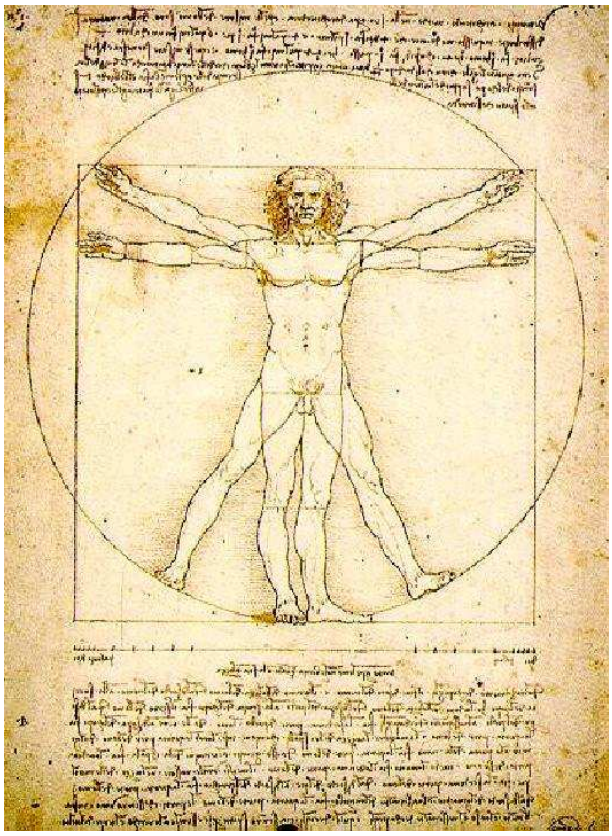
Architecture-driven Information Systems Engineering

H.A. Proper

Version of: 25-05-2005



The DAVINCI Series – The Art & Craft of System Alignment

Architecture-driven Information Systems Engineering

H.A. Proper

This textbook was entirely produced by means of open source software, in particular: LaTeX for type setting, Kile for editting, ISpell as spelling checker and XFig for drawings.

Contents

The DAVINCI Series

Version:
16-02-05

The subtitle of the DAVINCI series of lecture notes is *The Art & Craft of Information Systems Engineering*. On the one hand, this series of lecture notes takes a fundamental view (*craft*) on the field information systems engineering. At the same time, it does so with an open eye to practical experiences (the *art*) gained from information system engineering in industry.

The kinds of information systems we are interested in range from personal information appliances to enterprise-wide information processing. Even more, we regard an information system as a system that "handles" information, where "handling" should be interpreted in a broad fashion. The actors that do this "handling" can be computers, but can equally well be other "symbol wielding machines, but can also be humans. The mix of humans and computers/machines in information systems makes the field of information system engineering particularly challenging.

The concept of "information" itself is very much related to the concepts of *data*, *knowledge* and *communication*. Based on [FVSV+98], we will (throughout the DAVINCI series) use the following definitions:

Data – Any representation in some language. Data is therefore simply a collection of symbols that may, or may not, have some meaning to some actor.

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.

Knowledge – A relatively stable, and usually mostly consistent, set of conceptions posessed by a single (possibly composed) actor.

In more popular terms: "an actor's picture of the world".

Communication – An exchange of messages, i.e. a sequence of mutual and alternating message transfers between at least two human actors, called communication partners, whereby these messages represent some knowledge and are expressed in languages understood by all communication partners, and whereby some amount of knowledge about the domain of communication and about the action context and the goal of the communication is made present in all communication partners.

When referring to an information system, we therefore really refer to systems that enable the communication/sharing of knowledge by means of the representation (by human actors), storage, processing, retrieval, and presentation (to human actors) of the underlying representations (data). This also implies that we will treat *information retrieval systems*, *knowledge-based systems*, *groupware systems*, etc., as special classes of information systems.

The lecture notes in the DAVINCI series have been organized around five key aspects of an information system's life-cycle:

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.

Design – The description of the design of an information system. These descriptions should identify *how* an information 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.

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.

Maintenance – An information 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 information system operational as is.

Architecting – The processes which tie definition, design, deployment and maintenance 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.

Domain modeling – Modeling 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 modeling) these domains is fundamental to the other activities in information system engineering.

For each these aspects, attention will be paid to relevant theories, methods and techniques to execute the tasks involved. When put together, these aspects can be related as depicted in figure 1.

