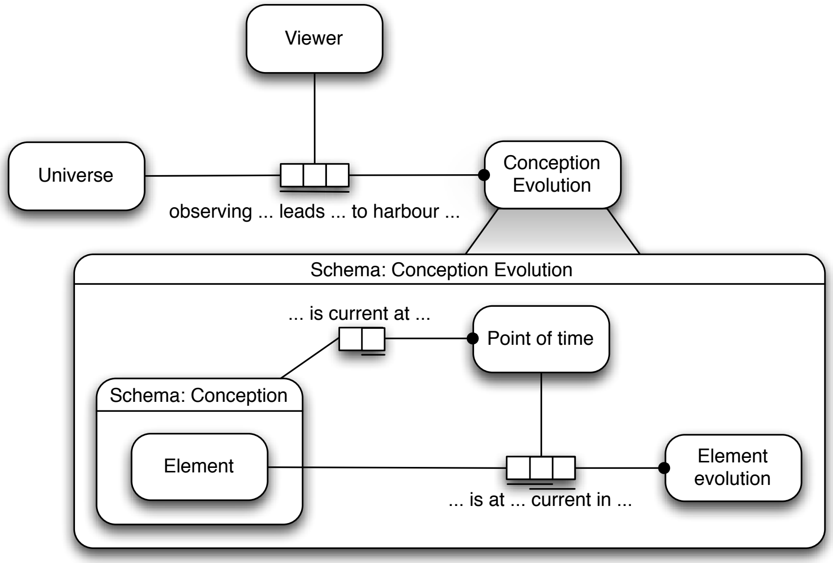


Fig. 4.18 Meta-model extended with evolution

To enforce the correspondence between conception (snapshots) and versions of element evolutions, we require:

within a Conception evolution:
  an Element is at **some** Point of time current in **some** Element evolution **iff**
  that Element is in a Conception **which** is current at **that** Point of time

Element evolutions are *partial* functions, which means that they are not required to be defined for all points in time. In other words, at some point in time an element evolution may not have an element version associated, which really means that the element evolution does not exist yet at that point in time (it has not been born yet), or that it has ceased to exist (it died). In other words, element evolutions are allowed to be re-born.

4.4 Conclusion

In this Chapter we have taken a highly fundamental look on enterprise systems, and the way they are modelled, including their decompositions and evolutions. The resulting meta-model (without the derived fact types) is depicted in Figure 4.19.

4.5 Questions

1. How are the terms ‘enterprise’, ‘domain’ and ‘universe’ be related to each other, given the definitions provided in this textbook?
   - Describe this relation in natural language.
   - Describe this relation in terms of an ORM diagram.
2. Not all conceptions of a domain produce models. Why not?
3. Beschouw een Autoproducent, zoals bijvoorbeeld Seat, BMW en Toyota.
   - Wat zijn de belangrijkste systemische eigenschappen?
   - Beschrijf het primaire gedrag van deze organisatie in termen van interne en externe functies.
4. Give an example of a reactive system, of a responsive system and of an autonomous system (other examples than the ones already given, of course).
5. From a modelling point of view, enterprises can be considered as systems containing a.o. concepts and links.
   - Why is it important to be aware of the aspect of subjectivity when creating models?
   - What view does an information system developer have when modelling organizations?
   - Why would an information system developer want to start by creating a model of an organization, instead of directly focusing on modelling an information system?

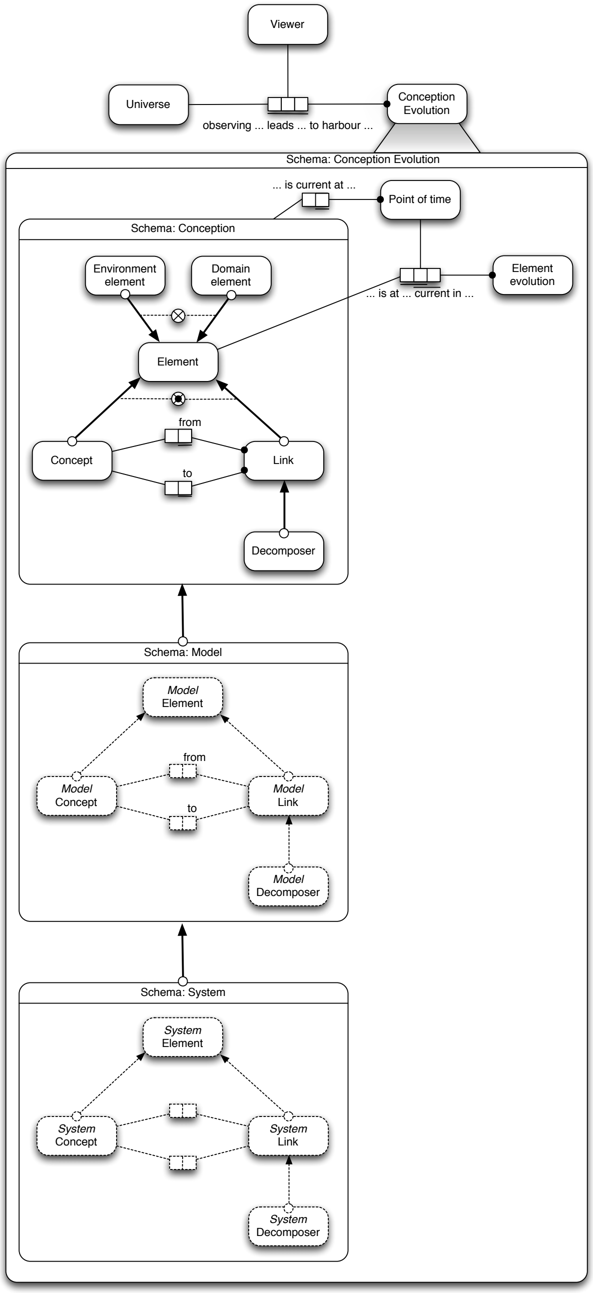


Fig. 4.19 Integrated meta-model of conceptions

6. Waarom zullen verschillende mensen wanneer ze verschillende domeinen modelleren toch verschillende modellen opleveren? Hoe kun je deze situatie verbeteren? Waarom zou je dit willen verbeteren?
7. Suppose you are requested by a large enterprise (a holding company holding some daughter companies) to create more insight into their own activities by creating some models of their organization. The focus of this models must, according to the board of directors, be on their internal information flows, since the organization has the impression that a lot of business efficiency is lost due to an incompetent set of information systems. Keeping in mind what is explained in the two previous chapters, give an impression of:

   - Where would you start modelling?
   - What would you model?
   - Why model that?

8. Consider a home cinema set.

   - Describe the systems elements.
   - Distinguish proper sub-systems.
   - Can you derive typical aspect systems and component systems?

   Explain your answers.
9. Consider a travel agency.

   - Describe the most important system characteristics and exposition characteristics.
   - Describe its behaviour in terms of internal and external functions.

10. Give some examples of:

   - Work systems that are not enterprise systems.
   - Enterprise systems.
   - Information systems.

Chapter 5
Object-Role Modelling

The previous Chapters we did (see Figure 3.5) refer to the fact that viewers are able to provide a description of the conception. However, we did not really follow up on this. This Chapter, however, will indeed take these descriptions as a starting point. In this Chapter, we will essentially provide a brief summary of the modelling approach from Domain Modelling. In Chapter ??, we will enrich this modelling approach with constructs that allow us to model work systems.

5.1 Natural language grounding of modelling

It is not an uncommon approach to base modelling on natural language analysis:

ORM [32], NIAM [103, 65], UML use cases [13], DEMO [86], KISS [55] and OOSA [21].

When people work together, they are bound to use some language. The language skills of the human race evolved hand-in-hand with the levels of organization of our activities. From organization of hunting parties by our pre-historic ancestors, to the organization of factories and businesses in the present. Without the use of language, it would not have worked. As a result, most (if not all) enterprises we see around us are social constructs that are the result of communication between actors, mostly human actors. This makes it all the more natural to base our modelling endeavours on the language we use most to talk about enterprises i.e. natural language.

5.2 The logbook heuristic

Natural language based modelling approaches such as ORM employ different variations of the so-called telephone heuristic. This heuristic presumes some viewer to observe a domain (including its *evolution*), and use a 'telephone' to convey their observations to some other person (or computer). This is depicted in Figure 5.1.

The left hand viewer tells the right hand viewer ‘what they see’ in terms of their observations.

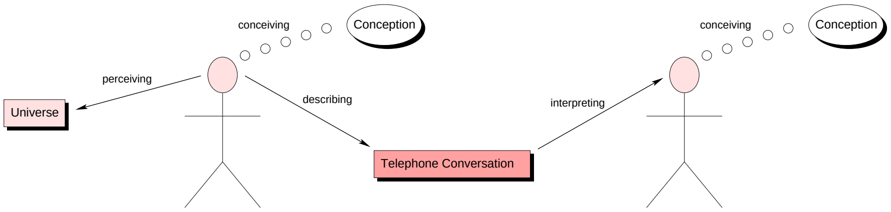


Fig. 5.1 The telephone heuristic

In this Chapter we are interested in having a “transcript” of the observations made during the telephone conversation from Figure 5.1. More specifically, we want to maintain a *logbook* of this telephone conversation, leading to the situation as depicted in Figure 5.2.

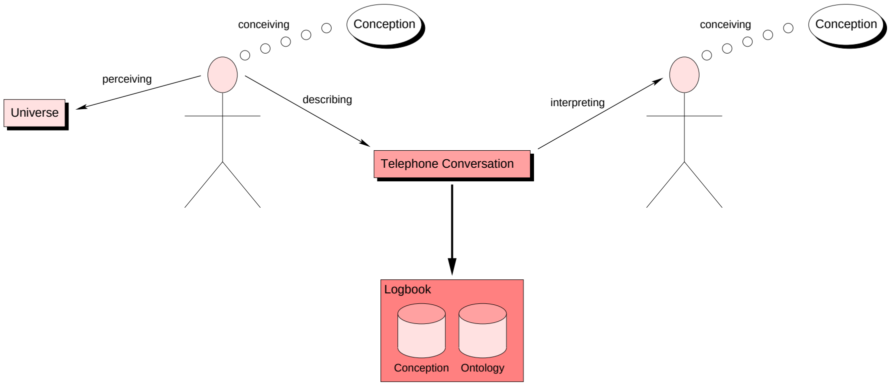


Fig. 5.2 Logging the telephone conversation

Even more, we could actually replace the second person from Figure 5.2, leaving only the original viewer and the logbook to maintain the transcript. This leads to the *logbook heuristic* as depicted in Figure 5.3.

Note that the logbook is regarded as having a *conception* based on an *ontology* as well. As a starting point, we will presume this ontology to consist at least of the situation as depicted in Figure 4.19. In the remainder of this text book, we will actually refine this ontology even further.

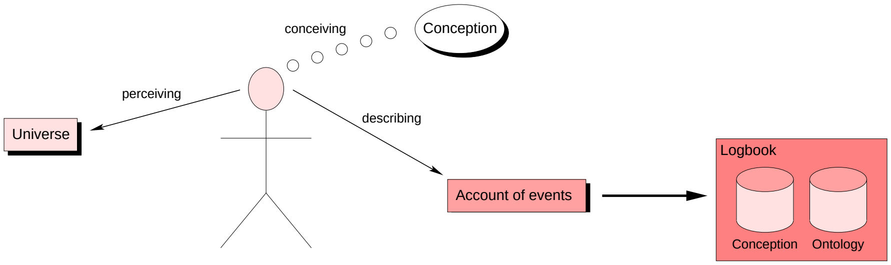


Fig. 5.3 The logbook heuristic

5.3 Verbalising conceptions

We presume the transcriptions that are entered into the logbook to refer to events ‘in the life’ of specific element evolutions of a viewer’s conception. These events refer to changes in the state of the elements, and are presumed to be reported in terms of facts about the elements. An example would be:

Person #001 was born at 22-05-1967
Person #001 received name Erik Proper at 23-05-1967
Person #001 lives at address: Koperwiekstraat 6, Rheden, The Netherlands, EU at 23-05-1967
Person #001 lives at address: 3/26 Rylatt Street, Brisbane, Australia at 28-06-1994
Person #002 lives at address: Koperwiekstraat 6, Rheden, The Netherlands, EU at 29-06-1994
Person #001 works for employer: University of Queensland at 28-06-1994
Person #003 works for employer: University of Queensland at 22-04-1995

When considering this transcript, it is easy to spot that it really deals with more than one element evolution. The following element evolutions *might* be discerned:

persons: #001; #002; #003
name: Erik Proper
addresses:
  Koperwiekstraat 6, Rheden, The Netherlands, EU;
  3/26 Rylatt Street, Brisbane, Australia
employer: University of Queensland
ownership of the name Erik Proper by person #001
living of person #001 at some address
living of person #002 at some address
habitation of address Koperwiekstraat 6, Rheden, The Netherlands by some person
habitation of address 3/26 Rylatt Street, Brisbane, Australia by some person
coworkership of person #001 for some employer
coworkership of person #002 for some employer
employment offered by University of Queensland to a group of people

In the above example, an important trade-off already comes to the surface. What should be selected as element evolutions:

coworkership of person #001 for some employer

and/or

employment offered by University of Queensland to a group of people

What is it that evolves? Either? Both? Ultimately, this is a subjective matter. To be able to better understand the underlying trade-off, we will now first focus on the transcription of a specific snapshot of a conception.

5.4 Elementary facts

Similarly to the *Domain Modeling* course, we require the facts in the transcripts to be elementary, in other words, no logical connectors such as *and* and *or*, and most likely no *not*s either.

Consider, the following domain:

> A person with name Erik is writing a letter to his loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.

We can rephrase this as the set of elementary facts:

A person is writing a letter
This person has the name Erik
This letter has a romantic nature
This letter has intended recipient Erik's loved one
The writing of this letter by Erik, occurs on a mid-summer's day
The writing of this letter by Erik, is done using a pencil
The writing of this letter by Erik, is done while the cat is watching
The writing of this letter by Erik, is taking place at a desk
This desk is located in a room
This room is romantically lit

Within these elementary facts, several objects can be discerned, such as: person Erik, letter, etc. Such objects can biological, physical, social, fictive, etc., in nature. The objects are regarded as playing a role in the facts. In the above example, we can isolate the objects and facts as follows:

[A person] is writing [a letter]
[This person] has [the name Erik]
[This letter] has a [romantic nature]
[This letter] has intended recipient [Erik's loved one]
[The writing of this letter by Erik] occurs on [mid-summer's day]
[The writing of this letter by Erik] is done using [a pencil]
[The writing of this letter by Erik] is done while [the cat] is watching
[The writing of this letter by Erik] is taking [a desk]
[This room] is lit in [a romantic] way

The roles played by the objects can be made more explicit as follows:

[A person (writer)] is writing [a letter (written)]
[This desk (positioned object)] is located in [a room (location)]

The facts, roles and objects are all considered to be concepts, while the connections between them are links. This leads to the refined meta-model as shown in Figure 5.4.

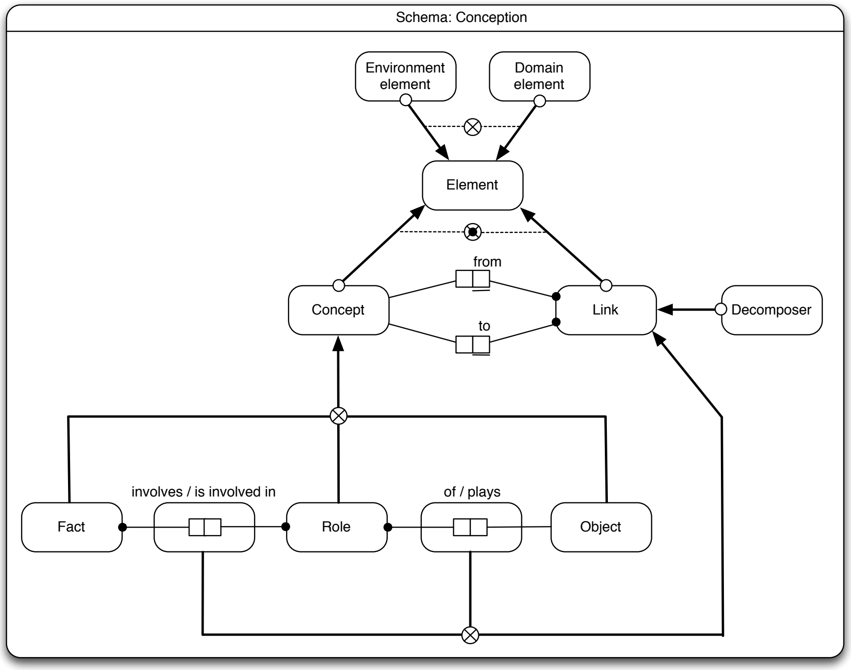


Fig. 5.4 Meta-model catering for facts

within a Conception:
  each Role [**which** is involved in **some** Fact]
    is an immediate composite of **that** Fact

5.5 From instances to types

Consider the elementary sentences:

Person “Erik” is examined by Doctor “Jones”
Person “Wil” is examined by Doctor “Smith”
Person “Marc” is examined by Doctor “Jones”

As we have learned in *Domain Modelling*, we can generalize these sentences to the “type” level:

A Person is examined by a Doctor

with sample population:

Person	Doctor
Erik	Jones
Wil	Smith
Marc	Jones

Formally, we introduce typing as a special kind of decomposition. This leads to the meta-model depicted in Figure 5.5.

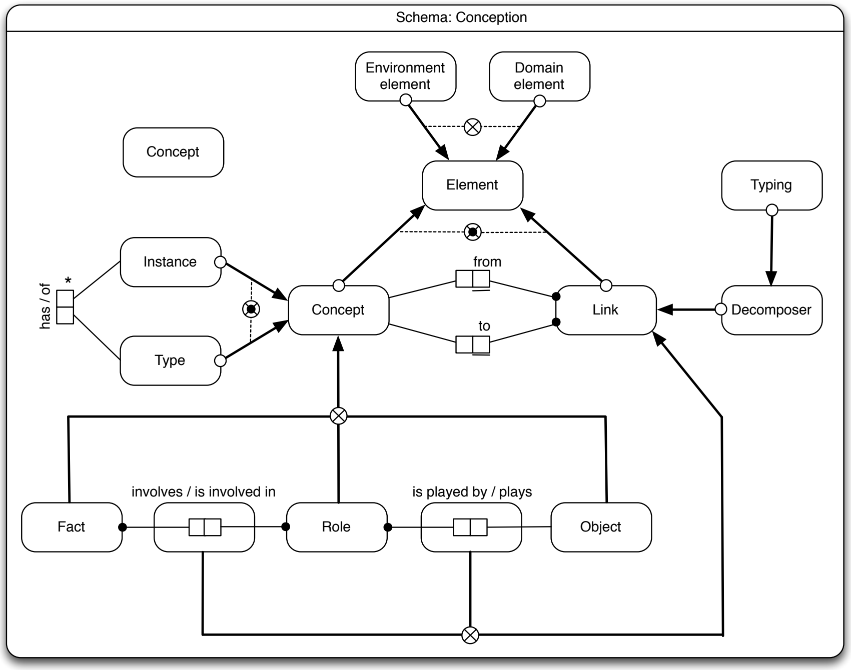


Fig. 5.5 Adding typing

With this extension, we are now able to use ORM's way of describing [32] to represent models. An example of types and instances is shown in Figure 5.6.

The derived fact type between types and instances is defined as:

an Instance has **some** Type **iff**
  some Typing **exists** [from **that** Instance] to **that** Type

With this definition we immediately have:

within a Conception:
  each Instance [**which** has **some** Type] is an immediate composite of **that** Type

Each typing must indeed be from an instance to a type:

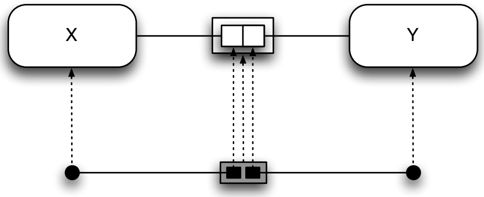


Fig. 5.6 Types and instances

```
within a Conception:
  if some Typing exists [from an instance Concept] to a type Concept then
    the instance Concept is an Instance and
    the type Concept is a Type
```

Note that we do not require each type to have an instance. In other words, we allow conceptions to identify types without having concrete instances. We also do not require all instances to be typed. If some object is regarded as being so unique that it does not have associated objects of the same type, then it does not have a type.

Typing should adhere to the classification of concepts into facts, roles and objects:

```
within a Conception:
  if an Instance has some Type then
    the Instance is a Fact iff the Type is a Fact and
    the Instance is a Role iff the Type is a Role and
    the Instance is an Object iff the Type is an Object
```

The links between facts and roles should not cross the typing boundary. In other words, only fact *types* can involve role types:

```
within a Conception:
  each Fact [which involves a Role Type] is a Type
```

Furthermore, fact types can only involve role *types*:

```
within a Conception:
  each Role [ which is involved in a Fact Type ] is a Type
```

Object types can only be involved in role *types*:

```
within a Conception:
  each Role [ which is played by an Object Type ] is a Type
```

We do not require objects involved in a role type to be a type. In other words, we do *not* demand:

```
within a Conception:
  each Object [ which plays a Role Type ] is a Type
```

In traditional ORM [32] this is, however, required. As we will see in the next Chapters, however, our general approach to enterprise modelling does require the ability to explicitly include instances in models (descriptions). An abstract example is shown in Figure 5.7.

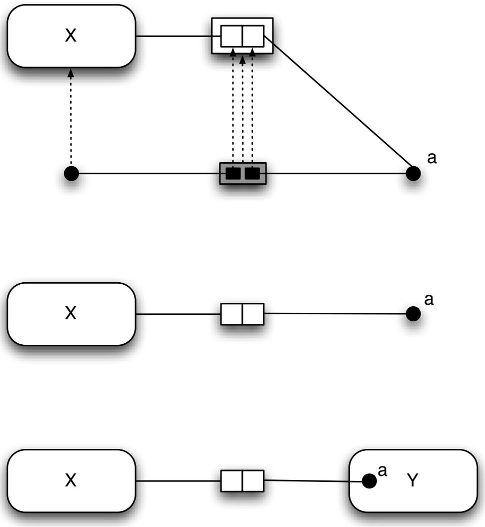


Fig. 5.7 Instances mixed with types

The first situation provides both types and instances. The second situation provides only the type level one would normally like to include in an ORM model. The third situation actually shows a situation in which one would like to signify that instance a is of type Y, but that we are only interested in instance a.

As a more concrete example, consider the following verbalizations regarding the reporting of accidents with insurance companies:

Accident #20 is reported to DigiSurance by client ‘John’
Accident #30 is reported to DigiSurance by client ‘James’

If these verbalisations are in the context of a single insurance company, as suggested by the above two, the fact type would be:

[Accident] is reported to DigiSurance by [Client]

or even:

[Accident] is reported by [Client]

to make DigiSurance even more implicit. If the domain under consideration deals with multiple insurance companies, the fact type would be:

[Accident] is reported to [Insurance Company] by [Client]

However, when we only deal with one insurance company, there may be several reasons why one may want to explicitly include “DigiSurance” in an ORM model. Possibly acknowledging it as an insurance company, but without explicitly introducing an object type Insurance Company. For example, when producing a domain model for an DigiSurance, the stakeholders may appreciate seeing explicitly which fact types explicitly refer to roles played by *their* company. The resulting ORM models are shown in Figure 5.8. In the top diagram we have declared DigiSurance to be an insurance company, while at the bottom one this has been left implicit.

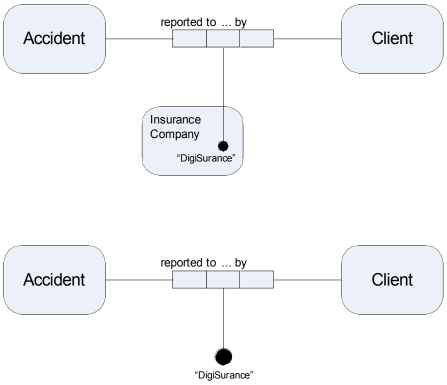


Fig. 5.8 Adding specific instances to an ORM model

Typing is treated as a form of decomposition. All other forms of decomposition should not cross the type-instance boundaries:

```
within a Conception:
  if a Concept is an immediate composite of some Concept and
    the former Concept is not an Instance of the latter
  then the former is a Type iff the latter is a Type
```

Players involved in role instances should behave as stipulated at the type level:

```
within a Conception:
  each Object [ which plays a Role Instance of some Role Type ]
  is an Instance of some Object Type which plays that Role Type
```

The same applies to facts:

```
within a Conception:
  each Fact [ which involves a Role Instance of some Role Type ]
  is an Instance of some Fact Type which involves that Role Type
```

Even more, as all role types of a fact type should be populated, the reverse should hold as well:

```
within a Conception:
  each Fact Instance [ of some Fact Type which involves some Role Type]
    involves a Role Instance of that Role Type
```

As an immediate result we have:

```
within a Conception:
  a Fact Type has some Fact Instance iff
    that Fact Type involves some Role Type
      which has a Role Instance which is played by that Fact Instance
```

We can express this in terms of role types:

```
(Total population of fact roles)
  within a Conception:
    a Role Type has a Role Instance which is involved in some Fact Instance iff
      that Role Type is involved in some Fact Type
        which has as Instance that Fact Instance
```

The last rule does not have a pendent for objects. In other words, we do not generally have:

```
within a Conception:
  each Object Instance [ of some Object Type which plays some Role Type ]
    plays a Role Instance of that Role Type
```

as this would require all instances of a player type to be involved in *all* roles in which the type is involved. However, we do have a weaker version as we will see in the next Section.

Facts should behave as a function from role types to instances:

```
within a Conception:
  if a Fact Instance [ which involves a Role Instance of some Role Type ]
    is the same as a Fact Instance
      which involves some Role Instance of that Role Type
  then the first Role Instance is the same as the second Role Instance
```

5.6 Subtyping

Sub-typing is an important feature of object-role modelling, which essentially boils down to a set of rules governing the typing of instances. In principle, one would like typing to be exclusive:

```
within a Conception:
  each Instance has at most one Type
```

We will indeed require this for instances other than objects:

```
within a Conception:
  each Instance [ which is not an Object ] has at most one Type
```

However, in the case of objects this is more subtle since we need the ability of introducing subtypes. Figure 5.9 shows an example of subtyping in terms of a specialization hierarchy. In general, sub-typing involves the identification of a subset of the population of some super-type. For example, in the situation depicted in Figure 5.9 *flesh eater* is a specific sub-set of *animals*. In different versions of ORM, different rules apply to sub-typing [37, 34, 32]. In this textbook we present a rather generic interpretation.

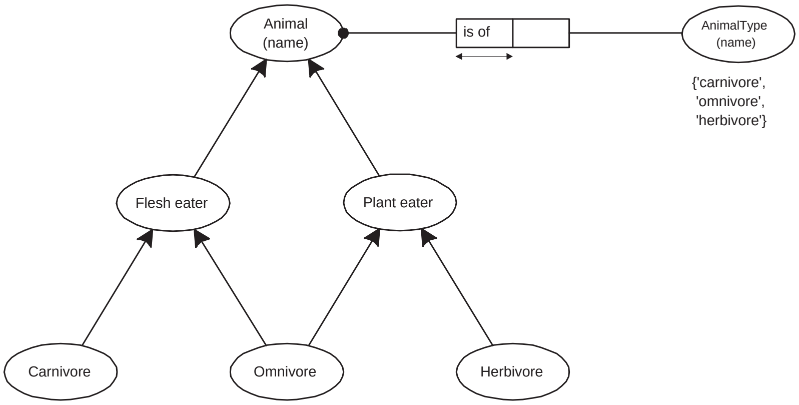


Fig. 5.9 Example of a subtyping hierarchy

Formally, we treat sub-typing as a set of regulations governing sub-typing. This leads to the refined meta-model as depicted in Figure 5.10.