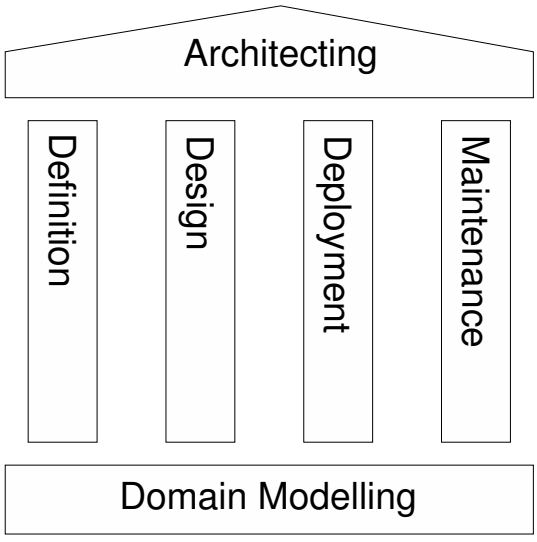


Figure 1: Aspects of Information Systems Engineering

The use of the name DAVINCI originates from earlier work [Pro98] done on architecture-driven information systems engineering. The work reported in [Pro98] was the result of a confrontation between industrial practice and a theoretical perspective on information systems and their

evolution [Pro94]. The result was a shared vision on the architecture-driven development of information systems by a Dutch IT consultancy firm. In this shared vision, a foundation was laid for an integrated view on information system engineering. At that stage, the name "DAVINCI" was also selected. Not as some artificial acronym, but rather to honour an inspiring artist, scientist, inventor and architect. To us he personifies a balance between art and engineering, between human and technology.

After the development of the first DAVINCI version, a more elaborate version [Pro04] was developed at the Radboud University Nijmegen in the form of lecture notes associated to a course on *Architecture & Alignment*. In this version, a more fundamental outlook on information system development was added to complement the practical orientation of the first version.

As a third step, we have now taken on the underlying philosophy of the first two DAVINCI documents, and used this as the source of inspiration to shape an entire line of lecture notes for a number of mutually related courses on different aspects of information systems engineering. In making this step we have also been able to anchor some of the fundamental research results from the co-authors, on subjects such as information modeling [BHW91, BW92a, HW93, HPW93, PW94, BBMP95, CHP96, CP96, PW95a, BFW96, HPW97, Pro97, HVH97, FW02, FW04a], information retrieval [BW90, BW91, BW92b, BB97, WBW00, SFG+00, PB99, PPY01, WBW01], (enterprise) information architecture [JLB+04] and information system engineering [Pro01, VHP04] into the core of the DAVINCI series.

Course Description

Short description

This course comprises three parts. It starts by discussing information systems and information systems engineering from an abstract point of view. As such, it ties the courses of the previous courses in the DAVINCI series together. This abstract view translates to taking a system theoretical perspective on information systems, their engineering and the role of architecture in bounding/guiding the engineering processes.

We then move on to the architecture level, by discussing both the definition of architecture in an information systems engineering context, its need, as well as its potential role as a means of negotiation and communication. Special attention will be paid to the mechanism of *architecture principles* as a mechanism to guide/bound the engineering processes.

In the third part, we zoom in on aspects of the process of architecting information systems, in particular on communication strategies and negotiation between different stakeholders. An information system involves organisational, human and technological issues, leading to a plethora of stakes and stakeholders. These stakes need to be balanced during the development of an information system's architecture.

Learning goals

After this course, students are able to:

1. argue about the different aspects of information system engineering and position the methods & techniques as presented in the other courses in the DAVINCI series relative to these aspects,
2. reason about organisations, information systems, and computerised information systems at an abstract (system theoretical) level,
3. argue the need, and potential role, for architecture in the context of information systems engineering,
4. discuss and relate different definitions of architecture,
5. reason about different levels of abstraction at which an information system, and its context, can be modelled/designed/architected, as well as identify situations for which these levels are relevant,
6. reason about the selection of modelling techniques, for given goals and situations during information systems engineering, that are most apt to the situation at hand,
7. develop, and reason about, communication and negotiation strategies relevant to information systems engineering.

Topics

1. System theory
2. Viewpoints

3. Architecture
4. Architecture principles
5. Communication & negotiation
6. Stakeholders

Outline

To be done

Preface

Version: 04-04-05

In 2005, the course "System Theory: Structure and Design" is taught for the first time. It is a continuation of the Architecture & Alignment course which was taught in the past as part of the information science curriculum. In the new curriculum, this course was renamed and moved to the third year. This text-book, which is a continuation of an older version [Pro04], is to be used as the lecture notes for this course. Over the next years, this course will be extended with an additional block of 3ec focussing on design rationale, eventually leading to an integrated course of 6ec focussing in Information Systems Architecture.

The intention of this document is that it evolves into a text-book on architecture-driven information system engineering at the Radboud University. The current version does require the reader to have a considerable willingness to read between the lines, as the text is still rather stenographic in nature and while parts of the text-book are still sketchy. Over the next few years, the text should be improved. The priority of this initial version of this textbook is on completeness of the topics that need to be covered, rather than readability and completeness of text. Students are advised to make notes during lectures.

The current version of this text-book shares two chapters with another course in the DAVINCIseries of lecture notes. These chapters (chapter 2 and 3) will be forked into two distinct versions of the next years. An introductory version will become part of the 1th year course "Information Intensive Organizations" (Modelleren van Organisaties), while a more fundamental version will remain part of this (3rd year) course.

My personal interest in the field of architecture-driven development of information systems was raised in 1997/1998 when I was employed as a consultant by Origin (now called Atos-Origin) and was first exposed to the field. This lead to a first document referred to as "Da Vinci" [Pro98]. Management at Origin gave me the freedom – for which I am still grateful – to work on "Da Vinci", drawing together many valuable ideas on architecture-driven development of information systems from several colleagues from within Origin. The resulting document stipulated a shared Origin vision on architecture-driven development of information systems. The name "Da Vinci" is not some artificial acronym. The name was chosen by me to honour an inspiring artist, scientist, inventor and architect.

To acknowledge the origins of my interest in the field, I have chosen to also use the name "Da Vinci" on this new document.

Needless to say that any feedback from either students or colleagues is more than welcome. Please sent any comments to the author at: `E.Proper@acm.org`

To do:

- Add a running example.
- Add questions from past exams.

Chapter 1

Introduction

Version:
22-04-05

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.

From: "The Restaurant at the End of the Universe", Douglas Adams, Pan Books Ltd.

This version of the introduction is still a mix between Dutch and English. The current contents is based on an older version (in English) and more recent materials from my inaugural address, which was written in Dutch. Over the next few years, this chapters content will be integrated further and converted into (International) English.

1.1 The digital era

This section is based on [Pro03] and still needs to be trimmed down to better fit the context of this textbook. It, obviously, also needs to be translated.

Onze maatschappij kan niet meer zonder computers. De rol van computers is allang niet meer beperkt tot de voor de hand liggende voorbeelden zoals tekstverwerkers, spelcomputers of websites. Ook allerlei andere vormen van reeds bestaande technologie raken steeds meer "doordrenkt" met computers. Van wasmachines tot rolstoelen, van vrachtautos tot vliegtuigen, van bibliotheek tot bushalte, van elektronische agenda tot digitale leeromgeving, overal vinden we technologische ondersteuning die niet meer zonder computers kan. Zelfs gebouwen worden dankzij computers steeds slimmer, onder andere door hoogwaardige beveiliging, voortdurende bewaking van het interne klimaat, automatische aanpassing van sfeerverlichting, etcetera.

De laatste 80 jaar heeft voor een belangrijk deel in het teken gestaan van de verdere ontwikkeling en uitbouw van de industriële samenleving; het industriële tijdperk. Als samenleving hebben we inmiddels de eerste schreden gezet in een nieuw tijdperk; het digitale tijdperk. Let wel, we noemen dit nieuwe tijdperk bewust niet het informatietijdperk.

In de laatste decennia van de vorige eeuw was regelmatig te horen en te lezen hoe we het industriële tijdperk achter ons zouden laten, en over zouden gaan naar het informatietijdperk [Cas00]. Naar mijn mening liepen die uitspraken zo'n honderd á honderdvijftig jaar achter de feiten aan.

Of, iets genuanceerder gezegd, ik ben van mening dat de naamgeving "het informatietijdperk" zo'n honderd á honderdvijftig jaar achter de feiten aanloopt. Mijns inziens is het informatietijdperk onlosmakelijk verbonden met de opkomst van het industriële tijdperk. Immers, tegelijkertijd met de industrialisering van de samenleving, kwamen er ook steeds meer informatie- en documentstromen op gang om diezelfde samenleving in goede banen te leiden. Natuurlijk waren er voor de industrialisering ook al informatiestromen, en heeft bijvoorbeeld de uitvinding van de boekdrukkunst deze stromen verder gestimuleerd. Echter, de informatiestromen zijn pas echt aangezwollen toen de industrialisering eenmaal op gang kwam. Gaandeweg zijn er steeds meer organisaties ontstaan, zoals financiële instellingen, overheden en regeluitvoerende instanties, die in essentie te zien zijn als grote informatieverwerkende fabrieken. Belastingaangiften, overschrijvingsformulieren, subsidieaanvragen en chartaal geld zijn allemaal voorbeelden van informatiedragers die door dergelijke fabrieken stromen.