The new derived fact type is defined as:

```
an Object Type is family of some Object Type iff
  the former is equal to the latter or
  the former is a subtype of the latter
  the latter is a subtype of the former
  some Object Type (is a subtype of the former AND is a subtype of the latter)
```

Sub-typing is a transitive and acyclic relationship. In other words:

```
within a Conception:
  each Object Type
    [ which is a subtype of some Object Type
      which is a subtype of some Object Type ]
    is a subtype of the last Object Type

within a Conception:
  if an Object Type is a subtype of some Object Type
    then the former is not equal to the latter
```

The semantics of sub-typing in terms of populations is that the population of a sub-type should be a subset of the population of the super-type:

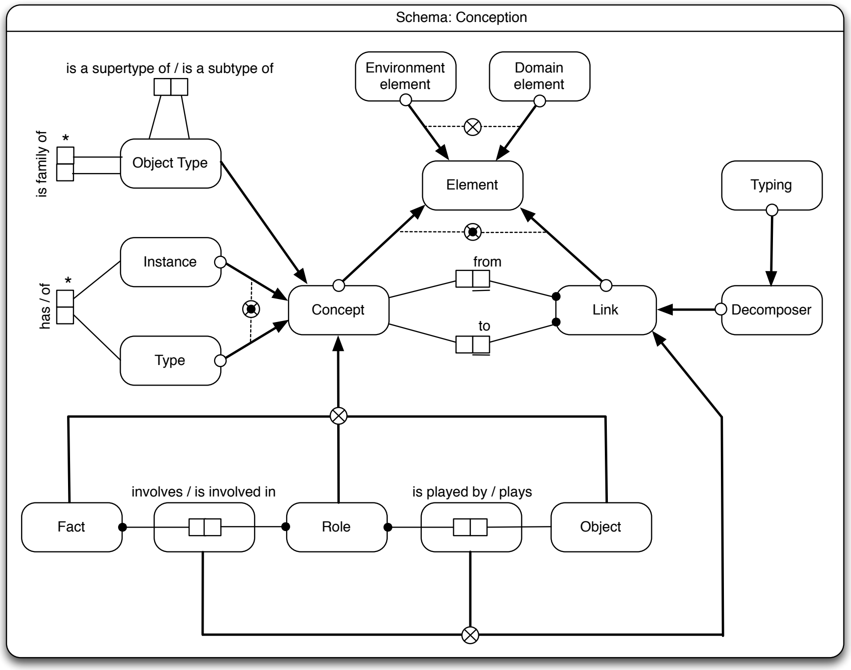


Fig. 5.10 Adding subtyping to the meta-model

```
within a Conception:
  each Object
    [ which is an Instance of some Object Type
      which is a subtype of some Object Type ]
  is an Instance of the latter Object Type
```

As promised, we would introduce a weaker pendant of the *Total population of fact roles* rule for objects. Each instance of an object type must play one of the roles of the object type or one of its super-types:

```
within a Conception:
  each Object
    [ which is an Instance of some Object Type ]
    plays some Role Instance of a Role Type which is played by
      an other Object Type
        which ( is a supertype of or equal to) the first Object Type
```

In other words, instances of a player type *must* be active in one of the associated roles. With this rule we can actually prove:

```
within a Conception:
  an Object is an Instance of some Object Type
  iff that Object Instance plays some Role Instance of a Role Type which is played
```

by **an other** Object Type
  which (is a supertype of **or equal to**) **the first** Object Type

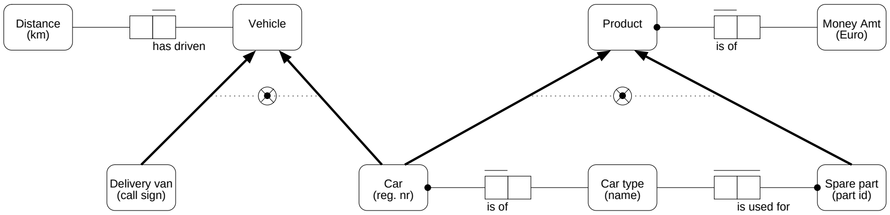


Fig. 5.11 Example of a multi-rooted subtyping hierarchy

When considering the ORM schema as depicted in Figure 5.11, one would expect the populations of Vehicle and Distance to be disjoint, while the populations of Vehicle and Product are expected to overlap. Thus far, we have not introduced any formal mechanism to enforce this type of behaviour other that the inclusion of populations for sub-types. Using the is family of derived fact type we can formalise this:

within a Conception:
 each Object Type **which** has **an** Object Instance
  which is an Instance of **another** Object Type
   is family of **the first** Object Type

5.7 Constraining subtypes

The population of a subtype can be restrained further. In ORM, this is commonly done using a so-called sub-type defining rule. Figure 5.9 showed an example of subtyping in terms of a specialization hierarchy.

The population of a subtype can be restrained further. Rules can be specified that specify the 'maximum' and 'minimum' population of a sub-type. Normally, subtyping only requires:

within a Conception:
 each Object
  [**which is an** Instance of **some** Object Type]
  plays **some** Role Instance of **a** Role Type
   which (is a supertype of **or equal to**) **that** Object Type

Graphically, this leads to the situation *i)* as depicted in Figure 5.12. The population of a subtyp can be restricted further by requiring it to be a sub/superset of some set of instances specified by a (subtype constraining) rule. Situations *ii)*, *iii)* and *iv)* of Figure 5.12 depict this graphically. In situation *ii)*, the population of *X* should at least consist of those match rule *R*. In general, this can be defined as follows:

within a Conception:
  each Object [**which** $\ll R \gg$] **is an** Instance of Object Type *X*

where "$\ll R \gg$" refers to an ORC expression over the role/object/fact types involved in the model.

While the situation from *ii)* provides a minimum population for *X*, the situation depicted in *iii)* does the reverse by demanding a maximum population. The population of *X* is limited to those instances of *X*'s *super-types* that match rule *S*:

within a Conception:
  each Object [**which is an** Instance of Object Type *X*] $\ll S \gg$

The situation provided in *iv)* combines the maximum and minimum population. Situation *v)* represents a very special case. In this case, the rule *R* actually fully determines the population of the subtype, since: in other words:

within a Conception:
  an Object **is** an Instance of Object Type *X* iff $\ll R \gg$

As a graphical abbreviation we will use the notation provided in situation *vi)*, which corresponds to the traditional notion of *specialization* from ORM [32].

A further interesting case of specialization was depicted in Figure 5.11. It illustrates how specialization hierarchies can have multiple roots. Based on [37] a graphical abbreviation can be used for total sub-typings without subtype defining rules, which generally corresponds to *generalisation*. This is depicted in Figure 5.13. The right-hand side provides an abbreviation for the situation depicted in the left-hand side.

In terms of our meta-model (see Figure 5.10), rules restricting subtyping, as well as other constraints that may be applied to the object/role/fact types in a model/conception are treated as properties added to types in the model. We will not model this explicitly in the meta-model as this essentially would require the extension of the meta-model with the syntax of a rule language such as ORC.

5.8 Objectification revisited

As discussed before, based on [33], we will treat ORM's objectification as an abbreviation. In other words, we will use the abbreviation as suggested in Figure 2.1 (page 14). Since object type *F* is allowed to have subtypes, we can actually have the situations as depicted in Figure 5.14 *i)* and *ii)*.

5.9 Abstraction

To introduce abstraction (*schema decomposition*), we start with an example domain taken from [16].

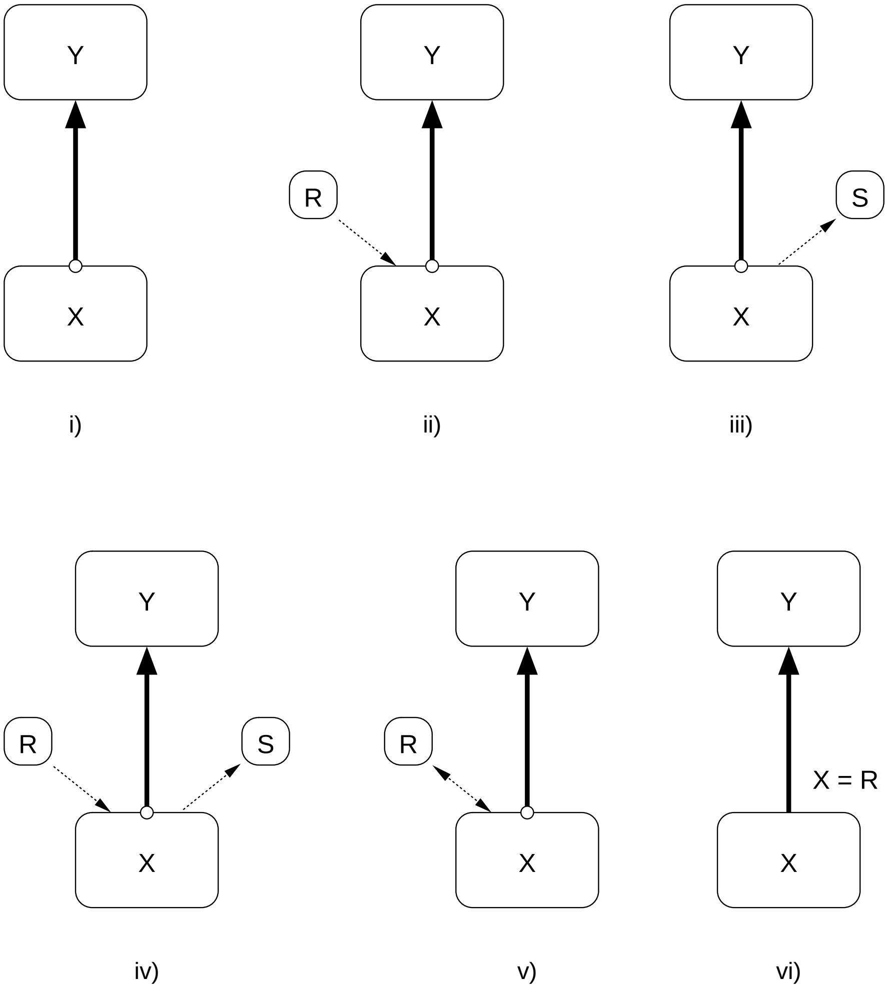


Fig. 5.12 Restricting subtyping

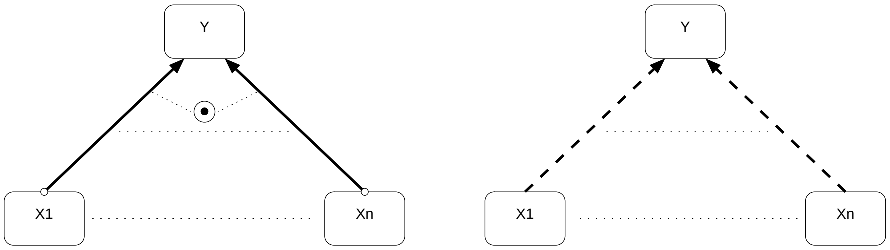
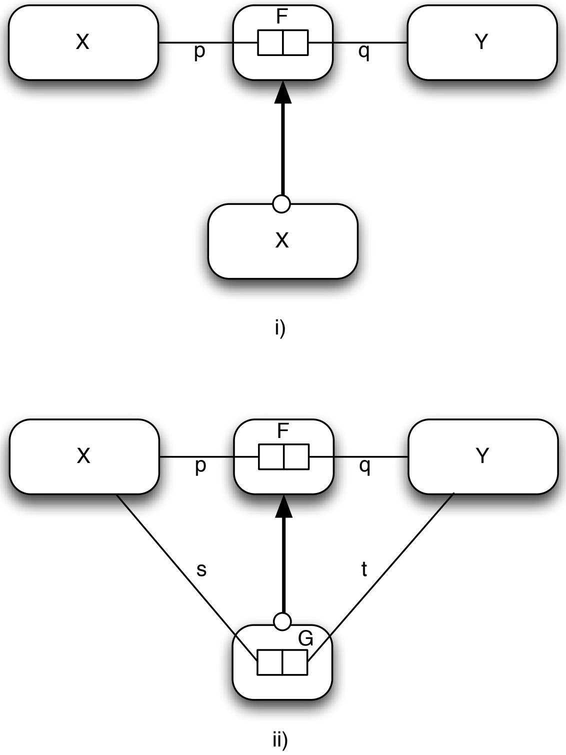


Fig. 5.13 Generalisation



i)

ii)

Fig. 5.14 Subtyping and objectification

For our example domain, we consider a bank. Figure 5.15 shows the top level abstraction of the banking domain. This schema displays five types: Bank, Client, Service, enjoy, of. The Bank type is an abstracted type and forms the top abstraction of the entire banking application. This is also the reason why the enjoy and of relationship types, together with the remaining object types playing a role in these relationship types, are drawn inside the Bank type. Both Client and Service types are abstractions themselves, although their underlying structure is not shown at the moment. When stepping down to a lower level of abstraction, the *void* in these types will be filled with more detail.

The Client and Service type are involved in a relationship type called enjoys. This is a many to many relationship where each client must at least enjoy one service and each service offering must be enjoyed by some person. The two black dots indicate that a client of the bank must indeed enjoy some service, and conversely each service must be used by some client. The arrow tipped bar spanning the two roles of the enjoys relationship type indicates that it is a many to many relationship. Similarly, the of relationship type models the fact that a bank has many clients, and clients can be client of many banks. The (name) suffix to Bank indicates that a

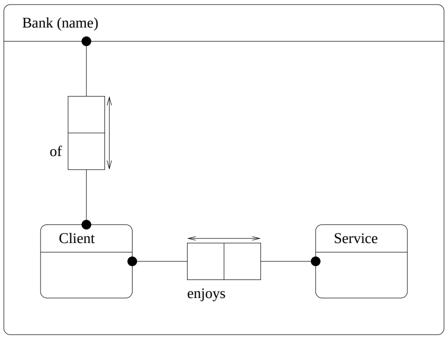


Fig. 5.15 The top diagram of the Bank domain

bank is identified by a name. Basically, the use of the (name) suffix is a graphical abbreviation of the schema fragment depicted in Figure 5.16. The broken ellipse of BankName type indicate that it is a *value type*; i.e. its instances are directly denotable (strings, numbers, audio, video, html).

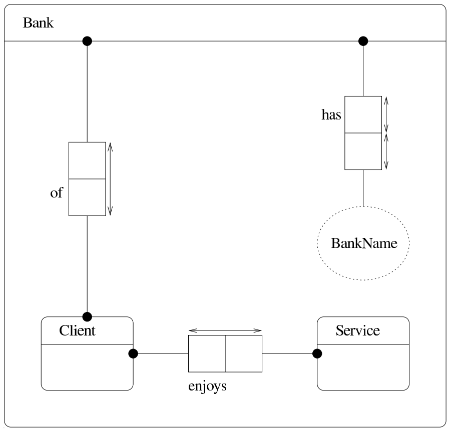


Fig. 5.16 Fully detailed top diagram

As a first refinement step we can now take a closer look at what a client is. The details of the Client type are shown in Figure 5.17. There we can see that each client is identified by a Client Nr, as indicated by the (nr) suffix to Client. Each client provides the bank with a unique address as indicated by the arrow tipped bar spanning the role of the lives at relationship type that is attached to Client. This address is mandatory for each client. This "mandatoryness" is indicated by the black dot. Address is a normal object type without any other types clustered to it. Therefore, it is drawn in the traditional ORM way using a solid ellipse. The (description) suffix to Address within the solid ellipse indicates that an address is identified by a description. This corresponds to the same underlying graphical abbreviation.

Clients must all provide at least one name, but they may have aliases. This leads to the arrow tipped bar spanning both roles of the has fact type, and the black dot on the client side. For authorization of transactions ordered by telephone or fax, the bank and the client agree upon a unique password. The combination of a password and address must uniquely identify a client (indicated in the diagram by the encircled U). Finally, clients may have a number of phone numbers at which they can be reached.

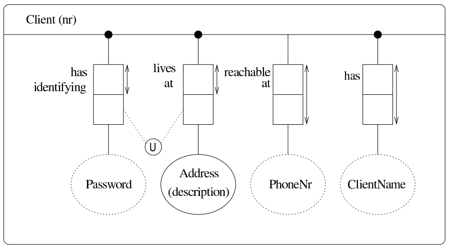


Fig. 5.17 Refinement of the client type

With respect to the abstractions, we can now say that the relationship types has identifying, lives at, reachable at, has (together with the types playing a role in these relationship types) are clustered to Client. For each abstracted type, like Client, such a clustering of types (from a lower level of abstraction) is provided. This could be an emptyset.

In this example we refer to relationship types used in the bank example by means of the text associated with these relationship types, such as has identifying. This text is a so-called *mix fix predicate* verbalization. These mix fix predicate verbalizations do not have to be unique. The verbalization has typically occurs numerous times in an average conceptual schema. For example: Client has Client Name and Client has Password. To uniquely identify relationship types (and types in general), each type receives a unique name. For instance Client Naming and Issued Passwords for the two earlier given examples.

The next refinement of the bank domain provides us with more details about the service types available from the bank. This is depicted in Figure 5.18. The Service type is a generalization of three basic types: Credit Card Account, Access Account, and Term Deposit Account. The Access Accounts and Credit Card Accounts are first combined into a so-called Statement Account. It should be noted that during a top-down modelling process, a type like Credit Card Account will start out as a ‘normal’ entity type like Address. However, as soon as other types are clustered to such an entity type, they become abstracted types.

The double lining around the Access Account type indicates that this type occurs in multiple clusterings. A CASE Tool supporting this kind of graphical representa-

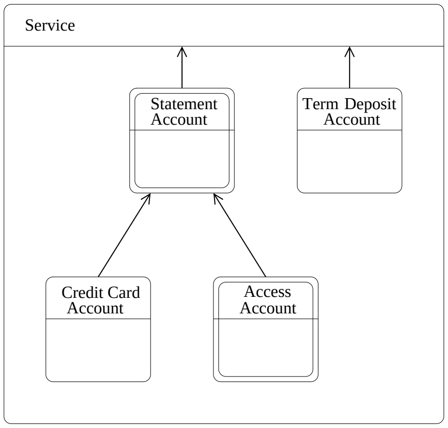


Fig. 5.18 Refinement of the service type

tion, could have a feature in which clicking on such a double lining results in a list of (abstracted) types in whose clustering this type occurs.

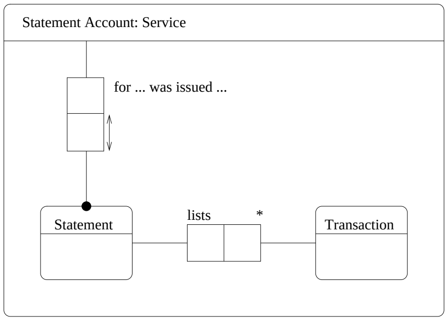


Fig. 5.19 Refinement of statement account

As stated before, a statement account is a generalization of an access account and a credit card account. The intuition behind a statement account is that for such an account regular statements are sent to the clients and that a transaction record is kept. These details of the statement account are shown in Figure 5.19. For each statement account, a number of statements can be issued. A statement lists a number of transactions This is captured by the lists fact type. This fact type is, however, derivable from the (to be introduced) issue date of a statement and the dates at which the transactions took place. This derivability is indicated by the asterisk.

One of the key features of the fact based modelling is inheritance of properties between types in specializations. Instances (populations) are inherited in the direction of the arrows. For example, each credit card account is a statement account. Other properties, like clustered types, are inherited downwards. Typically, proper-

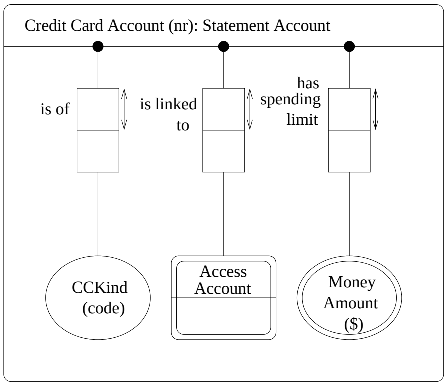


Fig. 5.20 Refinement of the credit card type

ties at the type level are inherited downward, while properties on the instance level are inherited upwards. The types clustered to Statement Account are therefore formally also part of the clusterings of Credit Card Account and Access Account. Nevertheless, to avoid cluttered diagrams, we have chosen not to show this inheritance explicitly in the diagrams. Therefore, the details of the Credit Card type do not show the details of Statement Account. The details of the Credit Card Type are provided in Figure 5.20. For each credit card the bank stores its kind, the spending limit, as well as the access account to which the credit card is linked. The suffix ": Statement Account" to "Credit Card Account (nr)" hints at the inheritance of the clustered types to Statement Account. In a CASE Tool supporting our technique, one could implement the facility that clicking on the Statement Account suffix leads to the inclusion of the clustered types introduced by Statement Account. Note that both Access Account and Money Amount have double lining, indicating that they occur in multiple clusters.

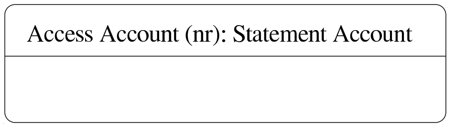


Fig. 5.21 Refinement of an access account

For Access Account, the details are shown in Figure 5.21. All extra information actually shown there is the identification of an access account; an Access Account Nr as indicated by the (nr) suffix. Similar to the Credit Card Account, all types clustered to Statement Account are also clustered to Access Account, but we do not display this graphically.

Figure 5.22 shows the details of a statement. Each Statement is issued on a unique date. This date, together with the Statement Account for which the Statement was issued, identifies each Statement. Note that we decided to draw some con-

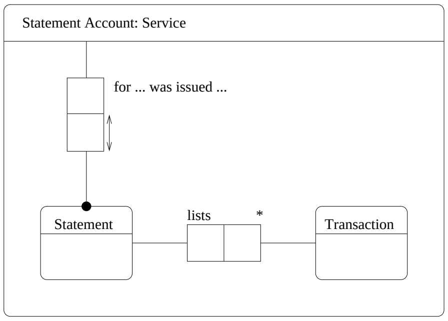


Fig. 5.22 Refinement of a statement

textual information of the Statement type to show how this type is identified. The for ... was issued ... and Statement Account types are not part of the clustering of Statement. The balance as listed on a Statement is, for obvious reasons, derivable from the Transactions that have taken place on this account.

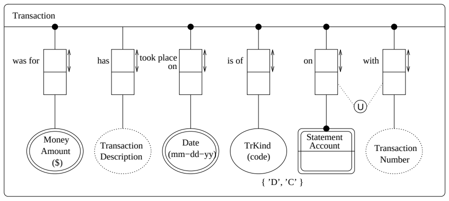


Fig. 5.23 Refinement of a transaction type

The refined view on a transaction is shown in Figure 5.23. A Transaction is identified by the combination of the account it is for and a unique (for that account) transaction number. Note that contrary to a Statement, all components needed for the identification of Transactions are part of the clustering. Each Transaction involves a certain money amount, occurs on a date, and is either a debit or credit transaction (depicted by TR Kind). Furthermore, for each Transaction, some (unique) description may be provided. This example also shows that we must allow for *mutually recursive* abstractions, as the Transaction and Statement Account refinements refer to each other.

Term deposits form a world on their own. This is elaborated in Figure 5.24. On each Term Deposit Account, a client can have a series of term deposits. Each time a Term Deposit matures, this term deposit can be rolled-over leading to a new Term Deposit on the current Term Deposit Account. A special kind of Term Deposit is the Long Term Deposit, which is a subtype of Term Deposit. As each subtype inherits all properties from its supertype, the Long Term Deposit type is an abstracted type as well. For these Long Term Deposits we store whether the deposit is to be automatically rolled-over into a new deposit (the short Term Deposits are of this kind

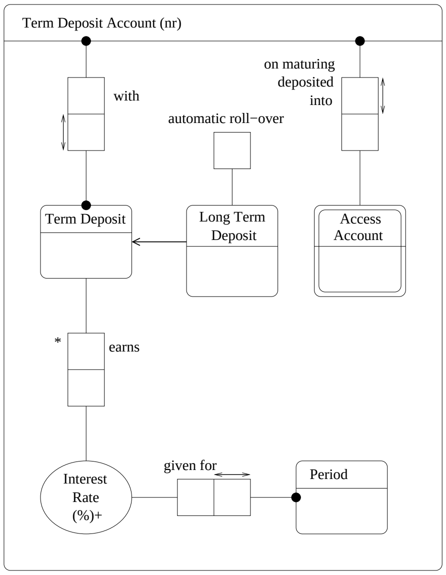


Fig. 5.24 Refinement of a term deposit account

by default). In the refinement of a Long Term Deposit, we shall also see what the so-called subtype defining rule for these Long Term Deposits is. Upon maturation, the invested amount including the interest accrued is transferred to a pre-nominated Access Account. Finally, the interest rate given on the deposit is derived from a table listing the Periods for which amounts can be invested. The details of the Period type are given below.

A Term Deposit itself is a clustering of the start and ending dates of the deposits and the money amount invested. This is depicted in Figure 5.25. A Long Term Deposit is a term deposit with a duration of more than 60 days. In Figure 5.26 the details of a long term deposit are shown, including the subtype defining rule. The Long Term Deposit type inherits all clustered types from Term Deposit, while not adding anything to this. Finally, the complete definition of the interest periods are given in Figure 5.27.

This completes the schema of the example domain. When modelling a domain like this, the modeler has the choice of using as many layers of abstraction as the modeler sees fit. We only provide a mechanism to introduce these abstractions and are (initially) not so much concerned with the ‘sensibility’ of abstraction steps. One may, for example, argue that the example given in this Section has been split up into too many abstraction levels.

Sometimes, an analyst may want to see the entire schema. This is quite easy to do by uniting all clusterings into one large schema. From the above discussed schema fragments, one can derive the complete ORM schema as depicted in Figure 5.28 by uniting all clusters. This is, however, still not the ‘lowest’ level at which an ORM

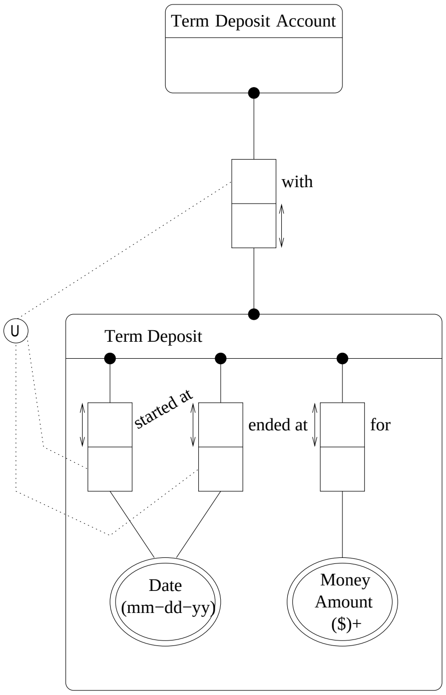
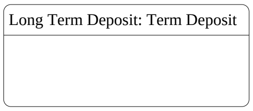


Fig. 5.25 Refinement of a term deposit



Long Term Deposit IS A
Term Deposit wich started at Date + 60 days > Date
at which ended THAT Long Term Deposit

Fig. 5.26 Refinement of a long term deposit

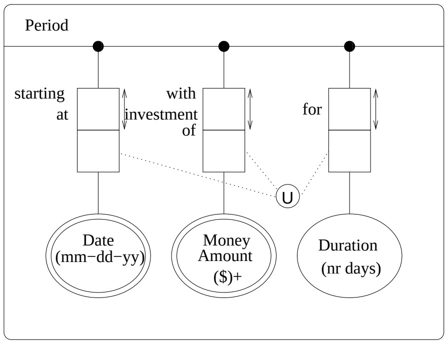


Fig. 5.27 Refinement of periods

diagram can be displayed, since we have used the standard abbreviations for simple identifications and the short notation for objectifications.

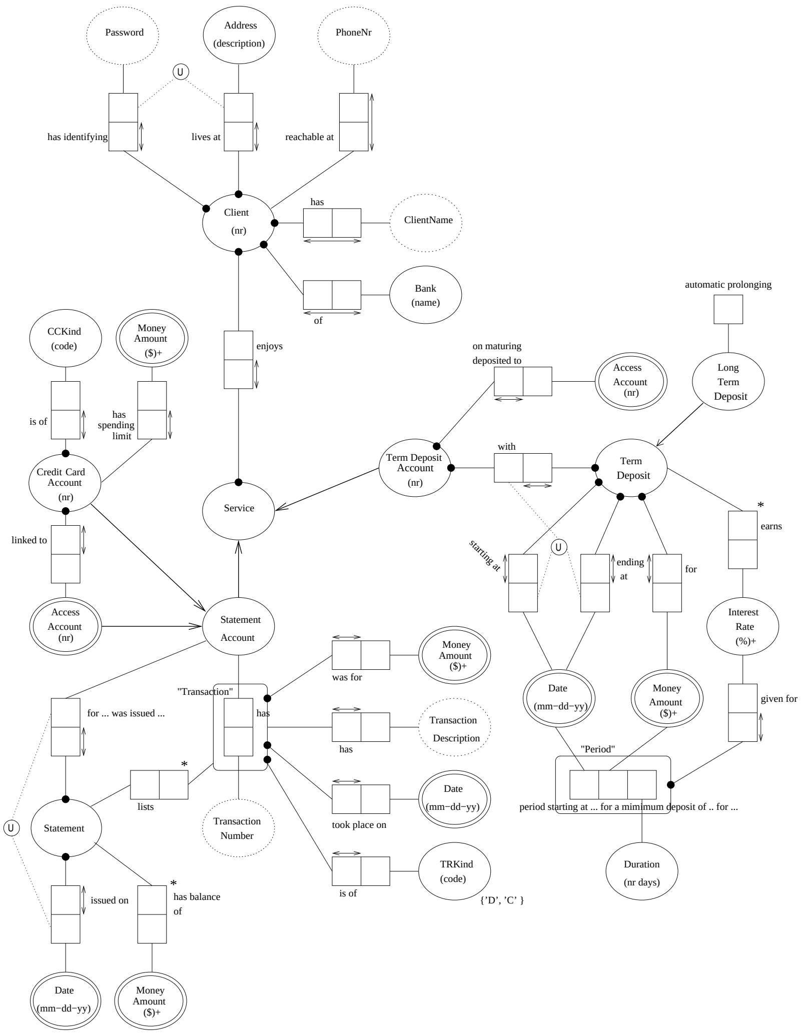


Fig. 5.28 Complete diagram of the Bank domain

Also when looking at a design procedure for ORM schemas as presented in [31] the decision to model a Transaction, say, as an objectification or a flat entity type is based on considerations of abstraction. When, for the modelling of the relationship types was for, has, took place on, and is of it is preferred to regard a transaction as an abstraction from its underlying relationships to a statement account and transaction number, then the objectified view is preferred to the flat entity view. This directly corresponds to the decision whether these underlying relationships should be clustered to the Transaction object type or not. Later we shall see that *set types*, *sequence types* and *schema typing* can be treated in a similar way. In [38, 39] it is shown that *set types*, *sequence types* and some other composed types are not fundamental when introducing a special class of constraints which correspond to the set theoretic notion of *axiom of extensionality*. This then allows us to regard set typing, sequence typing and schema typing as forms of abstraction.

The schema depicted in Figure 5.28 has the same *formal semantics* as the combination of all previous schema fragments. However, the *conceptual semantics* is different as the abstraction levels (the third dimension) are now missing. Schema abstraction is purely a syntactical issue, and thus carries no formal semantics. From the point of view of a modeler (and a participant of the universe of discourse), the abstractions do have a conceptual meaning. The abstractions represent certain choices of importance within the universe of discourse.

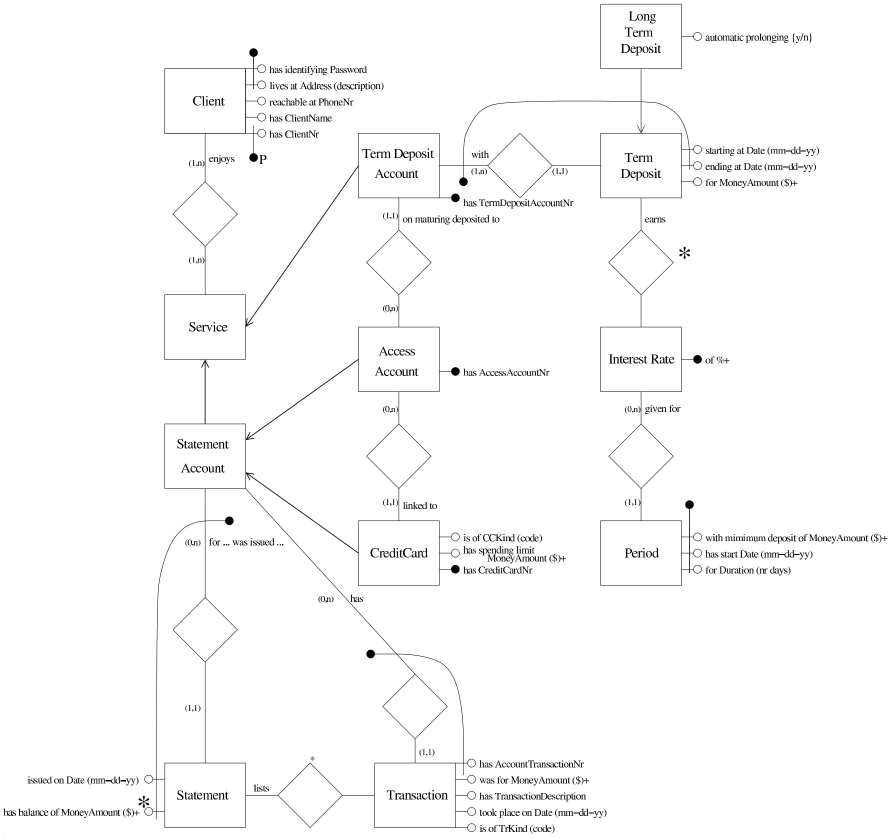


Fig. 5.29 Complete ER diagram of the Bank domain

An (E)ER view can easily be derived as well by uniting all clusterings except for the lowest ones, but interpreting these as attributes. The (E)ER view on this domain is given in Figure 5.29. The version we used there is based on the one discussed in [9]. Differing extended ER versions use different notations for this concept [20, 19, 22]. The names for attributes in this diagram are simply based on the verbalizations given in the ORM schema. For most ER modelers, the concept of using elaborate verbalizations is new. One could allow for the specification of specific attribute names to, for example, abbreviate with minimum deposit of MoneyAmount ($)+ to MinDeposit. In this article we do not discuss naming conventions in detail but rather focus on the underlying conceptual issues. In [14] we have provided a more detailed study of the relationship between different ER versions and ORM. A detailed case study is also presented there, in which the different concepts underlying these modelling techniques are related, together with a mapping of the (graphical) concepts between the two classes of data modelling techniques.

5.10 Complex object types

As a next step we will consider some construction mechanisms for the construction of complex types that partially build upon

5.10.1 Set types

In set theory we use $\wp(X)$ to denote the set consisting of all subsets of X (see for instance [60]). When modelling complex domains, we sometimes have the need to model set types being types whose instances can be regarded as being sets of other instances. This notion is the same as the notion of grouping introduced in the IFO data model [3]). An illustrative example, taken from [38], is shown in Figure 5.30. A Convoy is taken to consist of a set of Ships, where this set of ships really *identifies* the convoy. In other words, if two convoys contain the same set of ships, they really are the same convoy. This is actually similar to the existentiality axiom from set theory:

$$\forall_i \, [i \in X \Leftrightarrow i \in Y] : X = Y$$

In Figure 5.30 the existential uniqueness is expressed by the circle with the two horizontal bars. If there would be only one bar, this would be normal uniqueness of the associated role. The extra (slightly shorter) bar signifies this to be an *existential* uniqueness constraint.

Using the abstraction mechanism from the previous Section, we are able to more introduce a number of shorthand notations for set types. These are depicted in Figure 5.31.

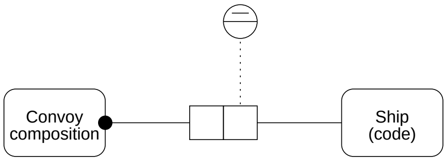


Fig. 5.30 Convoy of ships

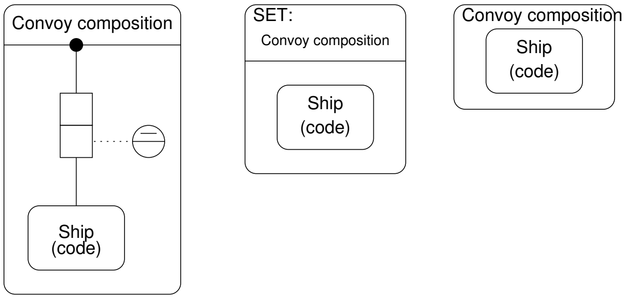


Fig. 5.31 Shorthand notations for convoys of ships

5.10.2 Multi-set types

A variation of sets is a multi-set. In a multiset, elements can occur multiple times. Using the existantial uniqueness constraints, a multi-set can be modeled as depicted in Figure 5.32. In the depicted domain, a train composition class is defined as a multi-set of types of carriages.

$$\forall_{i,f}\left[i \in^{f} X \Leftrightarrow i^{f} \in Y\right] : X = Y$$

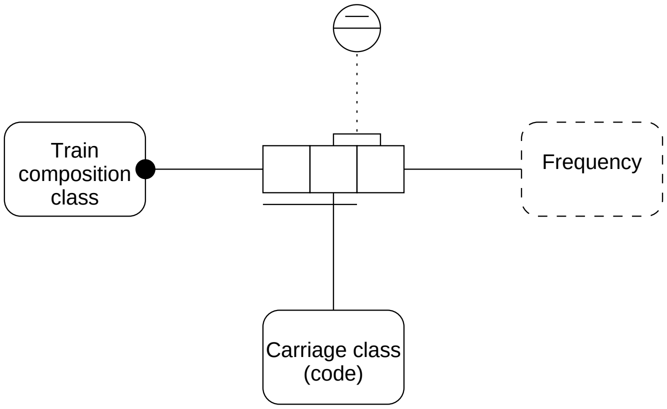


Fig. 5.32 Trains as sequences of carriages

Using the abstraction mechanism, we are again able to more introduce a number of shorthand notations for multi-set types. These are depicted in Figure 5.33.

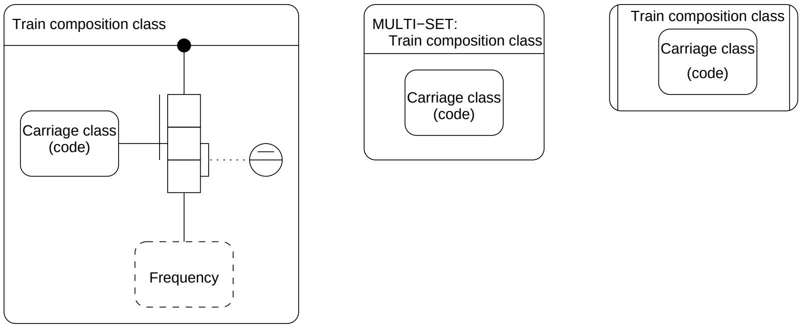


Fig. 5.33 Shorthand notations for trains as sequences of carriages

5.10.3 Sequence types

A specific train consists of a sequence of carriages. To model this compactly, we introduce the notion of a sequence type. This leads to the situation as depicted in Figure 5.34.

$$\forall_{i,p} [i = X[p] \Leftrightarrow i = Y[p]] : X = Y$$

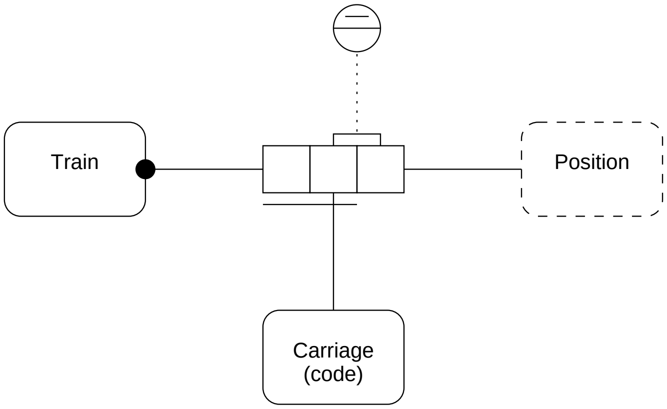


Fig. 5.34 Train as a sequence of carriages

Using the abstraction mechanism, we are again able to more introduce a number of shorthand notations for sequence types. These are depicted in Figure 5.35.

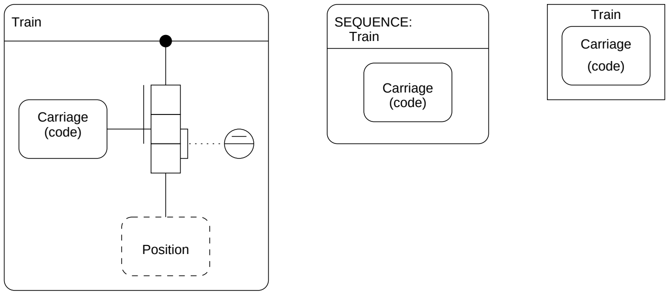


Fig. 5.35 Shorthand notations for trains as sequences of carriages

5.10.4 Schema types

In some situations we need types whose instances are really entire populations of other (smaller) schemas. An example of such a situation is shown in Figure 5.36.

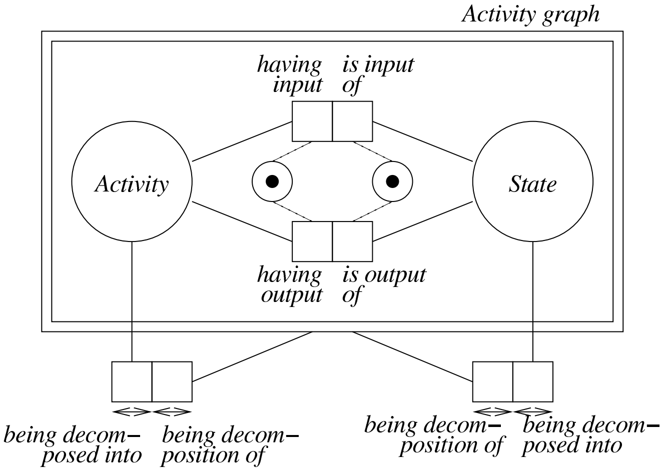


Fig. 5.36 Activity graphs usign a schema type

Using the abstraction mechanism, we are again able to more introduce a number of shorthand notations for schema types. These are depicted in Figure 5.37.

{*We should also add the grammar box!!*}

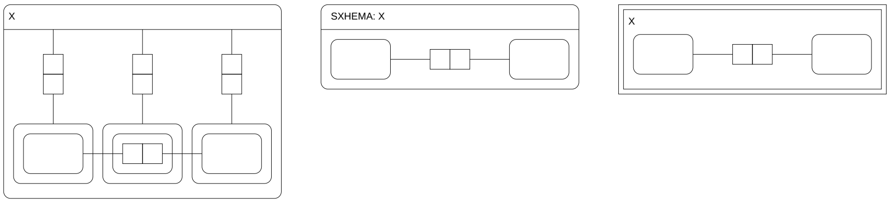


Fig. 5.37 Shorthand notations for schema types

5.11 Questions

1. Given the situation:

   > *A person with name Erik is writing a letter to his loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.*

   Produce a graph consisting of concepts and links depicting this domain.
2. Consider the following case:

   > *Een onderneming produceert en verkoopt een tiental soorten gevulde chocolade-artikelen. De verkoop geschiedt aan grossiers tegen prijzen die voor lange tijd vast zijn. In verband met achteruitgang in kwaliteit wordt op de verpakking een uiterste verkoopdatum vermeld. Alle afleveringen geschieden met eigen auto's.*
   >
   > *Voor de produktie van chocolade importeert de inkoopafdeling van de onderneming verschillende soorten cacaobonen uit tropische landen. Daartoe worden inkoopcontracten afgesloten die de behoefte voor ca. een half jaar dekken. De cacaobonenprijs is aan sterke schommelingen onderhevig. De ingekochte partijen hebben belangrijk uiteenlopende vetgehaltes, hetgeen mede in de inkoopprijs tot uitdrukking komt.*
   >
   > *De cacaobonen ondergaan afzonderlijk per partij in de voorbewerkingsafdeling enkele machinale bewerkingen, zoals zuiveren, schillen, breken, branden, malen en walsen.*
   >
   > *Aan het onstane halffabrikaat worden door de afwerkingsafdeling suiker, smaakstoffen en – in verhouding tot het vetgehalte – cacaoboter toegevoegd. Het aldus verkregen halffabrikaat is cacaomassa van een bepaalde standaardkwaliteit, dat in speciaal daartoe geconditioneerde opslagtanks wordt bewaard. De verschillende benodigde vulsels worden ingekocht bij derden. Naar rato van de ontwikkeling van de verkoop en de gewenste voorraadvorming worden de eindprodukten gemaakt. Dit geschiedt in één arbeidsgang met behulp van automatische vorm-, vul- en droogmachines.*
   >
   > *In de pakafdeling worden de goedkopere soorten gevulde chocolade automatisch en de duurdere soorten met de hand in sierdozen verpakt, waarna opslag in een magazijn volgt. Bij alle bewerkingen ontstaan gewichtsverliezen.*
   >
   > *In verband met de kwaliteitsachteruitgang kunnen de grossiers de niet tijdig door hen verkochte artikelen retourneren, mits dit gebeurt binnen 10 dagen na de uiterste verkoopdatum; meestal geschiedt deze teruglevering via de chauffeurs. De teruggenomen artikelen worden vernietigd. Creditering vindt plaats voor 20 van de door hen betaalde prijs. Verrekening hiervan geschiedt slechts bij gelijktijdige nieuwe afname.*
   >
   > *Elk van de artikelen is voorzien van een of twee cadeaubonnen, afgedrukt op de verpakking. De waarde van deze bonnen is e 0,10 per stuk. Op de artikelen met een prijs tot e 5,- komt één, op de overige artikelen (tussen e 5,- en e 11,-) komen twee bonnen voor.*

> *Op deze bonnen kunnen cadeau-artikelen (hand- en theedoeken e.d.) zonder bijbetaling worden verkregen.*
>
> *Voorts kunnen op deze bonnen meer duurzame gebruiksgoederen tegen verlaagde prijs worden verkregen. Hiervoor wordt elk halfjaar een folder uitgegeven, waarin per artikel is aangegeven hoeveel bonnen moeten worden ingeleverd en hoeveel daarnaast moet worden bijbetaald. In het algemeen is het door de afnemers bij te betalen bedrag iets lager dan de inkoopprijs voor de fabriek. Veelal dient de halfjaarlijkse behoefte door de fabrikant in één keer te worden besteld; latere aanvulling is in het algemeen niet mogelijk.*
>
> *Op de duurzame gebruiksgoederen wordt veelal garantie of service verleend. Hiervoor is met een gespecialiseerd bedrijf een contract afgesloten waarbij tegen een eenmalig vast bedrag per apparaat de garantie- en serviceverplichtingen worden overgedragen.*

Answer the following questions:

a. Produce elementary facts for this domain.
b. Produce an ORM model for this domain.

3. Given the following populations: Pop(Carnivore) $= \{a,b,c\}$, Pop(Omnivore) $=$ $\{d,e\}$ and Pop(Herbivore) $= \{f,g\}$. What are the populations of Animal, Flesh eater and Plant eater?
4. To have electrical power supplied to one's premises (i.e. building and grounds), an application must be lodged with the Electricity Board. The following tables are extracted from an information system used to record details about any premises for which power has been requested.

   The following abbreviations are used: *premises# = premises number*, *qty = quantity*, *nr = number*, *commercl = commercial*. Each premises is identified by its premises#.

   The electricity supply requested is exactly one of three kinds: "new" (new connection needed), "modify" (modifications needed to existing connection), or "old" (reinstall old connection). "Total amps" is the total electric current measured in Amp units. "Amps/phase" is obtained by dividing the current by the number of phases.

premises#	city	kind of premises	kind of business	dog on premises	breed of dog	qty of breed	supply needed
101	Brisbane	domestic	.	yes	Terrier	2	new
202	Brisbane	commercl	car sales	no	.	.	modify
303	Ipswich	domestic	.	yes	Alsatian	1	old
					Poodle	1	
404	Redcliffe	commercl	security	yes	Alsatian	3	new
					Bulldog	2	
505	Brisbane	domestic	.	no	.	.	modify
606	Redcliffe	commercl	bakery	no	.	.	old
...	...	...	...	...	...	...	...

Further details about new connections or modifications:

	load applied for (if known)			wiring	expected date for
premises#	total amps	nr phases	amps/phase	completed?	wiring completion
101	200	2	100	no	30-06-03
202	600	3	200	yes	.
404	.	.	.	no	01-08-03
505	160	2	80	no	30-06-03
...	...	...	...	...	

The population is significant with respect to mandatory roles. Each premises has at most two breeds of dog.

Produce a fact-based model for this domain. Use specialization when needed. Include *uniqueness*, *mandatory role*, *subset*, *occurrence frequency* and *equality constraints*, as well as *value type constraints* that are relevant. Provide meaningful names.

If a fact type is derived it should be asterisked on the diagram and a derivation rule should be supplied.

Produce both a flat fact-based model, as well as a version that uses abstraction/decomposition to split this domain into more comprehensible chunks.

Chapter 6
Modelling active domains

Enterprises are open active systems. In other words, when modelling an enterprise, we are modelling an open and active domain. This means that the facts we would see appearing in our logbook (see Figure 5.3) really refer to two kinds of things: *static* phenomenon and *changing* phenomenon.

6.1 Temporal ordering

Now consider the following verbalisations (at the type level):

A Person fills in a Form
A Person is examined by a Doctor
A Doctor produces a Diagnose
A Doctor writes a Prescription

This leads to the situation as depicted in Figure 6.1.

Thus far we have not discussed properties pertaining to temporal ordering. Suppose now that in this domain:

Before a Person can be examined by a Doctor, they should have filled in a Form.
Before a Doctor produces a Diagnose, a Person should have been Examined.
Before a Doctor writes a Prescription, a Person should have been Diagnosed.

This is, however, is still an incomplete picture. The production of a diagnose and the writing of a prescription should all pertain to the same person. Even more, as a person may visit a doctor twice for two different reasons, the diagnose and prescription really pertain to one specific doctor visit. This leads to the situation as depicted in Figure 6.2.

But also consider the situation as depicted in Figure 6.3. What is the semantic difference?

As a graphical abbreviation, we will use the notation as depicted in Figure 6.4 and Figure 6.5 respectively.

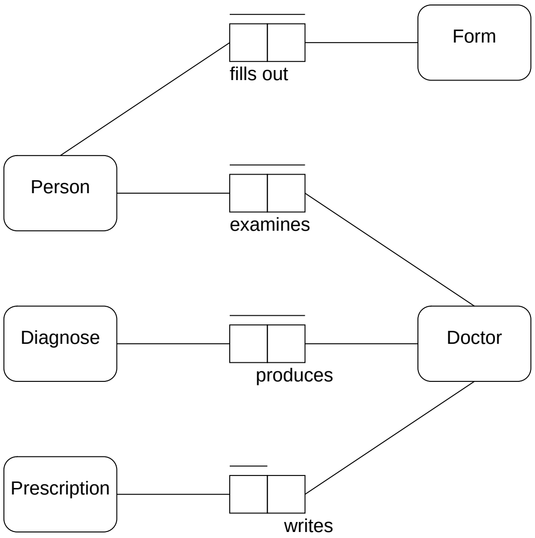


Fig. 6.1 Basic model of a visit to a Doctor

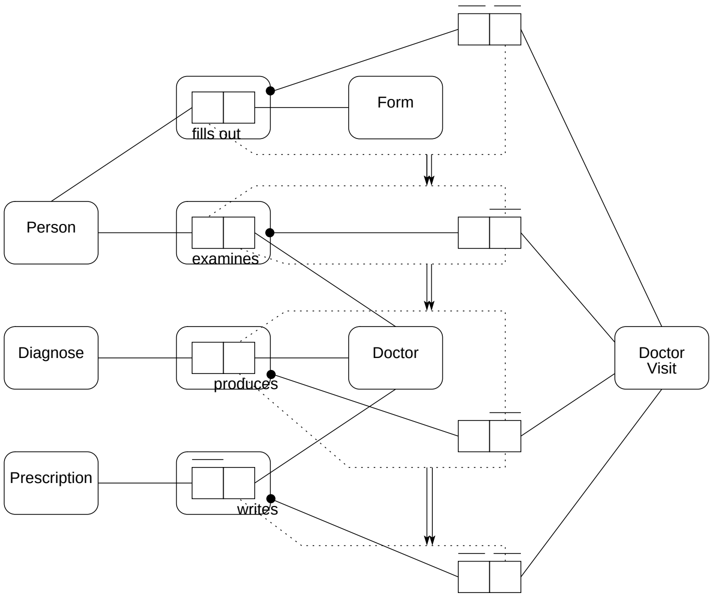


Fig. 6.2 Model of a visit to a Doctor with explicit entity

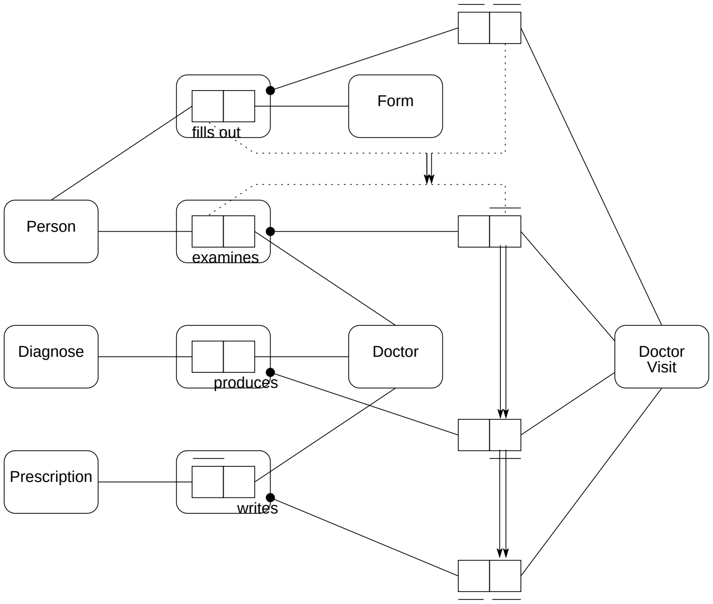


Fig. 6.3 Model of a visit to a Doctor with alternative semantics

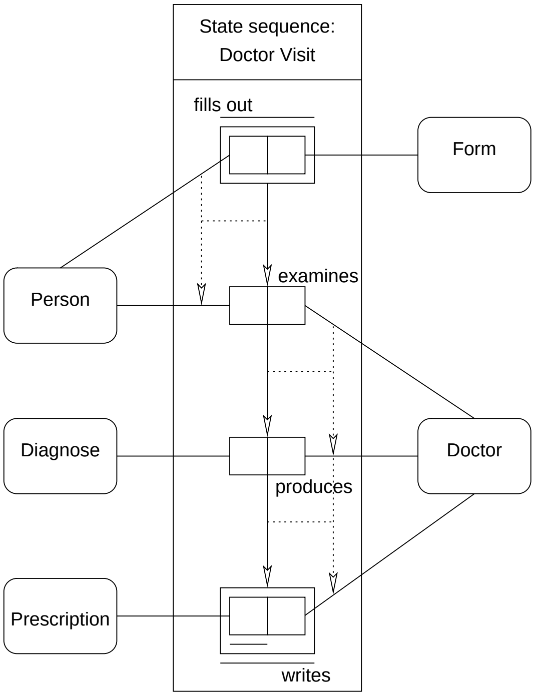


Fig. 6.4 Compact model of a visit to a Doctor

The temporal dependencies can actually involve more complex relationships using split and join operators such as illustrated in Figure 6.5. In one of the next Chapters we will discuss the semantics of such complex temporal dependencies in more detail.

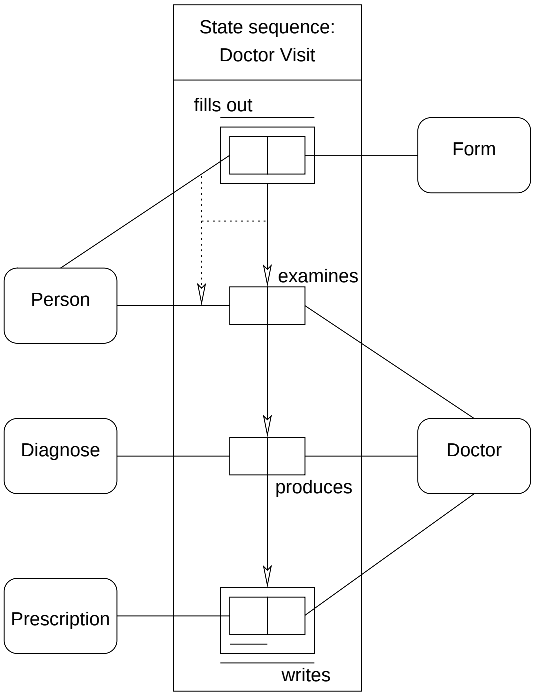


Fig. 6.5 Compact model of a visit to a Doctor with alternative semantics

{ *This Chapter should really also include a more detailed discussion on the semantics of temporal dependencies (see Figure 6.6, including complex operators such as shown in Figure 6.7.* }

6.2 Object-life cycles

When focusing on the roles that may be played by the instances of some object type and their temporal dependencies, we are able to define (at the type level) the 'usual' life-cycle of such objects. An example is shown in Figure 6.8.