In de tweede helft van de vorige eeuw waren de informatiestromen tot dermate grote proporties aangezwollen dat het bijna onmogelijk werd om ze nog handmatig te verwerken. Toen de computer zijn intrede deed, werd daar dan ook gretig gebruik van gemaakt om delen van deze informatiestromen per computer te verwerken.

Inmiddels kunnen we in de wereld om ons heen zien hoe computers steeds verder doordringen in alle delen van de samenleving. Het gebruik van computers is daarbij allang niet meer beperkt tot de informatieverwerkende fabrieken. Ook andere sectoren van de samenleving kunnen niet meer zonder. Het vervoer per trein, boot of auto, is inmiddels ook door allerlei vormen van gecomputeriseerde informatieverwerking omgeven. Zonder computers staan de vrachttreinen, vrachtschepen en vrachtauto's van "Nederland transportland" stil. En kan de moderne medische wereld nog zonder computer? We hoeven maar in een gemiddeld Westers ziekenhuis te gaan kijken om te beseffen dat men ook daar niet meer zonder kan.

De industrialisering is begonnen met het mechaniseren van de aandrijving. Eerst per stoommachine, toen per brandstofmotor en later per elektromotor. Inmiddels heeft deze mechanisering zelfs zijn weg gevonden naar onze tandenborstels. Het digitale tijdperk is feitelijk begonnen toen de verwerking van informatie werd gemechaniseerd door de inzet van computers. Het is een kwestie van tijd voordat ook de computer onze tandenborstel bereikt.

Een essentiële eigenschap die alle hedendaagse computers gemeen hebben is dat ze informatie verwerken in digitale vorm. Feiten, documenten, muziek, foto's, films, allemaal worden ze in digitale vorm door computers verwerkt. Zoals het stenen tijdperk vernoemd is naar de introductie van stenen gereedschappen, en het bronzen tijdperk naar het gebruik van bronzen gereedschappen, kunnen we het aankomende tijdperk het beste betitelen als het digitale tijdperk. Diverse auteurs die proberen een inschatting te maken van de mogelijke invloed van computertechnologie op mensen, maatschappij en organisaties, gebruiken daarbij dan ook metaforen zoals: Being Digital [Neg96], Digital Economy [Tap96] en Digital Places [Hor00]. Ook de Nederlandse overheid spreekt in haar toekomstbespiegelingen over "Digitale Delta" en "Digitale Economie". Men gebruikt zelfs al enige tijd het begrip de "Digitale Overheid", terwijl het Ministerie van Binnenlandse Zaken zich reeds waagt aan de "grondrechten in het digitale tijdperk". We kunnen dus stellen dat het leven steeds meer een digitaal leven wordt waarin informatie steeds meer in digitale vorm wordt verwerkt. In die zin lijken we ons dan ook te bevinden in de overgangsfase van het industriële tijdperk naar het digitale tijdperk. Enablers van het digitale tijdperk

1.2 Enablers of the digital era

Alvorens nader in te gaan op de rol van informatiesystemen (en hun ontwikkeling) in het digitale tijdperk, is het nuttig eerst kort stil te staan bij de belangrijkste enablers van het digitale tijdperk. Er zijn hierbij een drietal belangrijke voorwaardenscheppende factoren te onderkennen, twee van technologische en één van financiële aard.

Ten eerste worden computers steeds verder geminiaturiseerd. Dit maakt het mede mogelijk dat computers steeds makkelijker zijn te integreren in bestaande vormen van technologie. We hebben een ontwikkeling kunnen zien van zaalvullende mainframes, via PCs, tot alomtegenwoordige computers zoals we die kunnen terugvinden in onze mobiele telefoons, elektronische agendas, autonavigatiesystemen, wasmachines, bankpassen, etcetera.

Ten tweede worden computers steeds meer met elkaar verbonden. Hier hebben we een ontwikkeling kunnen zien van alleenstaande mainframes die middels ad-hoc verbindingen met elkaar werden verbonden, via het ontstaan van het Internet als structureel en robuust verbindingsmedium tussen diezelfde mainframes, tot aan de hedendaagse situatie waarin inmiddels naast de PC op het werk, de PC thuis en zelfs telefoons, DVD spelers, pacemakers, elektronische agenda's en spelcomputers via het Internet met elkaar verbonden zijn.

Tenslotte wordt de beschikbare rekenkracht en opslagcapaciteit steeds goedkoper. Let op: ik stel dat rekenkracht en opslagcapaciteit goedkoper wordt. Ik stel niet dat alle computergebaseerde apparatuur die wij kopen goedkoper wordt. PC's zijn, bijvoorbeeld, de afgelopen jaren niet dramatisch in prijs gedaald. De rekenkracht en opslagcapaciteit die je per Euro krijgt is echter wel fors toegenomen.