In Figure 6.9 we have depicted some abbreviations of typical constructs that may be used in the depiction of object-life cycles. These abbreviations are based on [27].

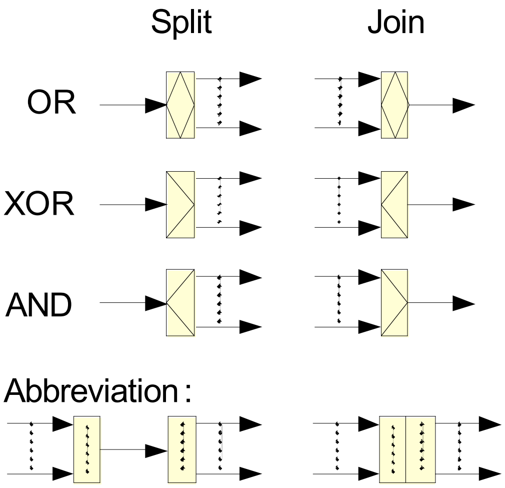


Fig. 6.6 Complex temporal dependencies

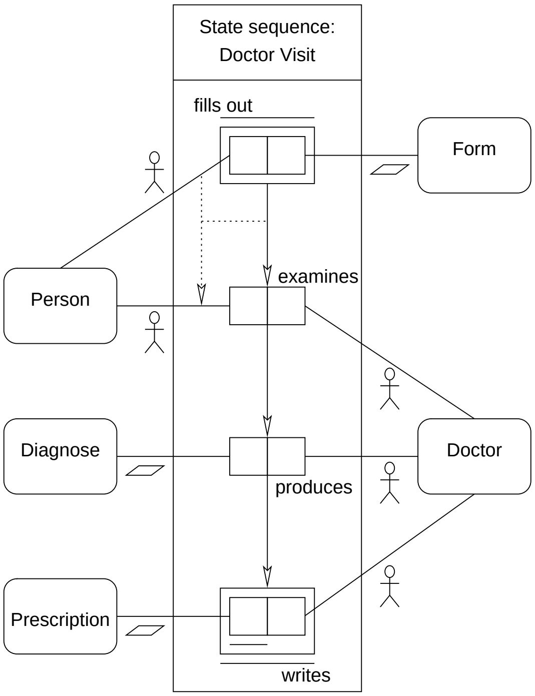


Fig. 6.7 Doctor visit with temporal dependencies

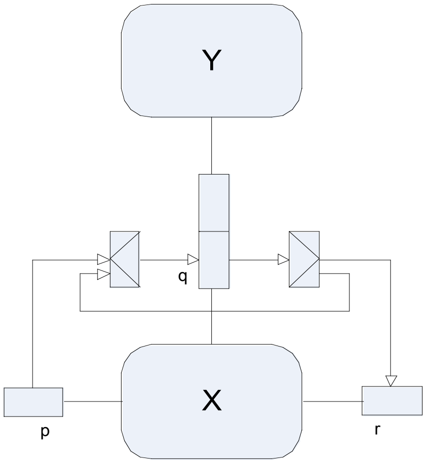


Fig. 6.8 Example object-life cycle

{*In a future version of these lecture notes, we should incorporate more of the constructions used in [27], more particularly defining derived "temporal roles" in terms of regular expressions such as "$p.(q+c).c$"*}

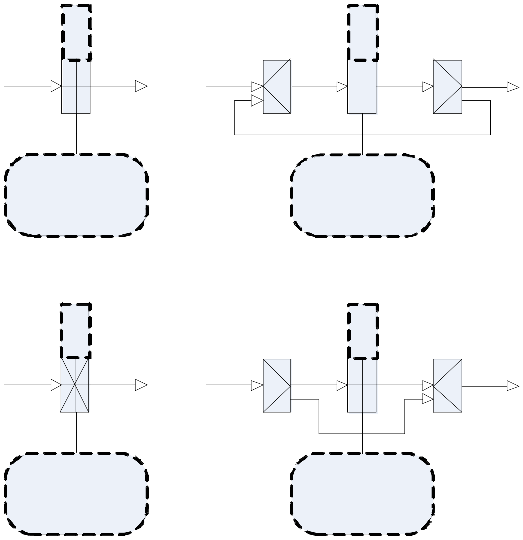


Fig. 6.9 Abbreviations for life-cycles

6.3 Facts about active domains

When modelling active domains, we will make a distinction between four kinds of fact (instances and types):

Change – Facts referring to changes in the domain. Examples are: *John's hair has turned grey, James has turned 20* and *stock levels have dropped 20 units*.

State – Facts referring to static phenomenon. These are facts not referring to change in the domain. Examples are: *John has blue eyes*, *the house is green*, *Jim weighs 80 Kilo's* and *John works for James*.

Initial-state – States that are actual when one first starts observing the activities in the domain.

Result-state – States that are the result of a change.

This classification is illustrated in Figure 6.10. With this classification we have added another level to out expanding meta-model of conceptions:

1. Conceptions exist.
2. Conceptions consist of elements
3. Conceptions consist concepts, links and decomposition
4. Conceptions also comprise types and instances
5. Special concepts in a conception are: Facts / Roles / Objects
6. Specific kinds of facts are: Change / State / Initial-state / Result-state.

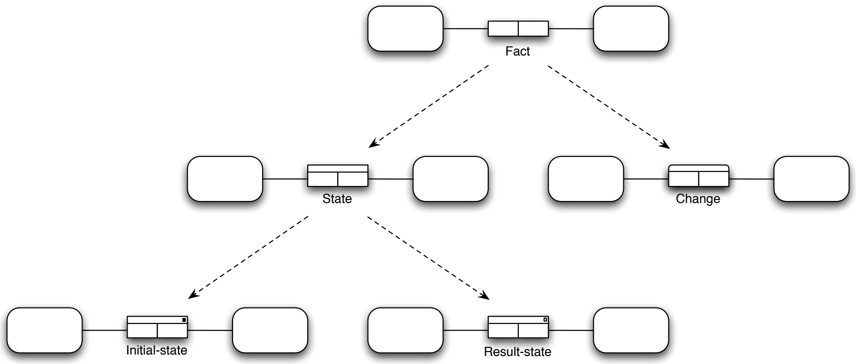


Fig. 6.10 Classes of facts

6.4 A classification of roles in change

When looking at changes that take place in an active domain, such as:

- *John's hair has turned grey*,
- *The car was painted red by Ian*,
- *James has turned 20* and
- *Stock levels have dropped 20 units*.

the objects playing a role in these facts play roles of differing classes. For example, Ian is the person *painter* doing the actual painting, and therefore plays an *agentive* role.

In this Section, we use the natural language approach as also advocated by ORM, to investigate the different classes of roles that may occur in facts. This requires the immediate introduction of some new concepts into our ontology, such as agentive, experiencing, circumstantial and predicative roles. These concepts allow us to reason about such things as: *when* does something happens (triggering), *what* happens (action), *who* / *what* makes it happen (agent), *who* / *what* does it happen *to* (subject) in *which* / *what* circumstances (context).

With these new concepts, we can typically take ORM domain models and "annotate" them in terms of the refined concepts. We will base this process of annotation on linguistic foundations, much in the same vein as the modelling approach from the *Domain Modeling* course. Based on these annotated ORM models, we will (in the next part of these lecture notes) be able to mechanically derive process models in a modelling notation (ArchiMate [59]) that is particularly suited for the modelling of business processes.

Let us now analyze this closer from a natural language perspective. Consider, once again, the following domain:

> *A person with name Erik is writing a letter to his loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.*

with elementary facts:

A person is writing a letter
This person has the name Erik
This letter has a romantic nature
This letter has intended recipient Erik's loved one
The writing of this letter by Erik, occurs on a mid-summer's day
The writing of this letter by Erik, is done using a pencil
The writing of this letter by Erik, is done while the cat is watching
The writing of this letter by Erik, is taking place at a desk
This desk is located in a room
This room is romantically lit

As mentioned before, within these elementary facts, several *players* can be discerned. In the above example, we can isolate the players and facts as follows:

[A person] is writing [a letter]
[This person] has [the name Erik]
[This letter] has a [romantic nature]
[This letter] has intended recipient [Erik's loved one]
[The writing of this letter by Erik], occurs on a [mid-summer's day]
[The writing of this letter by Erik], is done using [a pencil]
[The writing of this letter by Erik], is done while [the cat] is watching

[The writing of this letter by Erik], is taking place at [a desk]
[This desk] is located in [a room]
[This room] is lit in [a romantic way]

The writing of the letter is the central fact in the above domain. Several degrees of activeness exist with regards to the roles played in this domain, relative to the writing of the letter. This is where we find inspiration in theories regarding verbs and the 'things' that may play a (functional) role in these verbs. We limit ourselves to those classes that are indeed relevant when considering activity in systems. In decreasing scale of activity:

Agentive role – A role where the player is regarded as carrying out an activity.
In the example domain: The person.
Two sub-classes may be identified:

Initiating role – An agentive role, where the player is regarded as being the initiator of the activity.

Reactive role – A non-initiating agentive role.

Experiencing role – A role where the player is regarded as experiencing/undergoing an activity or change.
In the example domain: a letter, a loved one and the cat.
Three sub-classes may be identified:

Patientive role – An experiencing role, where the player is regarded as purposely undergoing changes (including its very creation).

Receptive role – An experiencing role, where the player is regarded as the beneficiary/recipient of the results of the activity.

Observative role – An experiencing role, where the player is regarded as observing/witnessing the activity.

Contextual role – A role where the player is regarded as being a part of the context in which the activity or change takes place.
Three sub-classes may be identified:

Instrumental role – A role where the player is regarded as being an instrument in an activity.
In the example domain: a desk and a pencil.

Locative role – A role, where the player is regarded as being the location of an activity, in terms of a spatial or temporal orientation.
In the example domain: the desk, the room and mid-summer's day.

Catalysing role – A role, where the presence of the player is regarded as being beneficial (either in a positive or a negative way) to an activity.

In the example domain: the room lit in a romantic way.

Predicative role – A role pertaining to the ownership of properties and/or objects.
Two subclasses may be identified:

Attributive role – A role where the player is regarded as being attributed to some other player.

Possessive role – A role where the player is regarded as being the owner of some other player.

The choice between these different levels of role is subjective. It depends on the viewer. The players of the four main classes of roles are regarded as *agents*, *subjects*, *contextuals*, and *predicators* respectively. Note that one object type/instance might be in multiple classes.

Any fact type/instance in which either an experiencing or an agentive role is present is a *change* since some thing is doing/experiencing something, which is a change in the domain. All other facts are states.

In our example, we have two facts which are a change:

1. [A person] is writing [a letter]
2. [The writing of this letter by Erik], is done while [the cat] is watching

6.5 Change types

Based on the above discussions, we can identify which roles are of which class, and mark this in the ORM model. For the purpose of these lecture notes, we are only interested in marking the agentive and experiencing roles.

For the example from Figure 6.5 we have the situation as depicted in Figure 6.11.

6.6 Questions

1. Produce a new version of our meta-model in which you include the new concepts of change, state, result-state and initial-state.
2. Refine this meta-model even further by adding the main role classes. Define derivation rules for *agents*, *subjects*, *contextuals*, and *predicators* respectively.
3. Given the situation:

   > *A person with name Erik is writing a letter to his loved one, at the desk in a romantically lit room, on a mid-summer's day, using a pencil, while the cat is watching.*

   Produce a graph consisting of facts, objects and roles depicting this domain.
4. Consider the following domain:

   > *Docent Proper voert de vakgegevens van Architectuur en Alignment in in het management informatiesysteem.*

   Wat zijn hier de concepten en de links? Wat zijn de changes, agents, subjects en predicators in dit domein?
5. Stel je maakt een ontwerp voor een geldautomaat. Wat zijn voor dat domein de belangrijkste systeem entiteiten en hun onderlinge relaties? Hoe werken ze

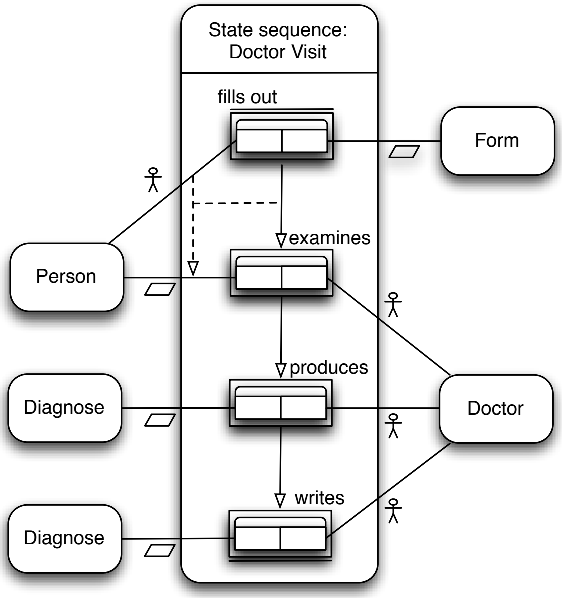


Fig. 6.11 Doctor visit with classification of roles

samen? Wat zijn hier de concepten en de links? Wat zijn de acties, actoren, actanden en predications.?

Chapter 7
The Act of Modelling

In this Chapter, which is based on the work reported in [80, 41], we turn to the question *how to model a domain*; a question to which there is no simple, one-size-fits-all answer.

7.1 What to model?

Before modelling, it is important to identify:

- What is the modelling goal?
- Who is the intended audience?
- What should the model be about?
- What will be done with the model descriptions?

The answers to these questions should really be included with the model descriptions as a kind of disclaimer/positioning of the resulting descriptions.

7.2 The modelling challenge

Some modelling approaches, such as NIAM [65] and ORM [32], suggest or prescribe a detailed procedure. Practice shows, however, that experienced modellers frequently deviate from such procedures [99]:

> *In most cases, the* information engineers *stated that they preferred to pay attention to a specific part of the problem domain, usually to fill clear lacunae in their insights in the problem domain. Their momentary needs strongly influenced the order in which the several modelling techniques were used. Modelling techniques were used as a means to increase insights or to communicate insights, be it in the problem domain itself or in a specific solution domain.*

Yet deviating from a modelling procedure should be done with some caution. While a pre-defined modelling procedure should never become "an excuse to stop thinking", situational specificity should not become an excuse for taking an ad-hoc approach to the modelling effort. A more stable anchor is needed upon which modellers can base themselves when making decisions during the modelling process. We believe that domain modelling requires a goal-bounded and communication-driven approach. With goal-bounded we hint at the fact that when modelling a domain, a modeller is confronted with a plethora of modelling decisions. These decisions range from the modelling approach used, the intended use of the results, to decisions pertaining to the model itself. For example:

- What parts of the domain should be considered relevant?
- What is the desired level of detail and formality?
- To what level should all stakeholders agree upon the model?
- Should the model be a representation of an actual situation (*system analysis*) or of a desired situation (*design*)?
- Should the model be a representation of *what* a system should do, or should it be a representation of *how* a system should do this?
- Should a certain phenomenon in the domain be modelled as a relationship, or is it an object on its own?

Having an explicit, and articulated, understanding of the modelling goals provides modellers with guidance in making the right decisions with regards to the above mentioned issues. Modelling goals, therefore, essentially provide the *means* to *bound* modelling space.

In most situations where a domain needs to be modelled, the modeller cannot merely passively observe the domain. Modellers will need to interact with representatives from the domain. These representatives then become *informers* (who are likely to also have a stake with regards to the system being developed). Therefore, modellers will need to communicate intensively with the informers in order to refine the model. What is more, numerous domain models that are produced during system development will need to be accepted and agreed upon – validated – by the informers (being stakeholders of the future system). The claim has often been voiced that in modelling practice, 'the process is just as important as the end result', suggesting that a correct end-result is not always a guarantee for success. A domain model should ideally be a product of a *shared understanding of a domain's stakeholders*. It requires a 'buy-in' by all stakeholders involved. A domain model that is correct from a theoretical or technical point of view but does not have the required support from the key stakeholders is arguably worse than a domain model with some flaws that does have such support.

A modellingprocess can thus be seen as a communication-driven process [29, 98]. The principles of natural language driven modelling approaches [65, 21, 55, 32] can be used as a basis for shaping the communication process between informer and modeller.

7.3 The process of modelling

In general, the goals underlying (business) domain modelling are [12]:

- articulate clear and concise meanings of business domain concepts and
- achieve a shared understanding of the concepts among relevant stakeholders.

Based on the results reported in [40], we consider domain modelling in the context of system development to chiefly concern three streams of (mutually influencing) activities:

Scoping environments of discourse – The aim of this stream of activities is to scope the environments of discourse that are relevant to the system being developed, and determine the set of actors associated to each of these environments.

Concept specification – For each of the identified environments of discourse, the relevant business domain concepts should be specified in terms of their:

- meaning,
- relationships to other concepts (and the constraints governing these relationships) and
- possible names used to refer to them.

Concept integration – The concepts as identified and defined in the different environments of discourse may well clash. As a part of this, homonyms and synonyms are likely to hold between different terminologies. The aim of this stream of activities is to determine how to deal with this, and act upon it.

Since these streams of activities can be expected to influence each other, it is not likely that they can be executed in a strict linear order.

In general, the processes that aim to arrive at a set of concepts together with their meaning and names, are referred to as *conceptualisation processes* [40]. When, as in the context of software development, conceptualisation is performed deliberately, as a specific task and with a specific goal in mind, it is referred to as an *explicit conceptualisation process*. The above mentioned stream of activities called concept specification is such an explicit conceptualisation process. In [40, 12] a reference model for conceptualisation processes is provided. This reference model distinguishes five streams of activities or *phases*:

Assess domain and acquire raw material – Domain modelling always begins with a brief scan or assessment of the domain to get a feeling for scope, diversity and complexity of the domain, as well as to identify the relevant stakeholders for the domain (usually but not necessarily a subset of the project stakeholders). In addition, the activity aims to bring together input documents of all sorts that provide a basic understanding of the environment of discourse that is relevant to the environment of discourse under consideration.

Scope the concept set – In this phase, formal decisions are to be made regarding the concepts that somehow play a role in the environment of discourse and how these concepts interrelate.

Select relevant concepts – The goal of this phase is to focus on those concepts in the environment of discourse that bear some relevance to the system to be developed. These are the concepts that should be defined and named formally in the next step.

Name and define concepts – All of the concepts selected in the previous phase should be named and defined. Defining the concepts may also include the identification of rules/laws/constraints governing instances of the defined concepts.

Quality checks – Final quality checks on the validity, consistency and completeness of the set of defined concepts.

These streamsshould essentially be regarded as sub-streams of the concept specification stream.

7.4 Ambition levels for modelling

We have made a distinction between four levels of ambition at which a modeller may approach the task of modelling a domain. These levels can also be regarded as the order in which a novice modeller may learn the art of domain modelling:

Singular – This level ofambition corresponds to the modelling approaches as described in e.g. NIAM [65] and ORM [32]. It involves the modelling of a *single* environment of discourse based on complete input; usually in terms of a *complete* verbalisation of (*only*) the relevant parts of the domain.

Elusive – At this level of ambition, modellers need to cope with the unavoidable iterative nature of the modelling process. As a modelling and/or system development process proceeds, the insight into the domain may increase along the way. This replaces the idealized notion of completeness of input with one of incremental input. The increments in the model are not related to a changing domain, but rather to improved ways of conceptualizing it.

Pluriform – At this next level of ambition, we recognize the fact that when developing a realistic system, we do not simply deal with one single unified environment of discourse (and related terminologies and concepts), but rather with a number of interrelated environments of discourse [81].

Evolving – The final ambition level recognizes the fact that domains themselves are not stable; they evolve over time [81]. As a result, what may have started out as a correct model of a domain, may become obsolete due to changes in the domain. New concepts may be introduced, or existing ones may cease to be used. However, subtle changes may occur as well, such as minor changes in the meaning of concepts, or the forms used to represent them.

In the next Section, we will discuss domain modelling at the *singular*, *elusive* and *pluriform* levels of ambition. The *evolving* level is omitted for now.

7.5 Meeting the challenge

This Section aims to discuss the domain modelling process with respect to three of the identified levels of ambition: *singular*, *elusive* and *pluriform*. We will structure our discussion by using the framework of activity streams for domain modelling as introduced in the previous Section.

At the first level of ambition we are only interested in the modelling of a single environment of discourse based on complete input. In terms of the above framework for domain modelling, this ambition level assumes that:

- No (further) scoping of the environment of discourse is needed.
- The domain has been assessed and raw material is available.
- Concept integration only needs to take place within the given environment of discourse.

Natural language driven modelling approaches like NIAM [65] and ORM [32] concern elaborately described ways of executing a domain modelling process at this ambition level. For example, the modelling procedure as described in ORM [32] identifies the following steps:

- *Step 1 – Transform familiar examples into elementary facts* This step involves the verbalisation in natural language of samples take from the domain.
- *Step 2 – Draw the fact types and apply a population check* In this step, a first version of the schema is drawn. The plausibility of the schema is validated by adding a sample population to the schema.
- *Step 3 – Trim schema and note basic derivations* In this step, the schema is checked to see if any of the identified concepts are basically the same, and should essentially be combined. Furthermore, derivable concepts (e.g. $\textsf{sales-price} = \textsf{retail-price} + \textsf{mark-up}$) are identified.
- *Step 4 – Add uniqueness constraints and check the arity of fact types* At this point, it is determined how many times an instance of an identified concept can play specific roles. For example, is a *person* allowed to *own* more than one *car*?
- *Step 5 – Add mandatory role constraints and check for logical derivations* This step completes the basic set of arity constraints on the relationships in the schema, by stating wether or not instances of a concept *should* play a role. For example, for each *car*, the *year of construction* should be specified.
- *Step 6 – Add value, set-comparison, and subtyping constraints* The ORM diagramming technique provides a rich set of graphical constraints. This step is aimed at specifying these constraints.
- *Step 7 – Add other constraints, and perform final checks* Finally, there may be some constraints in the domain that cannot be expressed graphically. In this last step, these constraints can be specified and

In terms of our framework for domain modelling processes, this procedure constitutes a rather specific way of executing the *concept specification* stream of activities. It is really geared towards the (conceptual) analysis of a domain in order to design a database, rather than a general analysis of concepts playing a role in a domain. The procedure presented above is not applicable to all situations and all modellers.

Even though the above order is very explicit, and therefore well suited for educational purposes, a goal-bounded approach to domain modelling requires a more refined view. The key question concerns the *goal* for which a domain is modelled. During the *definition* phase of the software development life-cycle, when the main goal is to support requirements engineering activities, the seven steps as described above are likely to be overkill. In such a context, modellers are likely to skip steps 6 and 7. The modelling procedure as discussed in [32], also requires modellers to identify how concepts (such as car, co-workers, patient, etc.) are identified in a domain (e.g. by means of a registration number, employee number, patient number, etc.). During the definition phase, these identification mechanisms are not likely to be relevant (*yet*).

During the *design* phase of a software system most of the seven identified steps are indeed needed. However, experienced modellers are also likely to merge steps 1-3, steps 4-5, as well as steps 6-7, into three big steps. The resulting three steps will generally be executed consecutively on a ‘per fact’ base. In other words:

1. For each fact type, execute 1-3.
2. For each fact type, execute 4-5.
3. For each fact type, execute 6-7.

Some more empirical background to this, experience based observation, can be found in e.g. [99, page 161].

The order in which the various modelling tasks are performed differs to a large extent. A clear distinction exists between prescribed modelling knowledge and applied modelling knowledge, in this respect. Whereas an almost strictly *linear* order of performing modelling tasks is prescribed, a very *opportunistic* order is actually used. This order seems to be determined by at least two essentially different factors-the problem domain and the information engineer.

Note that when an initial domain model already exists, e.g. as produced in the definition phase in support of requirements engineering, this will have to be used as a starting point for completion. In other words, in practice, a domain model is likely to develop incrementally along with the software development life cycle.

ORM is not the only modelling approach that is based on analysis of natural language. However, providing a full survey of such approaches is beyond the scope of this article. Nevertheless, two approaches are worth mentioning here. In [21] the Object-Oriented Systems Analysis method is presented. It uses a natural-language based approach to produce an Object-Relationship Model (accidentally also abbreviated as ORM) that serves as a basis for further analysis. The *way of working* used is not unlike that of ORM. Its *way of modelling*, however, has a more sketchy nature and has been worked out to a lesser degree. The KISS approach, as reported in [55], also uses natural language analysis as its basis. It provides some support in terms of a *way of working*, but does this in a rather prescriptive fashion that presumes some very particular (and limited) intermediary goals. A wide spectrum of modelling concepts are introduced (*way of modelling*) covering a wide range of diagramming techniques (not unlike the UML [13]).

Independent of the approach used, a modelling process always needs to be flanked by a continuous communication process with the stakeholders [98]. Communication brings along the aspect of documentation. Modelling itself can hardly do without face-to-face discussions; however, the (intermediate) results need to be recorded in such a way that they can be communicated effectively to the stakeholder community [28, 27]. In this respect we could argue that any modelling approach also needs a *way of communication/documenting*. Since documentation serves the purpose of communication, the documentation *language* should align with the accepted language concepts in the domain. In practice it turns out that graphical notations such as ORM or UML diagrams are not the most obvious way to communicate a model to stakeholders, since most domain stakeholders do not comprehend this kind of "IT language". Often, it is better to use more intuitively readable diagrams and natural language to communicate concepts and their relationships and constraints, while occasionally, a more mathematical or algorithmic style may be useful in certain expert domains.

Part II
Enterprise Layer

Chapter 8
Viewing Enterprise Systems

Several perspectives on enterprises can be discerned. Some examples include:

Actors – The actors in a work-system, their structures and their roles.
Structure – The structure of a work-system in terms of sub-systems.
Management – How the processes in a work-system are managed.
Functions – Specific functions within a work-system such as marketing, innovation, production, etc.
Security – The way in which the security of resources (including people and information) within a work-system is warranted.

In the remainder of this textbook we will study from essentially (note: due to time constraints we will not be able to discuss all of these) seven different perspectives:

Work flow – The flow of work from in the system.
Work roles – The assignment of work to roles identified in the system, and the (de)composition of these roles.
Work objects – The objects which are acted upon by the work performed in the system.
Work distribution – The distribution of work over multiple roles in terms of services offered by one role to another, governed by transactions.
Work value – The value exchanged between roles engaged in services and transactions.
Work agents – The agents playing active roles in the system, their priorities, competencies, etc.
Work rules – The rules governing the work performed by/in the enterprise.

The notion of "perspective" as such has thus far been defined only loosely. The aim of this Chapter is to refine this notion with the better defined concept of *viewpoint*. A viewpoint will typically be positioned as a vehicle for activities like *design*, *analysis*, *obtaining commitment*, *formal decision making*, etc. We regard all of these activities to be communicative in nature. A viewpoint essentially prescribes the concepts, models, analysis techniques and visualisations that are to be used in the construction of different views on a system, where a view is typically geared towards a set of stakeholders and related stakeholder concerns. Simply put, a view is what you see, and a viewpoint is where you are looking from.

In discussing the notion of viewpoint, we will first provide a brief overview of the origin of viewpoints. This is followed by a more precise definition of viewpoints, and the concept of viewpoint frameworks. We finish this Chapter with a discussion on the requirements one may want to put on a viewpoint when using them for enterprise modelling.

8.1 The need for viewpoints

When developing or analysing a system, one will typically want to focus on specific aspects/areas of the system so as to be able to zoom in on the issues of interest at that moment. Consider the design of a new office building. In such a design we can focus on certain locations within the design. For example: the reception area or the top floor. Alternatively, we could limit ourselves to one aspect of the design only. For instance: the layout of the power-lines in a new building, or the highway infrastructure in a city plan. These are all examples of how to obtain what essentially are sub-systems of a larger whole. In the past this has lead to the metaphor of the search lights as illustrated in Figure 8.1.

The concept of viewpoint is not new. For example, in the mid eighties, Multiview [104] already introduced the notion of views. In fact, Multiview already identified five viewpoints for the development of (computerized) information systems: Human Activity System, Information Modeling, Socio-Technical System, Human-Computer Interface and the Technical System. During the same period in which Multiview was developed, the so-called CRIS Task Group of the IFIP working group 8.1 developed similar notions, where stakeholder *views* were reconciled via appropriate “representations“. Special attention was paid to disagreement about which aspect (or *perspective*) was to dominate the system design (viz. “process”, “data” or “behaviour”). The CRIS Task Group identified several *human roles* (stakeholders!) involved in information system development, such as *executive responsible*, *development coordinator*, *business analyst*, *business designer*. The results of that work can be found in “*Information System Methodologies: A framework for understanding*“ [67] and in the proceedings of the CRIS conferences from 1982-1991 [69, 68, 70, 71, 100].

The use of viewpoints is not limited to the information systems community. It was also introduced by the software engineering community. In the 1990’s, a substantial number of software engineering researchers worked on what was referred to as “the multiple perspectives problem” [24, 54, 84]. By this term, the authors referred to the problem of how to organize and guide (software) development in a setting with many actors, using diverse representation schemes, having diverse domain knowledge, and using different development strategies. A general framework has been developed in order to address the diverse issues related to this problem [24, 54]. In this framework, a viewpoint combines the notion of *actor*, *role*, or

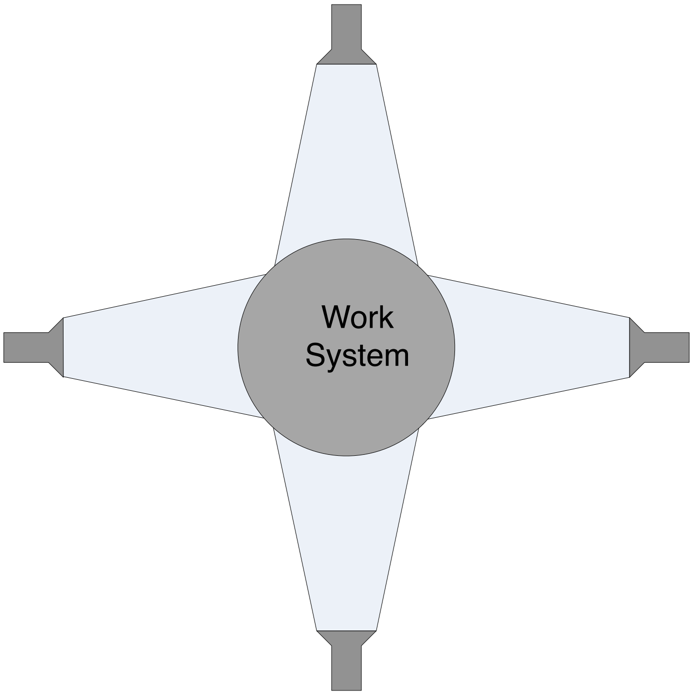


Fig. 8.1 Metaphor of the search lights

agent in the development process with the idea of a *perspective* or *view* which an actor maintains. A viewpoint is more than a *partial specification*; in addition, it contains partial knowledge of how to further *develop* that partial specification. These early ideas on viewpoint-oriented software engineering have found their way into the IEEE-1471 standard for architectural description [42] on which we have based our definition viewpoint below.

8.2 The purpose of viewpoints

In the context of work system engineering, viewpoints provide a means to focus on particular aspects of a system description. These aspects are determined by the *concerns* of the stakeholders with whom communication takes place. What should and should not be visible from a specific viewpoint is therefore entirely dependent on argumentation with respect to a stakeholder's concerns.

Viewpoints are designed for the purpose of serving as a means of communication in a conversation about certain aspects of a system. Though viewpoints can be used in strictly unidirectional, informative conversations, they can in general also be used in bi-directional classes of conversations (e.g. immediate response situations; cooperative modelling). The system engineer informs stakeholders, and stakeholders give their feedback (critique or consent) on the presented aspects. What is and what is not shown in a view depends on the scope of the viewpoint and on what is relevant to the concerns of the stakeholders. Ideally, these are the same; i.e. the viewpoint is designed with specific concerns of a stakeholder in mind. Relevance to a stakeholders concern, therefore, is the selection criterion that is used to determine which objects and relations are to appear in a view.

Below we list some examples of stakeholders and their concerns, which could typically serve as the basis for the definition/selection of viewpoints:

Middle-level management – The current situation with regard to the computerized support of a business process.

Architect – The requirements of a specific stakeholder with regard to the desired situation.

Upper-level management – The improvements which a new system may bring to a pre-existing situation in relation to the costs of acquiring the system.

End user – The potential impact of a new system on the activities of a prospective user.

System administrators – The potential impact of a new system on the work of the system administrators that are to maintain the new system.

Architect – What are the consequences for the maintainability of a system with respect to corrective, preventive and adaptive maintenance?

Upper-level management – How can we ensure our policies are followed in the development and operation of processes and systems? What is the impact of decisions (on personnel, finance, ICT, etc.)?

Operational manager – What new technologies do we need to prepare for? Is there a need to adapt maintenance processes? What is the impact of changes to existing applications? How secure are my systems?

Project manager (of system development project) – What are the relevant domains and their relations, what is the dependence of business processes on the applications to be built? What is their expected performance?

System developer – What are the modifications with respect to the current situation that need to be performed?

8.3 Defining viewpoints

In [42] the notion of a viewpoint has been defined as follows:

> *A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.*

Our definition aims to essentially be compatible with the above definition, but also aims to be more explicit. Firstly, we define a view to be:

View – A set of model descriptions of a domain from the perspective of a related set of interests of a viewer.

Views are constructed in accordance to a viewpoint, which is defined as a combination of a perspective and a viewing method:

Viewpoint – A specification of the conventions for constructing and using views. This involves: a way of thinking, a way of modelling, a way of communicating, a way of working, a way of supporting and a way of using

Viewing method – The method by means of which a view is constructed from models. A viewing method will typically make use of modelling methods for the creation of these models.

Perspective – A set of related interests in terms of which viewers may observe a domain.

A perspective may be dissected into different sub-perspectives, leading to a *perspectives framework*.

8.4 Viewpoint frameworks

In the context of work system engineering, a score of frameworks exist defining several perspectives from which to model a system. We will make a distinction between a *perspectives framework* splitting a (composed) perspective into more focused sub-perspectives, and a *viewpoint framework*. A *viewpoint framework* is a viewpoint which is decomposed into sub-viewpoints, where the decomposition is based on a perspectives framework. Note: a viewpoint framework is regarded as a composed(!) viewpoint.

Some examples of such frameworks are: The CRIS framework [67], Multiview [104], The Zachman framework [106], Kruchten's 4+1 framework [56], RM-ODP [48], ArchiMate [51] and TOGAF [96]. These frameworks have usually been constructed by their authors in attempt to cover all relevant aspects/concerns of the design/architecture of some class of systems. In practice, numerous large organisations have defined their own frameworks of viewpoints by which they describe their systems/architectures.

Some of the pre-existing frameworks are viewpoint frameworks, whole most of them are actually perspectives frameworks since they do not provide a specific method to use in creating views.

When putting the concepts of system engineering method, modelling method, model, view and viewpoint together, the meta-model as depicted in Figure 8.2 results.

We shall discuss two of these frameworks in more detail below. In the remainder of this textbook, we shall use a viewpoint framework covering four perspectives on work-systems: work-flow, work-roles, resource-roles and actors.

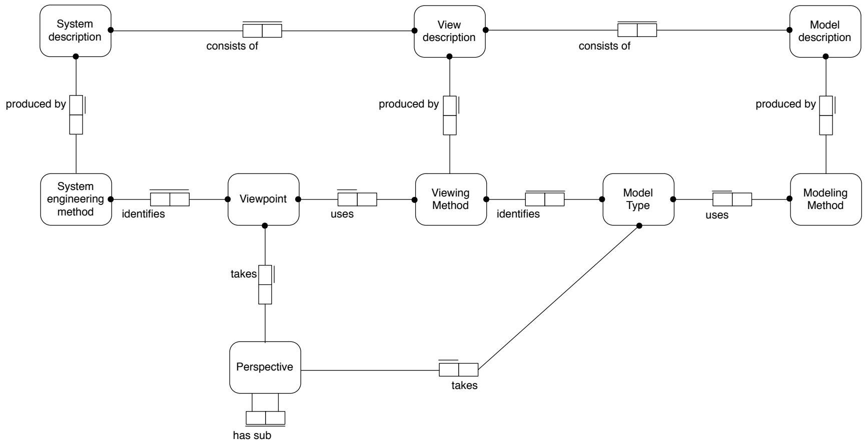


Fig. 8.2 Meta-model of core concepts introduced in this Chapter

8.4.1 The '4+1' framework

In [56], Kruchten introduces a viewpoint framework comprising five viewpoint. The use of multiple viewpoints is motivated by the observation that it "allows to address separately the concerns of the various stakeholders of the architecture: end-user, developers, systems engineers, project managers, etc., and to handle separately the functional and non-functional requirements". Kruchten does not explicitly document the motivation for these specific five viewpoints. This also applies to the version of the framework as it appears in [57, 13].

Viewpoint:	Logical	Process	Development	Physical	Scenarios
Goal:	Capture the services which the system should provide	Capture concurrency and sychronisation aspects of the design	Describe static organisation of the software and its development	Describe mapping of software onto hardware, and its distribution	Provide a driver to discover key elements in design Validation and illustration
Stakeholders:	Architect End-users	Architect System designer Integrator	Architect Developer Manager	Architect System designer	Architect End-users Developer
Concerns:	Functionality	Performance Availability Fault tolerance ...	Organisation Re-use Portability ...	Scalability Performance Availability ...	Understandability
Meta-model:	Object-classes Associations Inheritance ...	Event Message Broadcast ...	Module Subsystem Layer ...	Processor Device Bandwidth ...	Objects-classes Events Steps ...

Table 8.1 The '4+1' framework

The goals, stakeholders, concerns, and meta-model of the 4+1 framework can be presented, in brief, as shown in Table 8.1. Note: in [57, 13], the viewpoints have been re-named as follows:

- physical viewpoint → deployment viewpoint
- development viewpoint → implementation viewpoint
- scenario viewpoint → use-case viewpoint

to better match the terminology of the UML. The framework proposes modelling concepts (the meta-model) for each of the specific viewpoints. It does so, however, without explicitly discussing how these modelling concepts contribute towards the goals of the specific viewpoints. One might, for example, wonder whether object-classes, associations, etc., are the right concepts for communication with end-users about the services they require from the system. The 4+1 framework is based on experiences in practical settings by its author. This would make it even more interesting to make explicit the motivations, in terms of utility, for selecting the different modelling concepts. In [57, 13] this is also not documented. The viewpoints are merely presented 'as is'.

8.4.2 The RM-ODP framework

The Reference Model of Open Distributed Processing (RM-ODP) [48, 45, 44, 47] was produced in a joint effort by the international standard bodies ISO and ITU-T in order to develop a coordinating framework for the standardisation of open distributed processing. The resulting framework defines five viewpoints: enterprise, information, computation, engineering and technology. The modelling concepts used in each of these views are based on the object oriented paradigm. The goals, concerns, and associated meta-models of the viewpoints identified by the RM-ODP can be presented, in brief, as shown in Table 8.2.

Viewpoint:	Enterprise	Information	Computational	Engineering	Technology
Goal:	Capture purpose, scope and policies of the system	Capture semantics of information and processing performed by the system	Express distribution of the system into interacting objects	Describe design of distribution oriented aspects of the system	Describe choice of technology used in the system
Concerns:	Organisational requirements and structure	Information and processing required	Distribution of system Functional decomposition	Distribution of the system, and mechanisms and functions needed	Hardware and software choices Compliancy to other views
Meta-model:	Objects Communities Permissions Obligations Contract ...	Object classes Associations Process ...	Objects Interfaces Interaction Activities ...	Objects Channels Node Capsule Cluster ...	Not stated explicitly

Table 8.2 RM-ODP Framework

The RM-ODP provides a modelling language for each of the viewpoints identified. It furthermore states:

> *"Each language for creating views/models conform a viewpoint has sufficient expressive power to specify an ODP function, application or policy from the corresponding viewpoint."*

Again there is no detailed discussion regarding the utility of the concepts underlying each of these languages, from the perspective of the goals/concerns that are addressed by each of its viewpoints. Also, the RM-ODP does not explicitly associate viewpoints to a specific class of stakeholders. This is left implicit in the concerns which the viewpoints aim to address. In particular in the case of an international standard, it would have been interesting to see explicit motivations, in terms of utility to the different goals, for the modelling concepts selected in each of the views.

8.5 Topical focus of models

Several dimensions exist which can be employed in spanning viewpoint frameworks. Below we discuss possible dimensions to span such frameworks. The list op dimensions are based on work reported in [94, 26, 78, 50, 35, 30, 79, 59, 82].

Nature of the information – This refers to the nature of the content. Based on [30] we identify the following sub-classifiers:

Policy – Policy statements pertaining to the system.
Principles – Principles to which the (design of) the system should adhere.
Guidelines – Guidelines (operationalisations of the principles) which should be met by the future system.
Descriptions – Descriptions of what the (future) system (will) look(s) like.
Standards – Standards (to be) used in the creation/design of the system.

Type of information – This refers to the kind of information that may be contained in the model. Inspired by e.g. [94, 59], typical sub-classifiers would be:

Business – Business models, markets, products, etc.
Organisation – Work processes, culture, organisational structures, skills, etc.
Information – Domains of information/knowledge needed for the business activities.
Application – Automation of work.
Infrastructure – Underlying technological infrastructure.

Most of the "architecture frameworks" provide different sub-classifications for these classes of information.

Development chain – The focus with regard to the stages in a system's development. We shall use the following sub-classifiers:

System definition – A model representing aspects of a system's definition.
System design – A model representing aspects of a system's definition.

Constructed system – A model representing aspects of a constructed system.
Operational system – A model representing aspects of an operational system.

Systemic scope – The scope of the domain that is modeled by the model. We identify three sub-classifiers:

Use-case – The scope of a specific use-case of the work system, i.e. how the using system will use the subject system.
System component – A distinct component of a work system.
System – The entire work system and its direct environment is the domain that is to be modeled.
System of Systems – The scope is a set of work systems, in other words, a system of work systems.

Temporal scope – The temporal scope at which we regard domain. Sub-classifiers are:

Operational – The work system as it is currently (or in the near future) operational.
Tactical – The work system as it is ideally operational after the execution of a development projects.
Strategical – The work system as it is ideally operational after the execution of a number (a program) of development projects.

Implementation abstraction – The level of abstraction from underlying information technology. Based on the distinction from MDA, we define the following sub-classifiers:

Computing independent – A model which is independent of decisions regarding computerisation of certain activities.
Platform independent model – A model in which choices for computerisation of tasks has indeed been made, but does not yet make a choice for a specific technological platform.
Platform specific – A model which is tied to a specific technological platform. A program in a programming language such as C or Java would be an ultimate example of a platform specific model.

Systemic aggregation – The level of aggregation that is used in a model. Systemic aggregation is usually used as a means to "hide" information about specificities of a work system. It allows modelers (and viewers of the models) to focus on higher level issues. Based on [59] we identify the following sub-classifiers:

Detailed level – All (relevant) details are shown.
Coherence level – The model focuses on the coherence (between different aspects) of the modeled domain.
Overview level – The model provides an overview of the key issues of the modeled domain.

System qualities – Models may focus on different *qualities* of systems. In [49, 46] the following key qualities have been identified:

Efficiency – Is the relationship between the level of performance of the system (or a sub-system) and the amount of resources used, under stated conditions. Efficiency may be related to time behaviour (response time, processing time, throughput rates) and to resource behaviour (amount of resources used and duration of such use).

Functionality – Bears on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

Sub-characteristics of functionality are: suitability, accuracy, interoperability, compliance, security and traceability.

Reliability – Is the capability ofthe system (or a sub-system) to maintain its level of performance under stated conditions for a stated period of time. The mean time to failure metric is often used to assess reliability of systems. The reliability can be determined through defining the level of protection against failures and the necessary measures for recovery from failures.

Sub-characteristics of reliability are: maturity, fault tolerance, recoverability, availability and degradability.

Maintainability – Bears on the effort needed to make specified modifications to the system (or a sub-system). Modification may include corrections, improvements or adaptation to changes in the environment, the requirements and the (higher levels of the) design.

Sub-characteristics of maintainability are: analysability, changeability, stability, testability, manageability, reusability.

Portability – Is the ability of the system (or one of its components) to be transferred from one environment to another.

Sub-characteristics of portability are: adaptability, installability, conformance, replaceability.

Usability – Bears on the effort needed for the use of the system (or one of its sub-systems) by the actors in the environment of the system.

Sub-characteristics of usability are: understandability, learnability, operability, explicitness, customisability, attractivity, clarity, helpfulness and user-friendliness.

System realisation – The level of realisation of the services provided by a system to its environment. The underlying way of thinking is that a work system is a *supporting system*, which provides functionality to a *using system*, while making use of *infrastructure system(s)*. Sub-classifiers are therefore:

Using system – The way the environment will use the work system.

Supporting system – The way the work system will provide functionality and/or services to the environment.

Infrastructure system – The infrastructural facilities that are used by the work system in order to provide the functionality/services to the environment.

Actor kinds – What kind of actors *in the system* does the modelling focus on? Subclassifiers are:

- **Heterogeneous** – Heterogeneous (typically composed) actors.
- **Human** – The role of human beings in a work system.
- **Computerised intelligence** – The role of software agents/components that aim to show intelligent behavior.
- **Computerised** – The role of "traditional" software components.

Interrogatives – The English language, as well as most other languages, contains a class of words calledthe *interrogatives* [95]: *Which, when, how, what, why, where, whose, . . .* These words may be used to formulate questions concerning situations, people, or any other phenomenon we may perceive or conceive. In other words, we may use these interrogatives to identify different relevant aspects of a system. By using questions based on the interrogative words, insight may be gained into different aspects of a system, such as: *Actors, timing, processes, functionality, rationale, purpose, locality, structure, ownership, . . .* The so-called Zachman framework, [106, 93], is an example of a framework for information systems that is based on the *what*, *how*, *where*, *when*, *who*, *why* interrogatives, leading in their interpretation to information system aspects:

1. Data
2. Function
3. Network
4. People
5. Time
6. Motivation

It actually combines these aspects with five classes of audiences:

1. Time
2. Motivation
3. Planner
4. Owner
5. Designer
6. Builder
7. Contractor

{*We should really also discuss our own framework here as well. In other words, the different perspectives: work-value, work-services, etcetera, and relate that to a framework based on value/function/construction and an ontological/implementation level.*}

8.6 Questions

1. What is the difference between: a model, view and viewpoint?

2. Why is it important to carefully select relevant viewpoints when developing systems?
3. What are possible dimensions for refinements of system descriptions?
4. Why is it important to acknowledge the fact that different stakeholders will have different views on a pre-existing or a future system?
5. Identify, for the modelling languages you have see so-far as part of your studies:

    - The main modelling constructs of these languages.
    - Typical interests and viewers for which these languages may be useful.

Chapter 9
Work-Flow Modelling

This Chapter is concerned with a viewpoint for thework-flow perspective on work-systems. The focus of this viewpoint is on the questions:

1. What work is done? What activities are there?
2. What is the structuring of activities in sub-activities?
3. In which order can/should the work be done?
4. What is the expected throughput of a specific flow of activities?

Before presenting our viewpoint, which should be regarded as an example viewpoint and not as the ultimate answer, however, we request readers to first define their own viewpoint aimed at meeting these questions.

We start with an exercise:

1. Define a viewpoint suitable for answering the above questions.
2. Apply this viewpoint to the running case provided in Appendix A. Try to answer the questions posed above.
3. Have someone else use your viewpoint in creating models for the same domain. Do they produce the same model?

9.1 Way of thinking

When modelling a work-flow, one essentially takes the perspective that there is a *flow of work* in a work-system. In other words, some string of activities starting somewhere and then meandering through the work-system, possibly branching of in multiple directions and sometimes joining again, like a river flows through the land.

Just as a river system occupies a river basin, a work system can be regarded as occupying a “work basin”. Discerning specific flows of work can best be likened to tracing the flow of a piece of wood floating through a river basin. The behaviour of this piece of wood in the river basin, however, also depends on the flow of other objects in a river basin. Similarly, the flow of some specific piece of work also depends on other pieces of work flowing through the work basin.

When taking some work basin as our scope, the flow of work through this basin must enter the basin somewhere. This is presumed to be some event which is external to the scope of our basin. Ice melting through the warmth of the sun is an external trigger to the drops of water entering a river bed. There are also point where the flow will leave the work basin, just as water will leave a river basin when it enters the ocean through one of the arms of a river delta.

A specific flow of work in a work basin is referred to as a work case. Such a work case is triggered by some event which is external to the work basin under consideration. The flow of work in a work system is referred to as a work process. This leads to the following definitions:

Work case – A work process taking place in the work-system under consideration, which is triggered by an external trigger.
"*Paying the first month of Mr. Jones' pension.*"

Work case type – A class of similar work cases.
"*Paying the first month of people's pension.*"

Work process – A flow of work taking place in a work-system.

Work process type – A class of similar processes.

External trigger – An event taking place in the environment of the work-system under consideration.
"*Mr. Jones from Bristol turns 65 years of age.*"

External trigger type – A class of external triggers with similar treatment.
"*People turning 65 years of age.*"

The notion of use case as defined by UML is a specialisation of the notion of work case. In UML [13] a use case is defined as:

> *A description of a set of sequences of actions, including variants, that a system performs that yields an observable result of value to a particular actor.*

which can be read as:

> *A work case that yields an observable result of value to a particular actor.*

9.2 Way of modelling

9.2.1 Basic modelling language

Our proposed way of modelling is primarily inspired by ArchiMate [59]. However, as ArchiMate (which aims at an architectural level) does not provide enough detail about process triggering, we have extended the triggering notation with the symbols used by YAWL [2]. The key symbols of the resulting notation are depicted Figure 9.1. Symbols allowing for complex triggering types are shown in Figure 9.2.

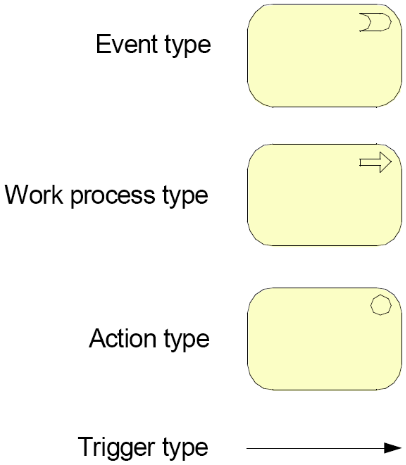


Fig. 9.1 Notation for describing work-flows

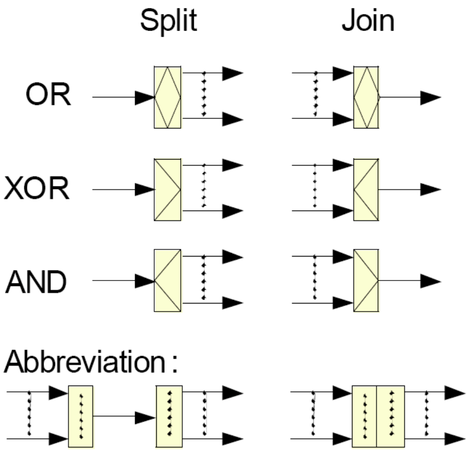


Fig. 9.2 Complex triggering types

When modelling work-flows in a system, one will typically produce dedicated model (view!) descriptions for each work-case discerned. An example is shown in Figure 9.3. A fragment of the corresponding ORM model (with enriched notation) is shown in Figure 9.4.

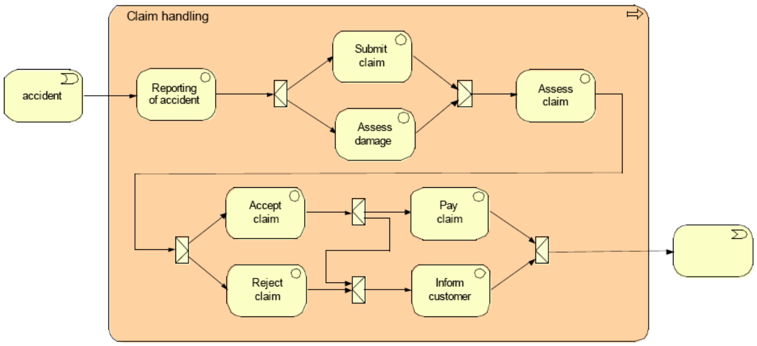


Fig. 9.3 Damage claim handling

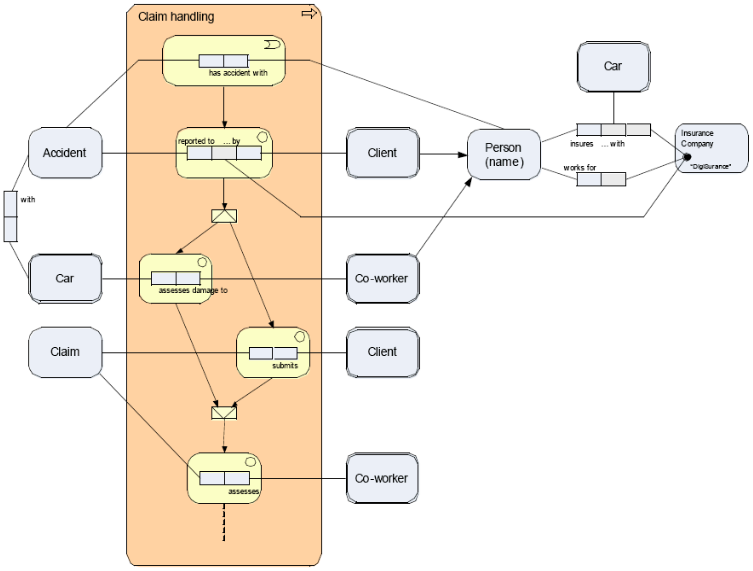


Fig. 9.4 Damage claim handling ORM model

9.2.2 *Petri-net based semantics*

In order to gain a more precise understanding of work-flow models, we will discuss their mapping to simple Petri-nets. In doing so we will, however, loose out on some semantic details. The exercise we undertake in this Section is purely for educational purposes. See e.g. YAWL [2] for a more detailed formalisation.

In Figure 9.5 the working of Petri-nets are illustrated. In Figure 9.6, the mapping from complex triggering in work-flow models to Petri-nets is given. Note that the

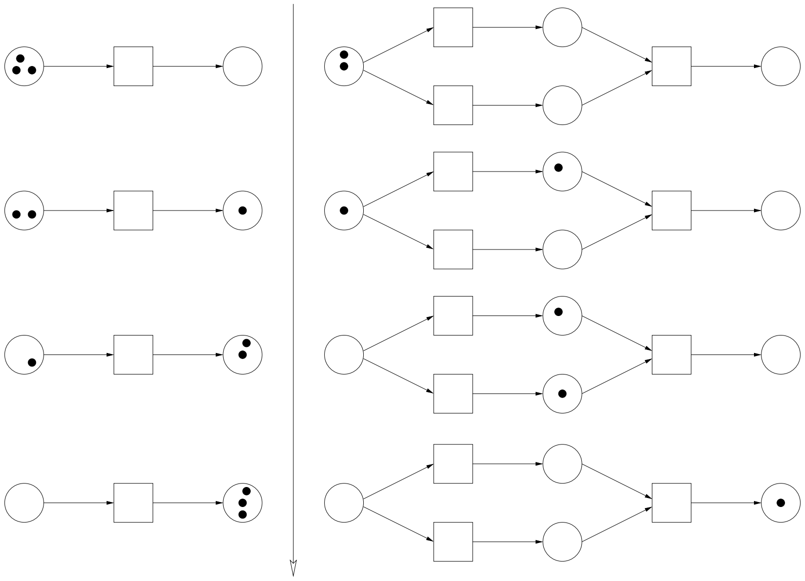


Fig. 9.5 Example of Petri-net executions

resulting Petri-net will only be applicable for one run of the process. An example mapping of a work-flow to Petri-nets is given in Figure 9.7.

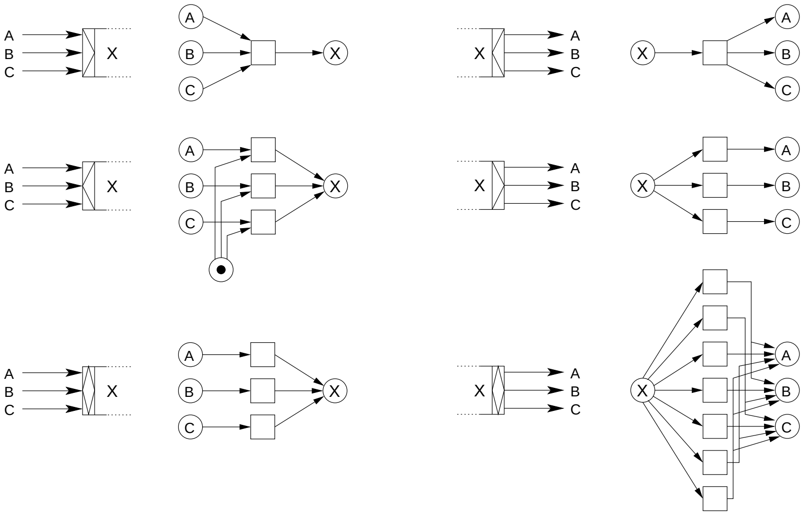


Fig. 9.6 Semantics of work-flows

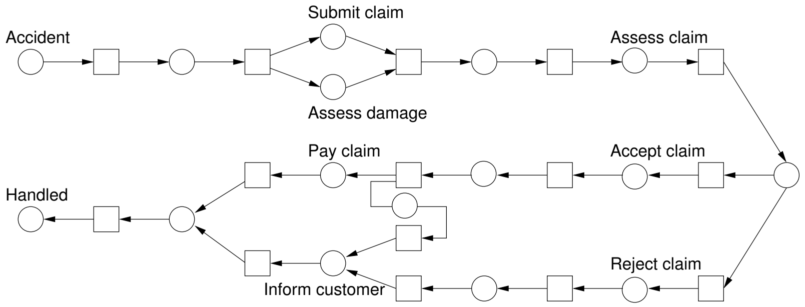


Fig. 9.7 Example mapping of a work-flow to a Petri-net

9.2.3 *Composition of work and actions*

Work-processes may be regarded as being composed of yet other processes. This is illustrated in Figure 9.8 where the claim handling process has been dissected into three sub-processes. Introducing the introduction of sub-processes allows for a grouping of activities into semantically meaningful units. In the example: Pre-phase, Handling and Post-phase identify phases in the claim handling process.

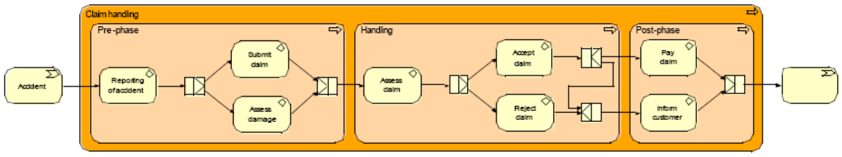


Fig. 9.8 Sub-processes in claim handling

We might also want to regard a sub-process as a single composed action. This is illustrated in Figure 9.9. In the top diagram, the action type Prepare is regarded as a black box, while in the bottom diagram the action type has been unfolded to show its internal details (white box). The sub-processes Handling and Post-phase cannot be treated as composed action types. A composed action type can only have a single incoming and outgoing trigger.

Also consider the example shown in Figure 9.12.

9.2.4 *Quantative semantics*

The Petri-net based semantics of work-flows focuses on the flow of specific work cases. In addition to the Petri-net based semantics, we identify a quantitative seman-

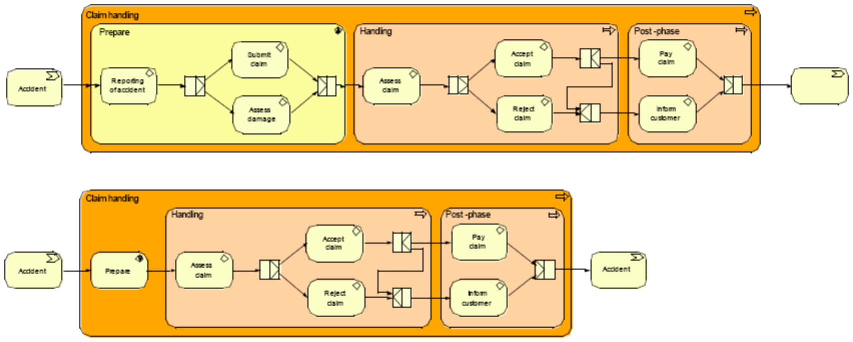


Fig. 9.9 Decomposition of actions

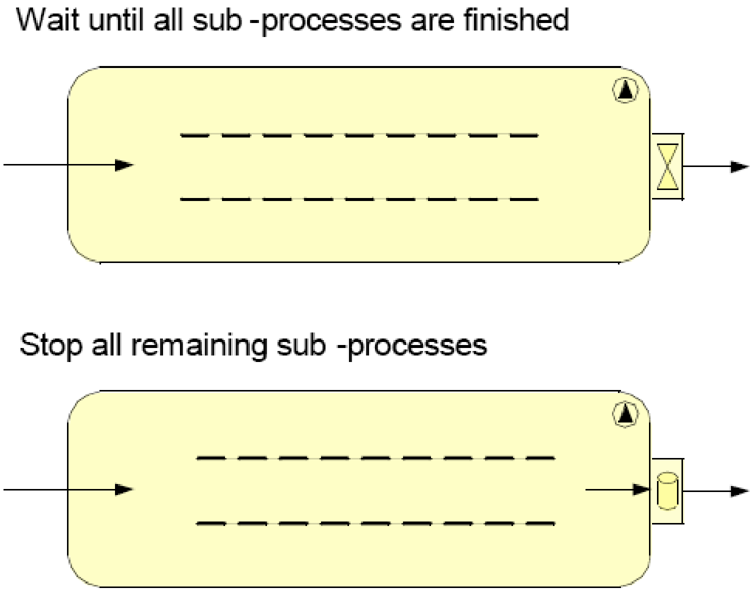

Wait until all sub -processes are finished
Stop all remaining sub -processes

Fig. 9.10 Termination of composed processes

tics. This is illustrated in Figure 9.13. The earlier used claim handling example can be made quantitative as depicted in Figure 9.14.

Some possible uses for a quantitive analysis are:

- Performance analysis.
- Optimisation of design.

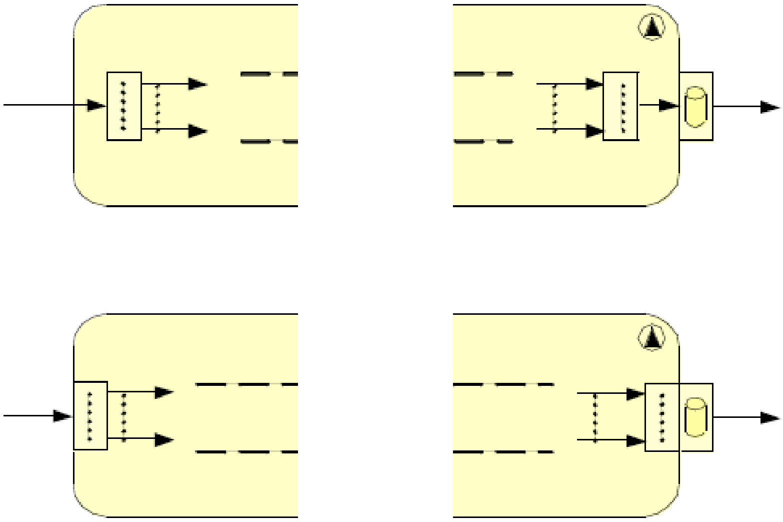


Fig. 9.11 Abbreviations

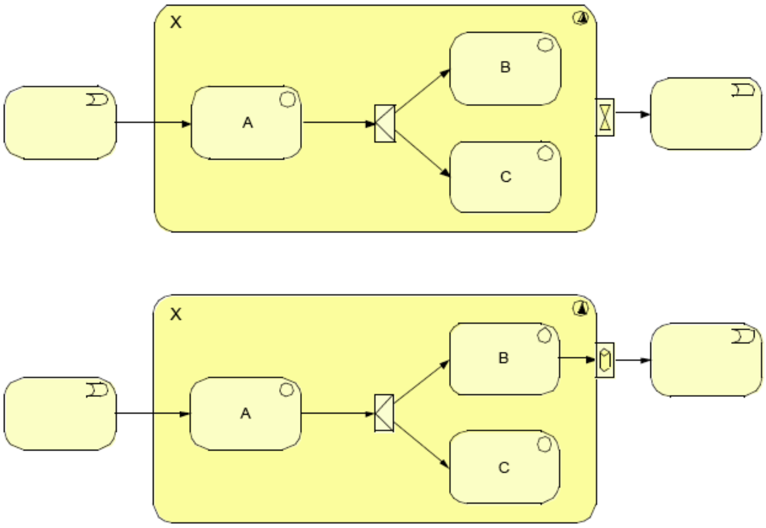


Fig. 9.12 Termination of composed action types

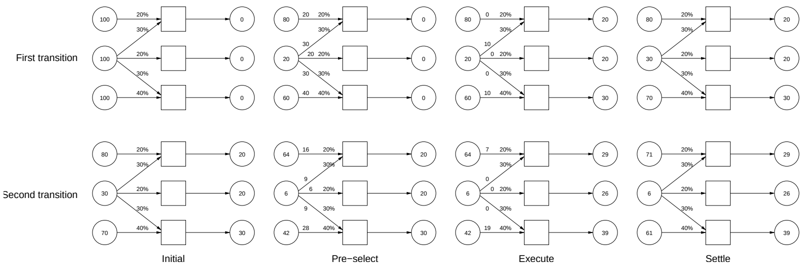


Fig. 9.13 Quantitative analysis of activities

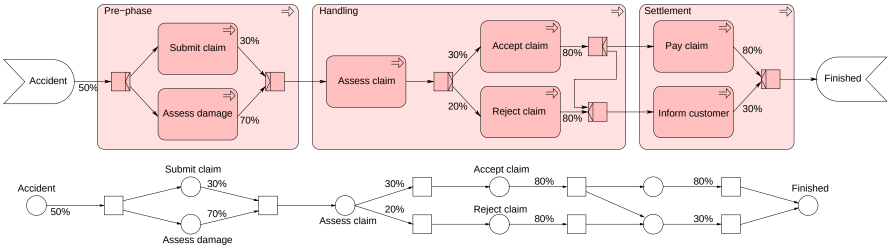


Fig. 9.14 Claim handling with quantitative information

9.2.5 *Mapping to UML 2.0 activity diagrams*

In this Section we look at the relationship between the work-flow modelling using the (enriched) ArchiMate notation and the activity diagrams from UML 2.0 [1]. UML activity diagrams focus on the flow of activities. Involvement of actors is represented by means of swimming lanes. The claim handling example from Figure 9.3 can be represented by means of UML activity diagrams as illustrated in Figure 9.15.

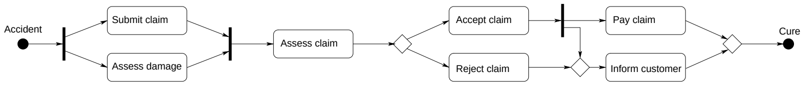


Fig. 9.15 Claim handling as an UML 2.0 activity diagram

9.3 Way of Working

Outline

1. Define for each of the key entity types, relevant work case

2. Identify key work cases
3. Verbalise each work case
4. Create an ORM model of the verbalisations
5. Determine temporal dependencies
6. Single-out activities
7. Detail model
8. Re-examine models
9. Identify process sub-phases

9.4 Questions

1. Finish Figure 9.4.
2. Pop-groepen (’bands’) verschijnen en verdwijnen. Ze worden ooit door een of meer personen opgericht en heffen zich ooit een keer op. In de tussentijd, dus gedurende hun bestaansperiode, kunnen mensen tot zo’n band toetreden of de band verlaten. Zoals bekend spelen bands (muziek)nummers (’songs’) en nemen een vaker gespeelde song meestal ook op (voor uitgave op CD etc.). Elke song is ooit door een of meer personen gecomponeerd en kan daarna door verschillende bands (’life’) gespeeld en/of opgenomen worden (zo is de song ’Dreaming of a white Christmas..’ in het verleden door heel wat bands op hun eigen wijze gespeeld..). Elke aparte opname van een song door een band wordt op een bepaalde datum en in een bepaalde studio opgenomen.Analyseer nu de hierboven beschreven situatie en produceer hier een work-flow model voor.
3. Consider the claim handling process with quantative information as depicted in Figure 9.14. Suppose 100 accidents are reported. Compute the number of iterations needed to finish at least one claim. When taking percentages of a number of tokens at a specific place, round-off downward.

/

Chapter 10
Work-Role Modelling

This Chapter is concerned with a viewpoint for the work-role perspective on work-systems. The focus of this viewpoint is on the actors that perform work in a work-system. We are specifically interested in questions such as:

1. What roles can be identified with regards to the work done in a work-system?
2. What are the required competencies of the actors performing these roles?
3. How are they structured? Relevant aggregations? Sub-roles?

Before presenting our viewpoint, which should be regarded as an example viewpoint and not as the ultimate answer, however, we request readers to first define their own viewpoint aimed at meeting these questions.

We start with an exercise:

1. Define a viewpoint suitable for answering the above questions.
2. Apply this viewpoint to the running case provided in Appendix A. Try to answer the questions posed above.
3. Have someone else use your viewpoint in creating models for the same do main. Do they produce the same model?

10.1 Way of thinking

Our work-role viewpoint really consists of three model types with regards to work-roles:

Role-structure models – In these models we are concerned with the composition of work-roles into more complex roles, as well as collaboration between roles, and generalisation/specialisation of roles.
Role-interaction models – These models focus on interactions between roles.
Role-involvement models – These models bridge the gap between work-roles and the work-flow models as discussed in the previous Chapter.

Collaborations between roles typically have a temporary, an informal nature, and no clear identity by means of which it is referred to persistently. They are therefore treated differently from aggregated roles. In the case of an aggregate work-role, objects playing the aggregate work-role are required to consist of a number of objects playing the aggregated work-roles.

10.2 Way of modelling

Our proposed way of modelling is, again, primarily inspired by ArchiMate [59].

10.2.1 Role-structure

The notation for the role-structure sub-viewpoint is provided in Figure 10.1.

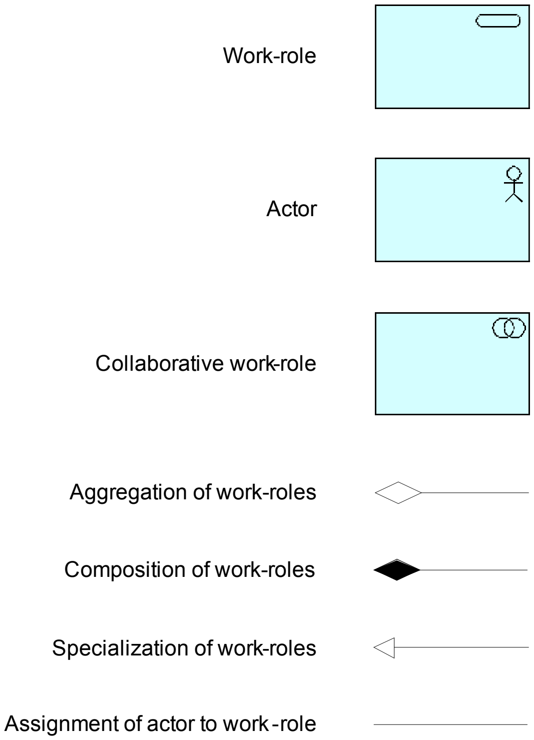

Work-role
Actor
Collaborative work-role
Aggregation of work-roles
Composition of work-roles
Specialization of work-roles
Assignment of actor to work-role

Fig. 10.1 Notation for role structures

An allowed abbreviation is shown in Figure 10.2.

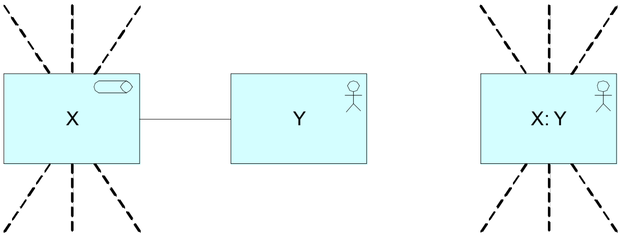


Fig. 10.2 Abbreviation for role structures

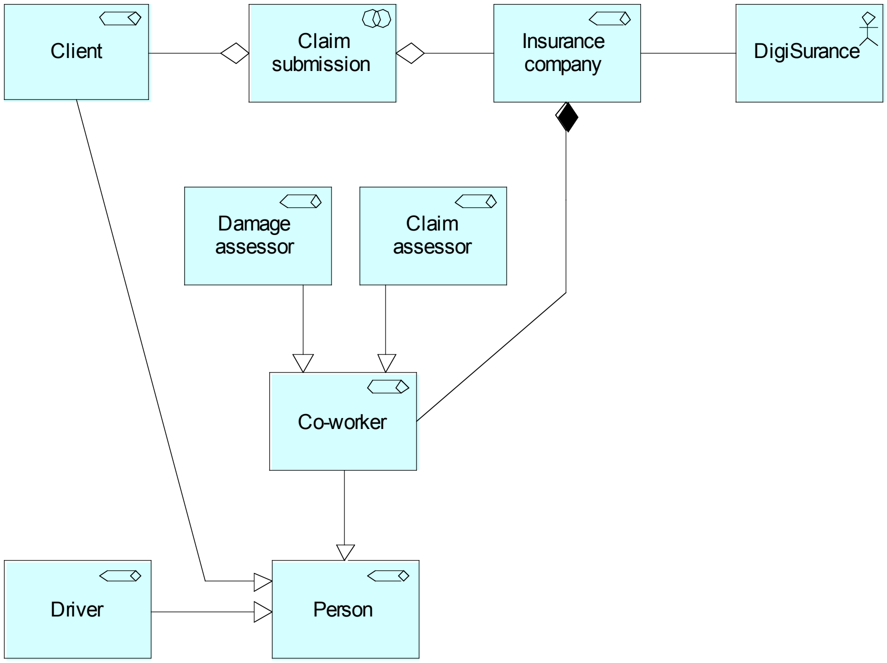


Fig. 10.3 Work-roles in damage claim handling

An example of the use of this notation is given in Figure 10.3. This example model focuses on the work-roles involved in the claim handling process. An alternative version is depicted in Figure 10.4 where we have used a more compact representation of the composition relationship between roles.

The embedding of the example from Figure 10.3in the underlying ORM model is provided in Figure 10.5. If in the context of a process, a role is regarded as being a

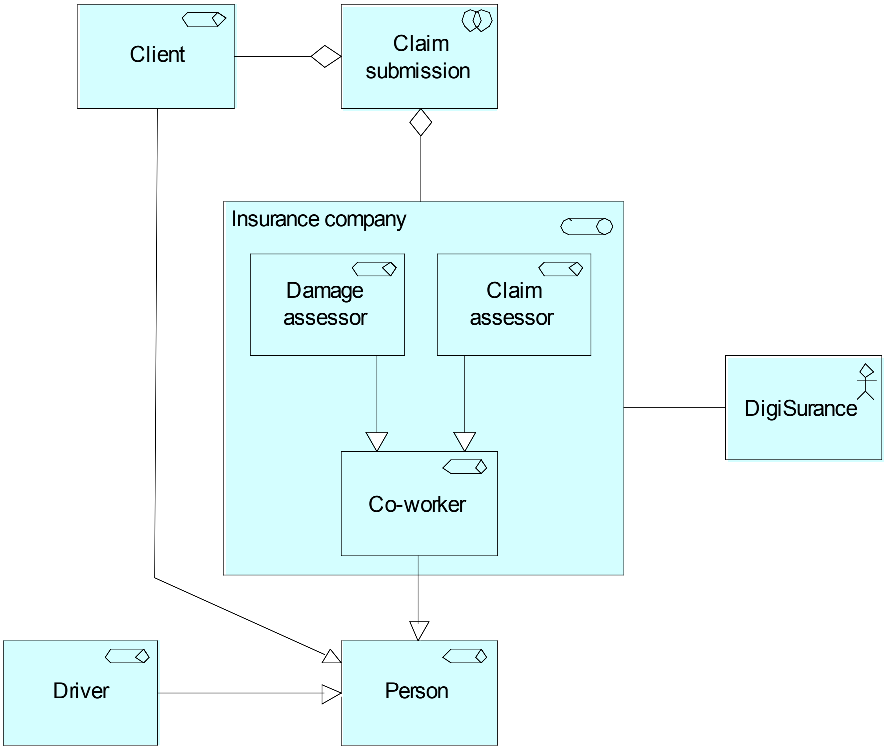


Fig. 10.4 Alternative for composition

work-role then the *definition role* of the associated object type is treated as a work-role as well. This is illustrated below in the case of client, co-worker and person. We have also added the (implied!) subset constraints on work-roles.

10.2.2 Role-interaction

To be able to explicitly model the interaction between roles, we first need to introduce the notion of a transaction between (the players of) work-roles. An example is shown in Figure 10.6.

A transaction takes place between a number of roles, of which one is the initiator and the other is the executor. This is signified using arrows the work-role with the outgoing arrow is the initiator, and the one with the incoming arrow the executor. In the example, all transactions are presumed to be initiated by the insurant. In line with the DEMO [85, 17] approach, we presume transactions to involve two kinds of activities: action and interaction. This is illustrated in Figure 10.7.

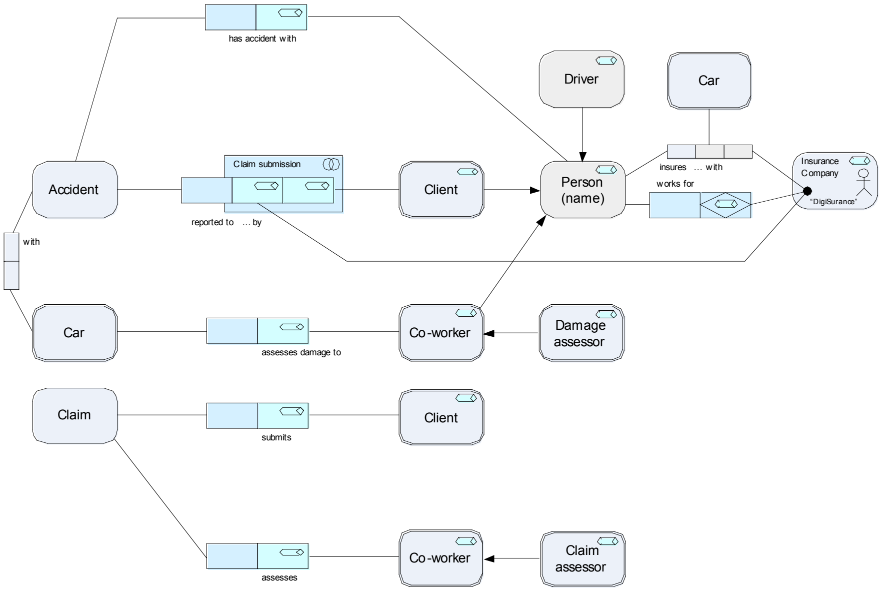


Fig. 10.5 Work-roles in damage claim handling ORM model

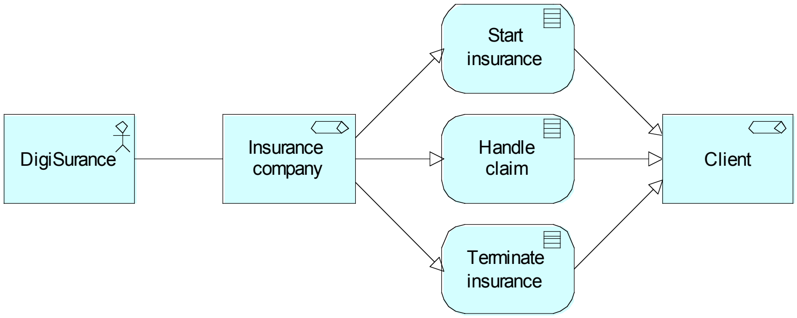


Fig. 10.6 Transactions between roles

The *action* refer to the production actions performed by the executor of a transaction, while the *interaction* is concerned with the coordination efforts surrounding the production process. The latter typically involves four major acts:

R-act – The player of the initiating role makes a *request* (pertaining to some desired change in the universe).

P-act – The player of the other role involved in the transaction *promises* to fulfill the request.

S-act – The player of the other role involved in the transaction *states* that they have fulfilled the request.

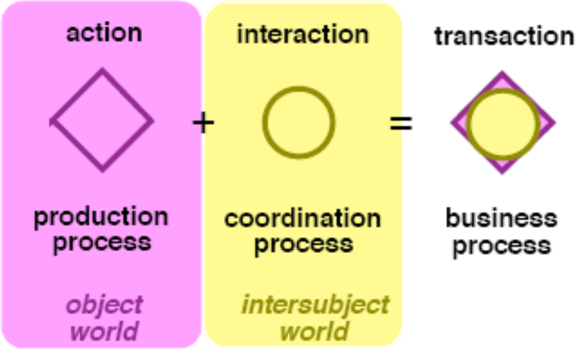


Fig. 10.7 Transaction = action + interaction

A-act – The initiator of the transaction *accepts* the result if the desired change in the universe has indeed come about.

A transaction can now be in three phases:

O-phase – While the players of the work-roles are involved in the **R**-act and **P**-act of a transaction, the transaction is in the *order* phase of the transaction.
E-phase – Once in the **P**-state, the transaction is in the *execution*-phase.
R-phase – When finally moving from the **S**-state to the **A**-state, the transaction is in its *result*-phase.

Another example, using the original DEMO notation, is shown in Figure 10.8.

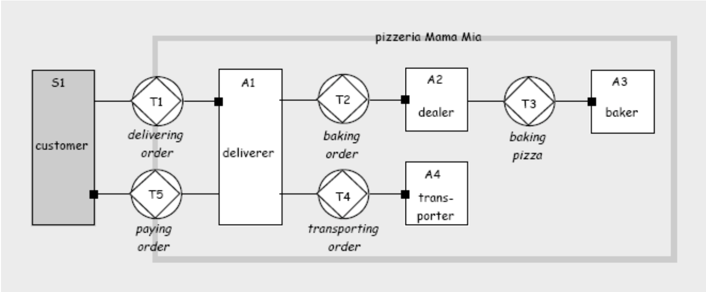


Fig. 10.8 Example transactions in DEMO

The **R**, **P**, **S** and **A** states of a transaction really only are common states one expects to move through. In reality, however, several break downs may occur during a transaction. This is illustrated in Figure 10.9.

One may want to explicitly model an executor's willingness/ability to execute the actions involved in a transaction. This essentially constitutes a kind of "pre-promise act". This leads to the introduction of the notion of a *service*:

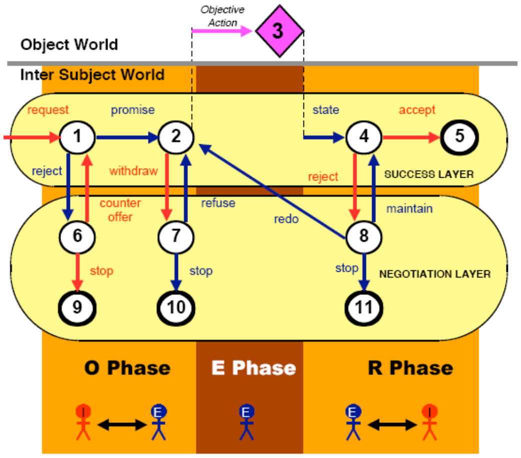


Fig. 10.9 Additional states of a transaction

Service (type) – A unit of work which achieves a pre-defined end result

The execution phase of a transaction corresponds to the delivery of a service. In the case of Figure 10.6 one may want to explicitly state that an insurance company offers the services of claim handling, taking out an insurance, and terminating one. This would lead to the situation as depicted in Figure 10.10.

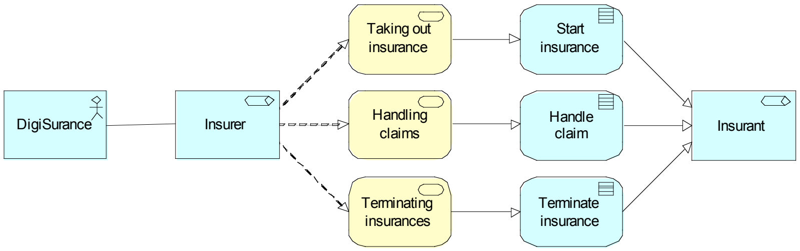


Fig. 10.10 Making services explicit

When making the services offered by players of a work-role explicit, one would need to clearly define the unit of work that will be provided when asked to perform the service. However, there is more to services than their "functional" behavior. Services may be delivered at different levels of *quality*. Basing ourselves on [49, 46], we define quality to be:

Quality – Is the totality of systemic properties of a system that relate to its ability to satisfy stated and/or implied needs.

The qualities we referred to, are really an abbreviation of quality properties, which are a special class of systemic properties:

Quality property – A systemic property, used to describe and asses the quality of a system.

These quality properties can be used to judge a service offered by a system, or express a desired quality level. In [49, 46] a range of key classes of quality properties are identified. These classes are referred to as quality attributes:

Quality attribute – A specific class of quality properties.

Each quality attribute may also have a number of sub-characteristics, zooming in on specific quality characteristics. In [49, 46] the following quality attributes have been identified:

Efficiency – Is the relationship between the level of performance of the system (or a sub-system) and the amount of resources used, under stated conditions. Efficiency may be related to time behavior (response time, processing time, throughput rates) and to resource behavior (amount of resources used and duration of such use).

Functionality – Bears on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs. Sub-characteristics of functionality are: suitability, accuracy, interoperability, compliance, security and traceability.

Reliability – Is the capability of the system (or a sub-system) to maintain its level of performance under stated conditions for a stated period of time. The mean time to failure metric is often used to assess reliability of systems. The reliability can be determined through defining the level of protection against failures and the necessary measures for recovery from failures. Sub-characteristics of reliability are: maturity, fault tolerance, recoverability, availability and degradability.

Maintainability – Bears on the effort needed to make specified modifications to the system (or a sub-system). Modification may include corrections, improvements or adaptation to changes in the environment, the requirements and the (higher levels of the) design. Sub-characteristics of maintainability are: analyzability, changeability, stability, testability, manageability, reusability.

Portability – Is the ability of the system (or one of its components) to be transferred from one environment to another. Sub-characteristics of portability are: adaptability, installability, conformance, replaceability.

Usability – Bears on the effort needed for the use of the system (or one of its subsystems) by the actors in the environment of the system. Sub-characteristics of usability are: understandability, learnability, operability, explicitness, customizability, attractivity, clarity, helpfulness and user-friendliness.

From the consumer of a service offered by a system, mainly the *functionality*, *reliability* and *usability* attributes will be of interest.

The repeated consumption of a service from one actor by another actor may be governed by a (formal or informal) contract. Such a contract defines the actual service (i.e. its functional behavior), as well as the non-functional quality properties that it should exhibit. For example: *after lodging an insurance claim, the client will* *be informed of the outcome within 10 workdays*. Such contracts make explicit the mutual commitments/responsibilities of the actors involved in a service delivery.

10.3 Role-involvement

The first type of role-involvement models considers roles and transactions to be black boxes. Consider the top most model depicted in Figure 10.11. John orders a Pizza from Pizzeria Mama Mia. In doing so he takes on the role of a pizza consumer, and *orders* a pizza. In the top most model we only see the fact that Pizzeria Mama Mia is able to take on the role of a Pizza Provider to indeed deliver a Pizza to John. This example is adapted from the DEMO documentation.

The second model in Figure 10.11 provides us with more details. The black box is of role transaction Provide Pizza is opened up. Providing pizza's really involves two sub-transactions: Deliver order and Pay order.

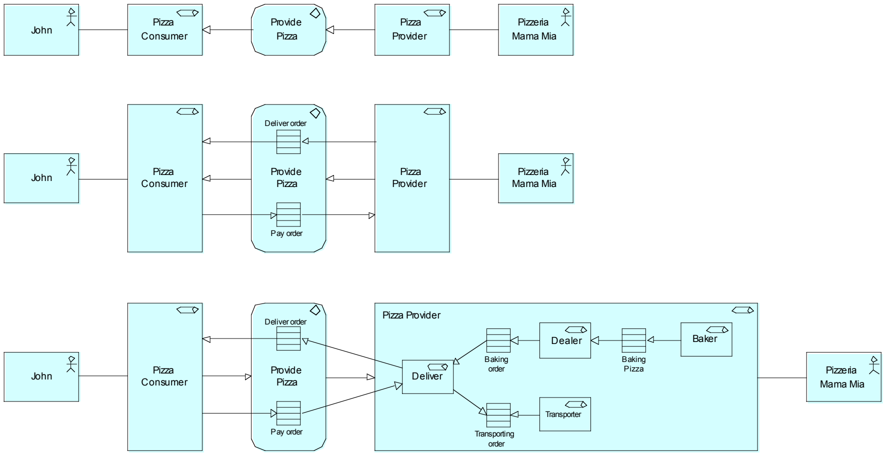


Fig. 10.11 From blackbox to whitebox

The next step, leading to the bottom model in Figure 10.11 allows us to peek inside the Pizza provider role. Within this role, additional roles are identified (making the Pizza provider into an aggregated role).

When regarding a system as a black box, weobtain *information hiding* with regards to the execution of transactions, i.e. service delivery. In other words, we do not see the way a particular service delivery is actually implemented. A.P. Barros has defined service information hiding in his PhD [8] as:

> *A technique (for business modelling) should allow the formulation of service requests to be independent of their actual processing. In other words, we should be able to talk about service independent of the (business) processes that implement them.*

Service information hiding allows us to provide one description of a service, while allowing for many different ways of implementation and/or delivery. This mechanism is useful for the development of software, but is equally useful for the design of organisations, or rather any work-system that is purposely designed.

A service delivery can be regarded as moving through a number of sub-transactions between the provider and consumer of the service. As such, a service delivery (execution of a transaction) can be modeled as any other activity. However, in this case the system providing the service is treated as a black-box.

The models in Figure 10.11 still provide no insight into the execution order of the transactions. Two possible orders are illustrated in Figure 10.12. Many more are possible.

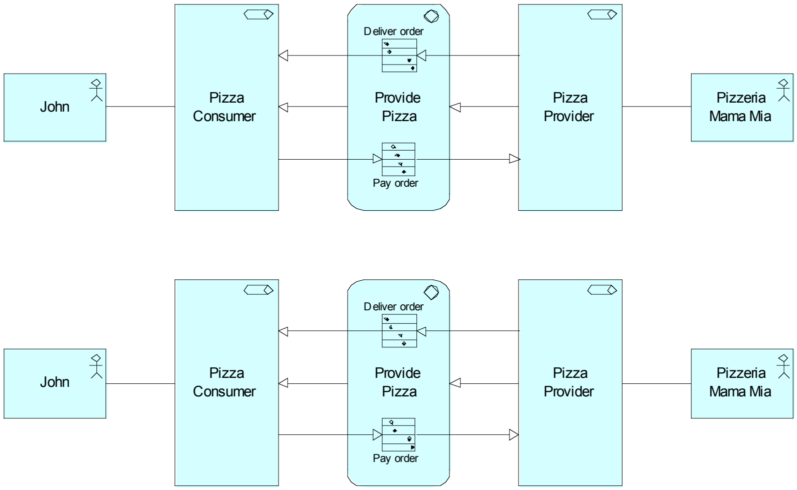


Fig. 10.12 Different execution orders of nested transactions

When a necessity exists to explicitly enforce the order in which transactions take place, one can do so using a diagram as shown in Figure 10.12. The temporal dependencies between the steps in the transactions could actually involve complex triggering, such as shown in Figure 9.1. The Deliver Pizza work process type could be decomposed further using a work-flow model. However, the different acts of the two comprising sub-transactions can be decomposed further as well into yet other sub-transactions and/or activities.

In modelling work-flows such as the Deliver Pizza work process type, we would also like to see which activities are performed by which roles. This will lead to the swimming lane models as exemplified in Figure 10.14.

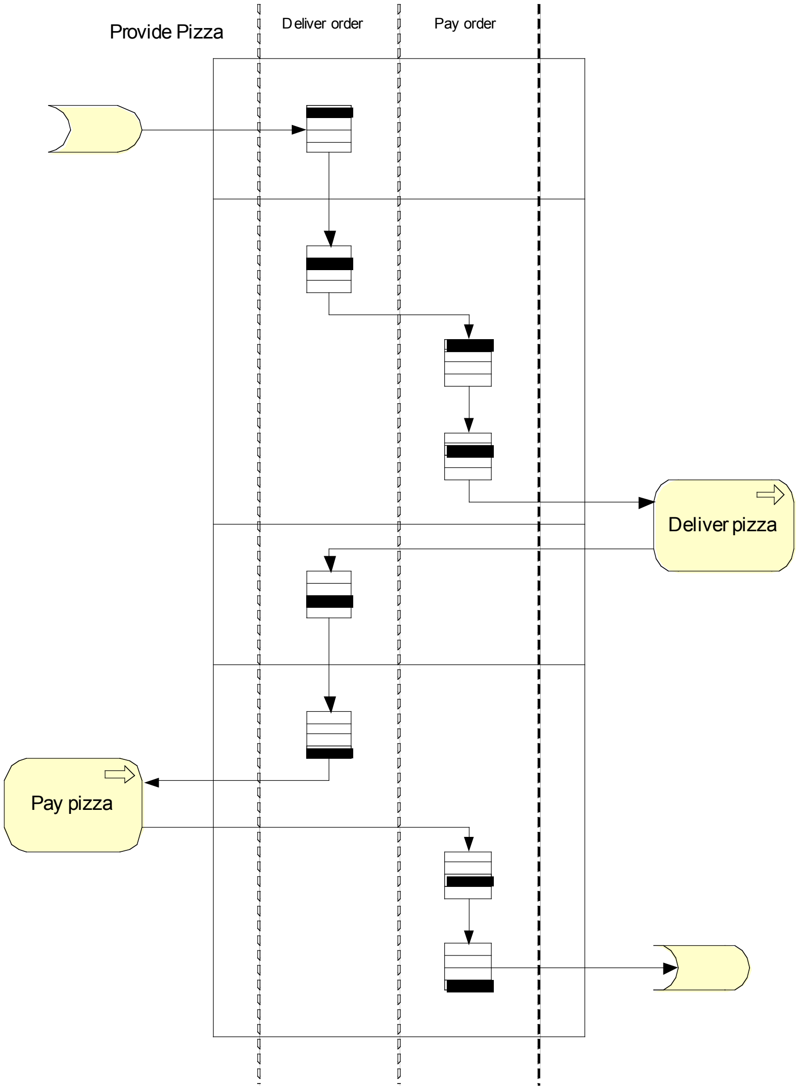


Fig. 10.13 Specifying the (default) execution order of sub-transactions

10.4 Questions

1. How do you represent that in a county, you must be insured if you're a driver?

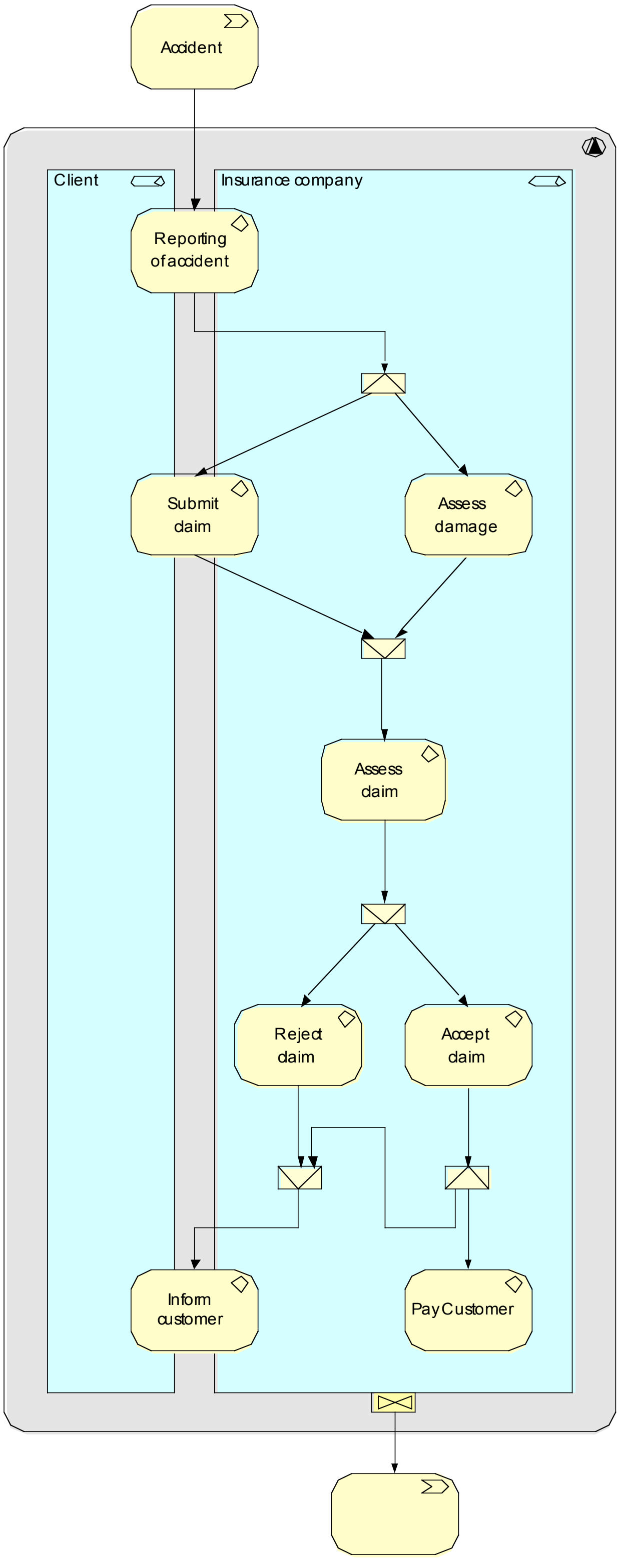


Fig. 10.14 Swimming lanes of role involvement

Chapter 11
Work Objects Modelling

This Chapter isconcerned with a viewpoint for the resources manipulated by work-systems. The focus of this viewpoint is on the questions:

1. What resources are manipulated by the work-system?
2. What is their structure?
3. What is their life-cycle/behavior?

Before presenting our viewpoint, whichshould be regarded as an example viewpoint and not as the ultimate answer, however, we request readers to first define their own viewpoint aimed at meeting these questions.

We start with an exercise:

1. Define a viewpoint suitable for answering the above questions.
2. Apply this viewpoint to the running case provided in Appendix A. Try to answer the questions posed above.
3. Have someone else use your viewpoint in creating models for the same domain. Do they produce the same model?

11.1 Way of thinking

Resources are regarded as objects that are essentially manipulated in work-flows. Being objects, these resources may actually have an internal structure as well. Even more, these resources will typically have their own life-cycle as well.

11.2 Way of modelling

Our proposed way of modelling is, again, primarily inspired by ArchiMate [59], but in this case also by [27]. An example resource model is depicted in Figure 11.1. A claim handling report consists of an accident report, a claim, a damage assessment and a claim assessment.

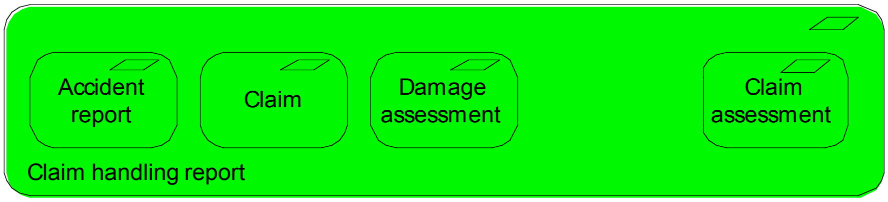


Fig. 11.1 Example resource model

The example resource as depicted in Figure 11.1 does not yet make clear how these resources are used in a work-flow such as the claim handling work-flow. This has been illustrated in Figure 11.2. It is now also shown how the or-split between assess claim and accept/reject claim takes as input the claim assessment.

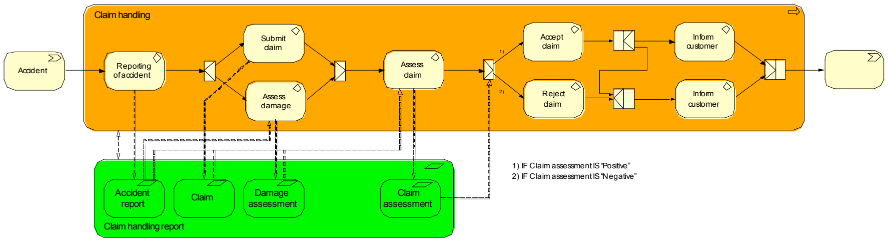


Fig. 11.2 Resources tied to work-flows

The resources from Figure 11.2 can also be related to an underlying ORM model. This is illustrated in Figure 11.3, where we have also further refined the ORM model of the insurance claim domain.

Requirements governing the life-cycle of resources may be modeled as depicted in Figure 11.4. At the bottom of this Figure we have used an abbreviation in line with the abbreviations suggested in Figure 6.9. A translation to an ORM based notation as provided in Figure 11.5.

Finally, the structure of resources may contain constraints which on their turn constrain the allowed flow of activities. This is illustrated in Figure 11.6. The mandatory roles in the left hand part of the diagram imply the temporal dependencies in the right hand side diagram.

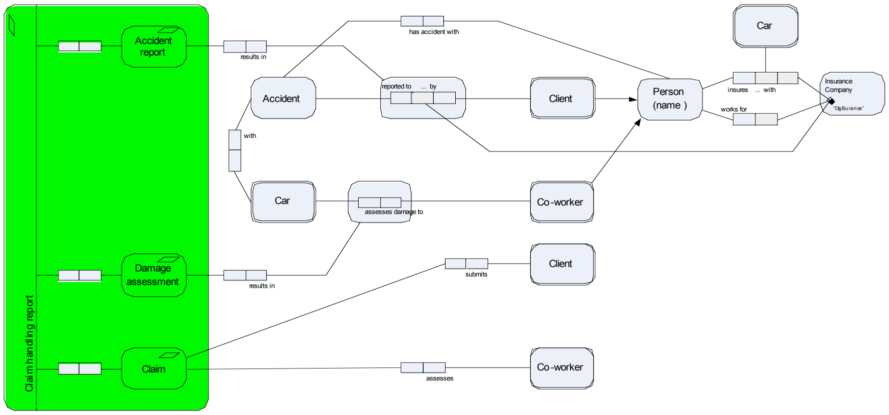


Fig. 11.3 Resource model related to ORM domain model

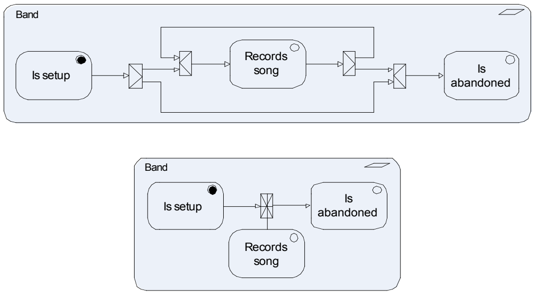


Fig. 11.4 Life-cycle of resources

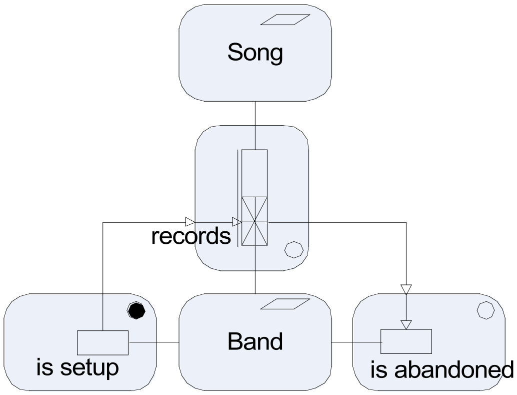


Fig. 11.5 Life-cycle of bands in ORM notation

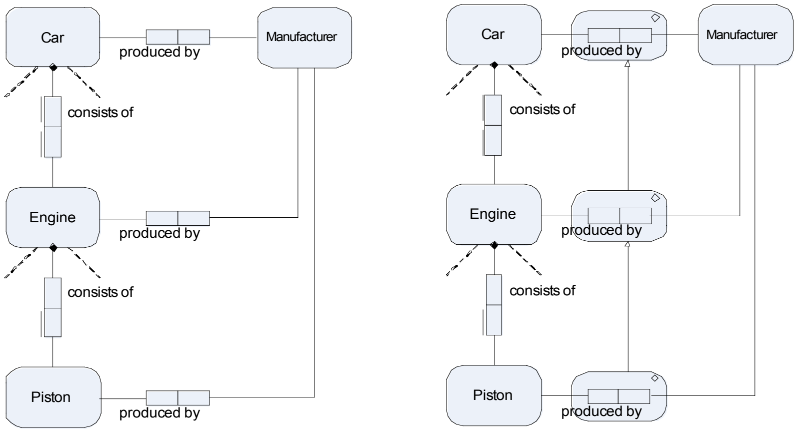


Fig. 11.6 Implied temporal dependencies

Appendix A
Car-rental case

Op een grote, sterk groeiende luchthaven is naast een aantal concurrenten een autoverhuurbedrijf gevestigd. Het autoverhuurbedrijf is een zelfstandig opererende onderneming die werkt op een franchise basis. Dit uit zich in de herkenbare huisstijl en de onlangs geïntroduceerde clubkaart waarmee klanten kortingen bij alle aangesloten bedrijven kunnen krijgen. Inmiddels is er al een groot aantal klanten met een clubkaart.

De luchthaven is gevestigd bij een grote metropool, die een constante stroom van zakelijke bezoekers trekt en in de vakantieperioden een groot aantal toeristen, die worden aangetrokken door de stad, de stranden in de buurt en de natuurgebieden in het achterland. Deze groepen vormen de clientèle van het verhuurbedrijf. Zowel de zakelijke reizigers als de toeristen nemen in negen van de tien gevallen een retour vlucht vanaf dezelfde luchthaven.

Het bedrijf bestaat uit een ruime verkoopbalie in de aankomsthal van de luchthaven en een klein kantoortje in de parkeergarage. Verder heeft het bedrijf een nauwe relatie met een garagebedrijf gevestigd op het terrein van de luchthaven.

Aan de verkoopbalie werkt een tiental verkopers. Zij helpen klanten bij het uitzoeken van een geschikte auto en sluiten de huurcontracten af. Na afloop van de huurperiode komen de klanten naar de verkoopbalie om de betaling (alleen met creditcard) af te handelen. Om in aanmerking te kunnen komen voor een huurauto moeten klanten tenminste 25 jaar oud zijn, minimaal 12 maanden in het bezit van een rijbewijs, kredietwaardig zijn en geen negatief verzekeringsverleden hebben.

Wanneer een klant een auto wil huren, vraagt de verkoper altijd eerst wat de klant precies zoekt, waarvoor hij de auto gebruiken, bijvoorbeeld vakantie, verhuizing of zakelijk en voor welke periode hij de auto wil huren. De verkoper checkt of de klant een clubkaart heeft en adviseert op basis van de klantbehoefte een auto uit een bepaalde tariefgroep. Hij controleert daarbij ook of er zón auto in de gewenste periode beschikbaar is. Zo niet, dan zal hij de klant een ander type auto adviseren, of vragen of de huurperiode eventueel aangepast moet worden.

Als de klant met het advies akkoord gaat, vraagt de verkoper om de adresgegevens van de klant en de bestuurder(s) en stelt een offerte op. Daarnaast kijkt de verkoper of de klant nog aanvullende verzekeringen wil afsluiten, zoals een afkoop van eigen risico of een verzekering voor inzittenden. Dit wordt ook in de offerte opgenomen. Wanneer de klant ingaat op de offerte, dan maakt de verkoper een huurcontract nadat hij de kredietwaardigheid van de klant heeft gecontroleerd. Tot slot vraagt de verkoper of de klant direct een auto wil reserveren of dat hij alleen een voorreservering wil doen. Een voorreservering houdt in dat de klant alleen een reservering voor een bepaalde tariefgroep heeft maar niet voor een specifieke auto. Als de klant een reservering maakt betekent dat tevens dat hij ook daadwerkelijk die specifieke auto mee zal krijgen.

Veel klanten maken een telefonische (voor)reservering. Het autoverhuurbedrijf stuurt de offerte dan op, per post of per fax. De klant heeft nu 10 dagen, na dagtekening, de tijd om op de offerte in te gaan door deze ondertekend terug te sturen.

Als de klant komt om de auto op de te halen moet hij het huurcontract ondertekenen, en betalen. Dit gebeurt pas nadat de verkoper heeft gecontroleerd of de klant voldoet aan de voorwaarden. Daarnaast moet hij ook een borgsom betalen en wordt er een kopie van zijn rijbewijs gemaakt.

Daarna kunnen de klanten in een grote parkeergarage hun auto ophalen en terugbrengen. Daar worden ze opgevangen door een contractbeheerder, die hen naar de auto brengt en uitleg geeft over de werking (startonderbreking, lichten, ruitenwissers enz.). Ook wordt een schadeformulier ingevuld waarop wordt aangegeven wat de bekende schades zijn van die auto. Na afloop van de huurperiode kan de klant de auto hier weer inleveren.

Als de klant de auto in ontvangst genomen heeft, wordt dit onmiddellijk geregistreerd. Als de klant de auto heeft terug gebracht wordt deze gecontroleerd op schade, en wordt gekeken of de klant de auto afgetankt heeft. Vervolgens wordt geregistreerd dat de auto is terug gebracht en ontvangt de klant de borgsom terug. Een eventuele schade of een niet volle tank wordt verrekend met de borgsom.

Niet alle adviestrajecten leiden daadwerkelijk tot het verhuren van een auto. Soms informeren klanten alleen en soms gaan ze niet akkoord met de offerte. Maar ook het afsluiten van een huurcontract is nog geen garantie. Soms blijkt dat de klant niet kan betalen, maar het kan ook gebeuren dat de klant op het moment van afhalen toch liever een ander type auto wil. De verkoper zal dan kijken of er nog een dergelijke auto beschikbaar is en eventueel het contract aanpassen. Tenslotte gebeurt het ook nog wel eens dat een klant helemaal niet komt opdagen. In dat geval vervalt de reservering en kan de auto weer aan een ander worden verhuurd.

Alle offertes en huurcontracten worden bewaard in het verkoopdossier van de klant en 5 jaar in het archief bewaard. In alle gevallen waarbij een reservering uiteindelijk toch niet doorgaat, wordt er een aantekening gemaakt op het huurcontract. Wat wel jammer is, is dat het moeilijk is om overzicht te houden van notoir lastige klanten. Immers, de verkoper moet dan in het archief gaan zoeken of er van deze klant al een dossier bestaat en of er aantekeningen op de huurcontracten gemaakt zijn.

Glossary

Active system – A special kind of system that is conceived of as begin able to change parts of the universe.

Actor – A system element that is conceived of as having some involvement in a system activity. This involvement is a special kind of system link, referred to as an activity participation.

Architecting – The processes which tie definition, design and deployment to the explicit and implicit needs, desires and requirements of the usage context. Issues such as: business/IT alignment, stakeholders, limiting design freedom, negotiation between stakeholders, enterprise architectures, stakeholder communication, and outsourcing, typify these processes.

Architecture – A model of which the system description, the so-called architectural description, is used during system engineering to:

- express the fundamental organisation of the system domain in terms of components, their relationships to each other and to the environment and
- the principles guiding its evolution and design,

and which's explicit intend is to be used as a means:

- of communication & negotiation among stakeholders,
- to evaluate and compare design alternatives,
- to plan, manage, and execute further system development,
- to verify the compliance of a system implementation's.

Aspect system – An aspect-system S' of a system S, is a sub-system, where the set of model links in S' is a proper subset of the set of the links in S.

Autonomous system – An open active system (possibly also a responsive system, but not a reactive system) where at least one expression is an action. A human being and most (if not all) organisation can be regarded as autonomous system.

Component system – A component-system S' of a system S, is a sub-system, where the set of model concepts in S' is a proper subset of the set of entities in S.

Computerised information system – A sub-system of an information system, in which all activities are performed by one or several computer(s).

Conception – That what results, in the mind of a viewer, when they interpret a perception of a domain.

Concept – Any element from a conception that is not a link.

Concern – A stakeholder's interest in properties of a system, relative to their stakeholder goals and the potential role played by the system in achieving these goals. This usually pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

Construction process – A process aiming to realise and test a system that is regarded as a (possibly artificial) artifact that is not yet in operation.

Definition process – A process aiming to identify all requirements that should be met by the system, the system description, and the engineering process.
In literature this process may also be referred to as requirements engineering.

Definition – The description of the requirements that should be met by both the desired information system as well as the documents documenting this information system. In literature this is also referred to as requirements engineering.
With regards to the information system, the resulting descriptions should identify: *what* it should do, *how well* it should do this, and *why* it should do so.
With regards to the documentation of the information system, the descriptions should identify *what* should be documented, *how well* it should be documented, and *why*/*what-for* these documents are needed.

Deployment process – A process aiming to make a system operational, i.e. to implement the use of the system by its prospective users.

Deployment – The processes of delevering/implementating an information system to/in its usage context. The design of an information system is not enough to arrive at an operational system. It needs to be implemented-in/delivered-to a usage context.

Description – The result of a viewerviewer denoting a conceptionconception, using some language to express themselves.

Design process – A process aiming to design a system conform stated requirements. The resulting system design may range from high-level designs, such as an strategy or an architecture, to the detailed level of programming statements or specific worker tasks.

Design – The description of the design of system. These descriptions should identify *how* a system will meet the requirements set out in its definition. The resulting design may (depending on the design goals) range from high-level designs to the detailed level of programming statements or specific worker tasks.

Domain modelling – Modelling of the domains that are relevant to the information system being developed. The resulting models will typically correspond to *ontologies* of the domains. These domains can pertain to the information that will be processed by the information system, the processes in which the information system will play a role, the processing as it will occur inside the information system, etc. Understanding (and modelling) these domains is fundamental to the other activities in information system engineering.

Domain – Any 'part' or 'aspect' of the universe a viewer may have an interest in.

Dynamic system – A special kind of system that is conceived of as undergoing change in the cause of time.

Element – The elementary parts of a viewer's conception.

Enterprise engineering – A system engineering process aimed at the creation of an enterprise system.

Enterprise system – An enterprise, being a work system, and/or one of its subsystem.

Enterprise – A work system which performs a systematic and *purposeful* activity.

Environment – The environment of a domain is that part of a viewer's conception of a universe, which has a direct link to the domain.

Information system – A sub-system of an organisational system, comprising the conception of how the communication and information-oriented aspects of an organisation are composed and how these operate, thus leading to a description of the (explicit and/or implicit) communication-oriented and information-providing actions and arrangements existing within the organisational system.

Information – The knowledge increment brought about when a human actor receives a message. In other words, it is the difference between the conceptions held by a human actor *after* interpreting a received message and the conceptions held beforehand.

Installation process – A process aiming to make a system operational, i.e. to implement the use of the system by its prospective users.

Interest – The specific reason(s) why a viewer observes a domain.
In the case of a system, this this is usually a confluence of the systemic properties of interest to the system viewer and the aspects of the system that are considered relevant (by the system viewer to these systemic properties).

Link – Any element from a conception that relates two concepts.

Maintenance – A system which is operational in its usage context, does not remain operational by itself. Both technical and non-technical elements of the system need active maintenance to keep the system operational as is.

Meta model – A conception of a viewer's viewpoint on the world.

Method – An integrated combination of a: way of thinking, way of controlling, way of working, way of modelling (a technique), and a way of supporting aimed at obtaining results.

Model concept – A concept from a conception which is a model.

Model element – An element from a conception which is a model.

Model link – A link from a conception which is a model.

Modelling technique – The combination of a way of modelling and a way of communicating.

Modelling – The act of purposely abstracting a model from (what is conceived to be) a part of the universe.

Model – A purposely abstracted domain (possibly in conjunction with its environment) of some 'part' or 'aspect' of the universe a viewer may have an interest in.

For practical reasons, a model will typically be consistent and unambiguous with regards to some underlying semantic domain, such as logic.

Open active system – A system that is an open system as well as an active system.

Open system – A special kind of dynamic system that is conceived as reacting to external triggers, i.e. there may be changes inside the system due to external causes originating from the system's environment.

Organisational system – A special kind of system, being normally active and open, and comprising the conception of how an organisation is composed and how it operates (i.e. performing specific actions in pursuit of organizational goals, guided by organizational rules and informed by internal and external communication), where its systemic property are that it responds to (certain kinds of) changes caused by the system environment and, itself, causes (certain kinds of) changes in the system environment.

Organisation – A group of actors with a purpose, who:

- interact with each other,
- form a network of roles,
- make use of (the services of) other actors.

An organisation in itself is an actor as well, and may as such participate in yet another organisations.

Perception – That what results, in the mind of a viewer, when they observe a domain with their senses, and forms a specific pattern of visual, auditory or other sensations in their minds.

Perspective – A set of related interests in terms of which viewers may observe a domain.

Reactive system – An open active system where each expression of the system is a reaction, and where each impression immediately causes a reaction.

Responsive system – An open active system (possibly also a reactive system) where it holds for at least one expression that a certain impression or a temporal pattern of impressions is a necessary, but not a sufficient dynamic condition for its occurrence. The receipt of an order is a necessary impression to a "sales system", for the expression "delivery of the ordered goods", but it is not a sufficient condition.

Stakeholder goal – The end toward which effort is directed by a stakeholder, in which the system (of which the stakeholder is indeed a stakeholder) plays a role.

This may pertain to strategic, tactical or operational end. The role of the system may range from passive to active. For example, a financial controller's goal with regards to a future/changed system may be to control engineering costs, while the goal of users of the system may be to get their job done more efficiently.

Stakeholder – Some actor in a system engineering community with a specific stake pertaining to a system's development, its operation or any other aspects that are critical or otherwise important.

Examples are: Users, operators, owners, architects, engineers, testers, project managers, business management, ...

Sub-system – A sub-system S' of a system S, is a system where the set of elements in S' is a subset of the elements in S.

System concept – Any element from a system that is a concept.

System description – The description of a system.

System domain – A domain that is conceived to be a system, by some viewer, by the distinction from its environment, by its coherence, and because of its systemic property.

System element – Any element from a system.

System engineering community – A group of objects, such as actors and representations, which are involved in a system engineering process.

System engineering – A process involving aimed at producing a changed system, involving the execution of four sub-processes: definition, design, construction and installation of the system. Processes that may be executed sequentially, incrementally, interleaved, or in parallel.

System exposition – A description of all the elements of the system domain where each element is specified by all its relevant aspects and all the roles it plays, being of importance for the interestinterest of the viewer. (The system viewer may conceive one and the same thing in the system domain to play more than one role in the system.)

System link – Any element from a system that is a link.

System type – A type that determines the potential kinds of systemic property, elements of the system domain and roles of the elements in achieving the systemic properties.

System viewer – A viewer of a system domain.

Systemic property – A meaningful relationship that exists between the domain of elements considered as a whole, the system domain and its environment.

System – A special model of a system domain, whereby all the things contained in that model are transitively coherent, i.e. all of them are directly or indirectly related to each other and form a coherent whole.

A system is conceived as having assigned to it, as a whole, a specific characterisation (a non-empty set of systemic property) which, in general, cannot be attributed exclusively to any of its components.

Universe – The ‘world’ under consideration.

Viewer – An actor perceiving and conceiving (part of) a domain.

Viewing method – The method by means of which a view is constructed from models. A viewing method will typically make use of modelling methods for the creation of these models.

Viewpoint – A specification of the conventions for constructing and using views.

This involves: a way of thinking, a way of modelling, a way of communicating, a way of working, a way of supporting and a way of using

View – A set of model descriptions of a domain from the perspective of a related set of interests of a viewer.

Way of communicating – describes how the abstract concepts from the way of modelling are communicated to human beings, for example in terms of a textual or a graphical notation.

The way of communicating essentially forms the bridge between the way of modelling and the way of working, it matches the abstract concepts of the way of modelling to the pragmatic needs of the way of working.

Note that it may very well be the case that different modelling technique are based on the same way of modelling, yet use different notations.

Way of conceiving – A set of modelling concepts by which viewers are to observe domains. This usually takes the form of a meta models.

Way of controlling – The managerial aspects of system engineering. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [52, 92]).

Way of delivering – The languages, conventions and documentation standards used in producing deliverables during system engineering.

Way of describing – The medium and 'notations' used to represent the concepts as identified in a way of conceiving. It describes how the abstract concepts from the way of conceiving are communicated to human beings, for example in terms of a textual or a graphical notation.

Note that it may very well be the case that different modelling techniques are based on the same way of conceiving, yet use different notations.

Way of learning – The process and measures that enable continuous improvement of consecutive executions of the method. It should provide answers to questions such as: *How can we learn from past experiences? How can the method be refined to reflect new experiences?*

Way of modelling – Identifies the *core concepts* of the language that may be used to denote, analyze, visualize and/or animate system descriptions.

Way of supporting – The support to system development that is offered by (possibly automated) tools. In general, a way of supporting is supplied in the form of some computerised tool (see for instance [63]).

Way of thinking – Articulates the assumptions on the kinds of problem domains, solutions, engineers, analysts, etc. This notion is also referred to as *die Weltanschauung* [91, 104], *underlying perspective* [62] or *philosophy* [7].

Way of using – Identification of heuristics that:

- define situations, classes of viewers and interests, for which a given viewpoint is most suitable,
- provide guidance in tuning a given viewpoint to specific situations, classes of viewers and their interest at hand.

  For example, in terms of the set of modelling concepts to be used, effective notations, visualisations, etc.

Way of working – Structures (parts of) the way in which a system is engineered. It defines the possible tasks, including sub-tasks, and ordering of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.

Work system – An open active system in which actor perform processes using information, technologies, and other resources to produce products and/or services for internal or external actors.

References

1. UML 2.0 Superstructure Specification – Final Adopted Specification. Tech. Rep. ptc/03–08–02, OMG (2003). URL `http://www.omg.org`
2. Aalst, W.v.d., Hofstede, A.t.: YAWL: yet another workflow language. Information Systems **30**(4), 245–275 (2005)
3. Abiteboul, S., Hull, R.: IFO: A Formal Semantic Database Model. ACM Transactions on Database Systems **12**(4), 525–565 (1987)
4. Ackoff, R.: Towards a System of System Concepts. Management Science **17** (1971)
5. Alter, S.: A general, yet useful theory of information systems. Communications of the Association for Information Systems **1**(13) (1999). URL `http://cais.isworld.org/articles/1-13/default.asp`
6. Alter, S.: The work system method for understanding information systems and information system research. Communications of the Association for Information Systems **9**(9), 90–104 (2002). URL `http://cais.isworld.org/articles/default.asp?vol=9&art=6`
7. Avison, D.: Information Systems Development: Methodologies, Techniques and Tools, 2nd edn. McGraw–Hill, New York, New York, USA (1995), ISBN-10: 0077092333
8. Barros, A.: On the Conceptualisation of Workflow Specifications. Ph.D. thesis, University of Queensland, Brisbane, Queensland, Australia (1998)
9. Batini, C., Ceri, S., Navathe, S.: Conceptual Database Design – An Entity–Relationship Approach. Benjamin Cummings, Redwood City, California, USA (1992)
10. Bemelmans, T.: Bestuurlijke Informatiesystemen en Automatisering, 7th edn. Kluwer, Deventer, The Netherlands, EU (1998), ISBN-10: 9026727984. In Dutch
11. Bertalanffy, L.v.: General Systems Theory – Foundations, Development, Applications, revised edn. George Braziller, New York, New York, USA (2001), ISBN-10: 0807604534
12. Bleeker, A., Proper, H., Hoppenbrouwers, S.: The Role of Concept Management in System Development – A practical and a theoretical perspective. In: J. Grabis, A. Persson, J. Stirna (eds.) Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, pp. 73–82. Faculty of Computer Science and Information Technology, Riga, Latvia, EU (2004), ISBN-10: 998497670X
13. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modelling Language User Guide. Addison Wesley, Reading, Massachusetts, USA (1999), ISBN-10: 0201571684
14. Bronts, G., Brouwer, S., Martens, C., Proper, H.: A Unifying Object Role Modelling Approach. Information Systems **20**(3), 213–235 (1995)
15. Checkland, P.: Systems thinking, systems practice. John Wiley & Sons, New York, New York, USA (1981), ISBN-10: 0471279110
16. Creasy, P., Proper, H.: A Generic Model for 3–Dimensional Conceptual Modelling. Data & Knowledge Engineering **20**(2), 119–162 (1996)

17. Dietz, J.: Enterprise Ontology – Theory and Methodology. Springer, Berlin, Germany, EU (2006), ISBN-10: 9783540291695
18. Eertink, H., Janssen, W., Oude Luttighuis, P., Teeuw, W., Vissers, C.: A Business Process Design Language. In: Proceedings of the First World Congress on Formal Methods (1999)
19. Elmasri, R., Navathe, S.: Fundamentals of Database Systems. Benjamin Cummings, Redwood City, California, USA (1994). Second Edition
20. Elmasri, R., Weeldreyer, J., Hevner, A.: The category concept: An extension to the entity–relationship model. Data & Knowledge Engineering **1**, 75–116 (1985)
21. Embley, D., Kurtz, B., Woodfield, S.: Object–Oriented Systems Analysis – A model–driven approach. Yourdon Press, New York, New York, USA (1992), ASIN 0136299733
22. Engels, G., Gogolla, M., Hohenstein, U., Hülsmann, K., Löhr–Richter, P., Saake, G., Ehrich, H.D.: Conceptual modelling of database applications using an extended ER model. Data & Knowledge Engineering **9**(4), 157–204 (1992)
23. Falkenberg, E., Verrijn–Stuart, A., Voss, K., Hesse, W., Lindgreen, P., Nilsson, B., Oei, J., Rolland, C., Stamper, R.a. (eds.): A Framework of Information Systems Concepts. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU (1998), ISBN-10: 3901882014
24. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: a framework for integrating multiple perspectives in system development. International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments **2**(1), 31–58 (1992)
25. Franckson, M., Verhoef, T. (eds.): Introduction to ISPL. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU (1999), ISBN-10: 9076304858
26. Franckson, M., Verhoef, T. (eds.): Specifying Deliverables. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU (1999), ISBN-10: 9076304823
27. Frederiks, P.: Object–Oriented Modeling based on Information Grammars. Ph.D. thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (1997), ISBN-10: 9090103384
28. Frederiks, P., Weide, T.v.d.: Deriving and paraphrasing information grammars using object–oriented analysis models. Acta Informatica **38**(7), 437–88 (2002)
29. Frederiks, P., Weide, T.v.d.: Information Modeling: the process and the required competencies of its participants. In: F. Meziane, E. Métais (eds.) 9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004), Manchester, United Kingdom, EU, *Lecture Notes in Computer Science*, vol. 3136, pp. 123–134. Springer, Berlin, Germany, EU (2004)
30. Greefhorst, D., Koning, H., Vliet, H.v.: Dimensies in architectuurbeschrijvingen. Informatie **45**(11), 22–27 (2003). In Dutch
31. Halpin, T.: Conceptual Schema and Relational Database Design, 2nd edn. Prentice–Hall, Englewood Cliffs, New Jersey, USA (1995)
32. Halpin, T.: Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design. Morgan Kaufmann, San Mateo, California, USA (2001), ISBN-10: 1558606726
33. Halpin, T.: Objectification. In: T. Halpin, K. Siau, J. Krogstie (eds.) Proceedings of the 10th Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'05), held in conjunctiun with the 17th Conference on Advanced Information Systems (CAiSE'05), Porto, Portugal, EU, pp. 519–532. FEUP, Porto, Portugal, EU (2005), ISBN-10: 9727520774
34. Halpin, T., Proper, H.: Subtyping and Polymorphism in Object–Role Modelling. Data & Knowledge Engineering **15**, 251–281 (1995)
35. Heuvel, W.J.v.d., Proper, H.: De pragmatiek van Architectuur. Informatie **44**(11), 12–14 (2002). In Dutch
36. Hofstede, A.t., Proper, H., Weide, T.v.d.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems **18**(7), 489–523 (1993)
37. Hofstede, A.t., Weide, T.v.d.: Expressiveness in conceptual data modelling. Data & Knowledge Engineering **10**(1), 65–100 (1993)

38. Hofstede, A.t., Weide, T.v.d.: Fact Orientation in Complex Object Role Modelling Techniques. In: T. Halpin, R. Meersman (eds.) Proceedings of the First International Conference on Object–Role Modelling (ORM–1), pp. 45–59. Key Centre for Software Technology, University of Queensland, Brisbane, Australia, Magnetic Island, Queensland, Australia (1994)
39. Hofstede, A.t., Weide, T.v.d.: Deriving Identity from Extensionality. International Journal of Software Engineering and Knowledge Engineering **8**(2), 189–221 (1997)
40. Hoppenbrouwers, S.: Freezing Language; Conceptualisation processes in ICT supported organisations. Ph.D. thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (2003), ISBN-10: 9090173188
41. Hoppenbrouwers, S., Bleeker, A., Proper, H.: Facing the Conceptual Complexities in Business Domain Modeling. Computing Letters **1**(2), 59–68 (2005)
42. Recommended Practice for Architectural Description of Software Intensive Systems. Tech. Rep. IEEE P1471–2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA (2000), ISBN-10: 0738125180. URL `http://www.ieee.org`
43. Iivari, J.: Contributions to the theoretical foundations of systemeering research and the PI-OCO model. Tech. Rep. 150, University of Oulu, Oulu, Finland, EU (1983), ISBN-10: 9514215435
44. ISO: Information technology – Open Distributed Processing – Reference model: Architecture (1996). URL `http://www.iso.org`. ISO/IEC 10746–3:1996(E)
45. Information technology – Open Distributed Processing – Reference model: Foundations (1996). URL `http://www.iso.org`. ISO/IEC 10746–2:1996(E)
46. ISO: Kwaliteit van softwareprodukten. ten Hagen & Stam, Den Haag, The Netherlands, EU (1996), ISBN-10: 9026724306. In Dutch
47. Information technology – Open Distributed Processing – Reference model: Architectural semantics (1998). URL `http://www.iso.org`. ISO/IEC 10746–4:1998(E)
48. Information technology – Open Distributed Processing – Reference model: Overview (1998). URL `http://www.iso.org`. ISO/IEC 10746–1:1998(E)
49. Software engineering – Product quality – Part 1: Quality model (2001). URL `http://www.iso.org`. ISO/IEC 9126–1:2001
50. Janssen, R., Proper, H., Bosma, H., Verhoef, D., Hoppenbrouwers, S.: Developing an Architecture Method Library. Tech. rep., Ordina Institute, Gouda, The Netherlands, EU (2001)
51. Jonkers, H., Veldhuijzen van Zanten, G., Buuren, R.v., Arbab, F., Boer, F.d., Bonsangue, M., Bosma, H., Doest, H.t., Groenewegen, L., Guillen Scholten, J., Hoppenbrouwers, S., Iacob, M.E., Janssen, W., Lankhorst, M., Leeuwen, D.v., Proper, H., Stam, A., Torre, L.v.d.: Towards a Language for Coherent Enterprise Architecture Descriptions. In: M. Steen, B. Bryant (eds.) 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003), Brisbane, Queensland, Australia, pp. 28–39. IEEE, Los Alamitos, California, USA (2003), ISBN-10: 0769519946
52. Kensing, F.: Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon–DeMarco Approach. In: T. Bemelmans (ed.) Beyond Productivity: Information Systems Development for Organizational Effectiveness, Amsterdam, The Netherlands, EU, pp. 325–338. North–Holland, Amsterdam, The Netherlands, EU (1984)
53. Kishore, R., Zhang, H., Ramesh, R.: A Helix–Spindle Model for Ontological Engineering. Communications of the ACM **47**(2), 69–75 (2004)
54. Kotonya, G., Sommerville, I.: Viewpoints for requirements definition. IEE/BCS Software Engineering Journal **7**(6), 375–387 (1992)
55. Kristen, G.: Object Orientation – The KISS Method, From Information Architecture to Information System. Addison Wesley, Reading, Massachusetts, USA (1994), ISBN-10: 0201422999
56. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software **12**(6), 42–50 (1995)
57. Kruchten, P.: The Rational Unified Process: An Introduction, 2nd edn. Addison Wesley, Reading, Massachusetts, USA (2000), ISBN-10: 0201707101
58. Langefors, B.: Editorial notes to: Computer Aided Information Systems Analysis and Design. Studentlitteratur, Lund, Sweden, EU (1971)

59. Lankhorst, M., et al.: Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer, Berlin, Germany, EU (2005), ISBN-10: 3540243712
60. Levy, A.: Basic Set Theory. Springer, Berlin, Germany, EU (1979)
61. Lindgreen, P.: A General Framework for Understanding Semantic Structures. In: E. Falkenberg, C. Rolland, E. El Sayed (eds.) Information System Concepts: Improving the understanding – Proceedings of the second IFIP WG8.1 working conference (ISCO–2), Alexandria, Egypt. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1992), ISBN-10: 0444895078
62. Mathiassen, L.: Systemudvikling og Systemudviklings–Metode. Ph.D. thesis, Aarhus University, Aarhus, Denmark, EU (1981). In Danish
63. McClure, C.: CASE is Software Automation. Prentice–Hall, Englewood Cliffs, New Jersey, USA (1989), ISBN-10: 0131193309
64. Meriam–Webster: Meriam–Webster Online, Collegiate Dictionary (2003). URL `http://www.webster.com`
65. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design: a fact oriented approach. Prentice–Hall, Englewood Cliffs, New Jersey, USA (1989), ASIN 0131672630
66. Ogden, C., Richards, I.: The Meaning of Meaning – A Study of the Influence of Language upon Thought and of the Science of Symbolism. Magdalene College, University of Cambridge, Oxford, United Kingdom, EU (1923)
67. Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Assche, F.v., Verrijn–Stuart, A.: Information Systems Methodologies: A Framework for Understanding. Addison Wesley, Reading, Massachusetts, USA (1988), ISBN-10: 0201544431
68. Olle, T., Sol, H., Tully, C. (eds.): Information Systems Design Methodologies: A feature analysis. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1983), ISBN-10: 0444867058
69. Olle, T., Sol, H., Verrijn–Stuart, A. (eds.): Information Systems Design Methodologies: A Comparative Review. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1982), ISBN-10: 0444864075
70. Olle, T., Sol, H., Verrijn–Stuart, A. (eds.): Information Systems Design Methodologies: Improving the practice. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1986)
71. Olle, T., Sol, H., Verrijn–Stuart, A. (eds.): Information Systems Design Methodologies: Computerized assistance during the information systems life cycle. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1988), ISBN-10: 0444705120
72. Paulk, M., Curtis, B., Chrissis, M., Weber, C.: Capability Maturity Model for Software, Version 1.1. Tech. Rep. SEI–93–TR–024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA (1993)
73. Peirce, C.: Volumes I and II – Principles of Philosophy and Elements of Logic. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA (1969), ISBN-10: 0674138007
74. Peirce, C.: Volumes III and IV – Exact Logic and The Simplest Mathematics. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA (1969), ISBN-10: 0674138005
75. Peirce, C.: Volumes V and VI – Pragmatism and Pragmaticism and Scientific Metaphysics. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA (1969), ISBN-10: 0674138023
76. Peirce, C.: Volumes VII and VIII – Science and Philosophy and Reviews, Correspondence and Bibliography. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA (1969), ISBN-10: 0674138031
77. Proper, H.: A Theory for Conceptual Modelling of Evolving Application Domains. Ph.D. thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (1994), ISBN-10: 909006849X
78. Proper, H. (ed.): ISP for Large–scale Migrations. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU (2001), ISBN-10: 9076304882

79. Proper, H.: Architecture–driven Information Systems Engineering. DaVinci Series. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU (2004)
80. Proper, H., Bleeker, A., Hoppenbrouwers, S.: Object–Role Modelling as a Domain Modelling Approach. In: J. Grundspenkis, M. Kirikova (eds.) Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'04), held in conjunctiun with the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004),, vol. 3, pp. 317–328. Faculty of Computer Science and Information Technology, Riga, Latvia, EU (2004), ISBN-10: 9984976718
81. Proper, H., Hoppenbrouwers, S.: Concept Evolution in Information System Evolution. In: J. Gravis, A. Persson, J. Stirna (eds.) Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Riga, Latvia, EU, pp. 63–72. Faculty of Computer Science and Information Technology, Riga, Latvia, EU (2004), ISBN-10: 998497670X
82. Proper, H., Hoppenbrouwers, S., Veldhuijzen van Zanten, G.: Communication of Enterprise Architectures. In: Enterprise Architecture at Work: Modelling, Communication and Analysis [59], pp. 67–82, ISBN-10: 3540243712
83. Proper, H., Verrijn–Stuart, A., Hoppenbrouwers, S.: Towards Utility–based Selection of Architecture–Modelling Concepts. In: S. Hartmann, M. Stumptner (eds.) Proceedings of the Second Asia–Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, New South Wales, Australia, *Conferences in Research and Practice in Information Technology Series*, vol. 42, pp. 25–36. Australian Computer Society, Sydney, New South Wales, Australia (2005), ISBN-10: 1920682252
84. Reeves, J., Marashi, M., Budgen, D.: A software design framework or how to support real designers. IEE/BCS Software Engineering Journal **10**(4), 141–155 (1995)
85. Reijswoud, V.v., Dietz, J.: DEMO Modelling Handbook, vol. 1, 2nd edn. Delft University of Technology, Delft, The Netherlands, EU (1999)
86. Reijswoud, V.v., Mulder, J., Dietz, J.: Commucation Action Based Business Process and Information Modelling with DEMO. The Information Systems Journal **9**(2), 117–138 (1999)
87. Ropohl, G.: Philosophy of Socio–Technical Systems. In Society for Philosophy and Technology **4**(3) (1999)
88. SBVR Team: Semantics of Business Vocabulary and Rules (SBVR). Tech. Rep. dtc/06–03–02, Object Management Group, Needham, Massachusetts, USA (2006). URL `http://www.omg.org/docs/dtc/06-03-02.pdf`
89. Seligmann, P., Wijers, G., Sol, H.: Analyzing the Structure of I.S. Methodologies, an alternative approach (1989)
90. Simon, H.: The architecture of complexity. In: Proceedings of the American Philosophical Society, vol. 106, pp. 467–482 (1962)
91. Sol, H.: A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In: T. Olle, H. Sol, C. Tully (eds.) Information Systems Design Methodologies: A Feature Analysis, Amsterdam, The Netherlands, EU, pp. 1–7. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1983), ISBN-10: 0444867058
92. Sol, H.: Information Systems Development: A Problem Solving Approach. In: W. Cotterman, J. Senn (eds.) Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy. Georgia State University, Atlanta, Georgia (1988)
93. Sowa, J., Zachman, J.: Extending and formalizing the framework for information systems architecture. IBM Systems Journal **31**(3), 590–616 (1992)
94. Tapscott, D., Caston, A.: Paradigm Shift – The New Promise of Information Technology. McGraw–Hill, New York, New York, USA (1993), ASIN 0070628572
95. Thomson, A., Martinet, A.: A Practical English Grammar, fourth edn. Oxford University Press, Oxford, United Kingdom, EU (1986), ISBN-10: 3810905798
96. TOGAF – The Open Group Architectural Framework (2005). URL `http://www.togaf.org`

97. Veld, J.i.t.: Analyse van organisatieproblemen – Een toepassing van denken in systemen en processen. Stenfert Kroese, Leiden, The Netherlands, EU (1992), ISBN-10: 9020722816. In Dutch
98. Veldhuijzen van Zanten, G., Hoppenbrouwers, S., Proper, H.: System Development as a Rational Communicative Process. In: N. Callaos, D. Farsi, M. Eshagian-Wilner, T. Hanratty, N. Rish (eds.) Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics, vol. XVI, pp. 126–130 (2003), ISBN-10: 9806560019
99. Verhoef, T.: Effective Information Modelling Support. Ph.D. thesis, Delft University of Technology, Delft, The Netherlands, EU (1993), ISBN-10: 9090061762
100. Verrijn–Stuart, A., Olle, T. (eds.): Methods and Associated Tools for the Information Systems Life Cycle. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1991), ISBN-10: 0444820744
101. Weide, T.v.d., Paulussen, G.: Domain Modeling. DaVinci Series of Lecture Notes. Institute for Information and Computing Sciences, Radboud University Nijmegen (2006)
102. Wijers, G., Heijes, H.: Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In: B. Steinholz, A. Sølvberg, L. Bergman (eds.) Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering, Stockholm, Sweden, EU, *Lecture Notes in Computer Science*, vol. 436, pp. 88–108. Springer, Berlin, Germany, EU (1990), ISBN-10: 3540526250
103. Wintraecken, J.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer, The Netherlands, EU (1990)
104. Wood–Harper, A., Antill, L., Avison, D.: Information Systems Definition: The Multiview Approach. Blackwell, Oxford, United Kingdom, EU (1985), ISBN-10: 0632012168
105. xAF working group: Extensible Architecture Framework version 1.1 (formal edition). Tech. rep. (2006). URL `http://www.xaf.nl`
106. Zachman, J.: A framework for information systems architecture. IBM Systems Journal **26**(3) (1987)

About the authors

Erik Proper is Principal Consultant at Capgemini and Professor in Information Systems at the Radboud University Nijmegen. He has a mixed industrial and academic background. His interests lie mainly in the field of conceptual modelling, enterprise engineering and enterprise architecting. He was co-initiator of the ArchiMate project, and currently also serves on the board of the ArchiMate foundation as well as the Netherlands Architecture Forum (NAF).