Een extreem, maar wel tot de verbeelding sprekend voorbeeld van de miniaturisering en de toenemende interconnectiviteit, is het zogenaamde "smart dust" [Koe03]. Het gaat hierbij om heel kleine computertjes met ingebouwde communicatiemiddelen, die daadwerkelijk kunnen zweven in de lucht. Omdat deze computertjes slechts bestaan uit één chip zijn ze ook nog eens erg goedkoop te produceren. Naast het onvermijdelijke militaire gebruik zijn mogelijke toepassingen: het detecteren of traceren van luchtvervuiling, het bewaken van verkeersstromen en het verkennen van planeten. Wat de uiteindelijke invloed van dergelijke technologie op de maatschappij zal zijn laat zich vooralsnog raden, maar "smart dust" is wel een mooie illustratie van de drie zojuist geschetste enablers van het digitale tijdperk.

Het zijn de miniaturisering, de toenemende interconnectiviteit, en de dalende prijs van rekenkracht, die het mogelijk maken dat computers daadwerkelijk doordringen tot in elke uithoek van onze samenleving. Nog even en dan herinnert mijn tandenborstel me er aan weer eens een afspraak te maken met de tandarts.

De enablers van het digitale tijdperk bieden legio mogelijkheden voor nieuwe toepassingen van gecomputeriseerde informatieverwerking. Dit betekent ook dat computers een toenemende invloed zullen hebben op de manier waarop we met zijn allen leven en werken. Diverse auteurs hebben zich gewaagd aan voorspellingen van deze invloed. Zie bijvoorbeeld [TC93, Kee91, Neg96].

Een kritische vraag die we daarbij moeten stellen blijft uiteraard of deze ontwikkelingen in alle gevallen leiden tot een nuttige of wenselijke toevoeging aan de samenleving. Zit de moderne mens, bijvoorbeeld, nu echt wel te wachten op een tandenborstel die je herinnert aan afspraken met je tandarts? Hoeveel mensen maken er inmiddels echt, vrijwillig, gebruik van de chipknip? Zijn de baten van gecomputeriseerde informatieverwerking werkelijk hoger dan de kosten? In het stellen en beantwoorden van dergelijke vragen ligt de basis voor het vakgebied "information systems engineering".

In the digital age, the role of information becomes paramount. This puts a considerable emphasis on the role of "systems" that handle this information, i.e. *information systems*. As a result, the effort of creating these information systems becomes more and more crucial in modern day society.

De introductie van de computer heeft in eerste instantie geleid tot software engineering. Naarmate dit gebied beter begrepen werd, en computers meer en meer geïntegreerd raakten in de samenleving, ontstond echter een steeds grotere behoefte aan een dieper inzicht in de relatie tussen computers en de context waarin deze gebruikt worden. Zo ontstond, aan het begin van het digitale tijdperk, het vakgebied "information system engineering". Het zijn, naar mijn mening, met name de information system engineers die ervoor dient te waken dat de digitalisering in

goede banen wordt geleid. Zij zijn de vormgevers, architecten en planologen van het digitale tijdperk.

1.3 The information systems area

Many different terminologies are used in discussing different aspects from the information systems area. In this introduction, we start by first defining a basic terminology which we will use, and refine, throughout this text-book. In doing so, we will draw from three main sources. We will (mainly) draw terminology pertaining to:

- architectures from the *IEEE recommended practice for software intensive systems* [IEE00],
- systems and information systems from the *framework of information system concepts* [FVSV+98],
- development processes from the *information services procurement library* [FV99, Pro01].

1.3.1 Information systems

As stated in [FVSV+98], "*information systems*" concerns the use of "*information*" by individual or groups of people in organizations, in particular through computer-based systems. In line with [FVSV+98], we use the term "*organization*" here, and throughout this textbook, in the most general sense. Not only large companies are meant. One-man companies, profit- and non-profit-oriented organizations, clusters of companies interacting with each other, even the community of all Internet users and similar communities, may all be considered organizations.

The concept of information system can roughly be defined as that aspect of an organization 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 computerized at all. A filing cabinet used to store and retrieve several dossiers is, in essence, an information system. The kind of information systems we are interested in, however, are indeed presumed to have some computerized core parts.

What we may perceive to be an information system, may 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:

- 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. [FVSV+98]) to be interpreted in at least three different ways:

- As a technical system, implemented with computer and telecommunications technology.

- As a social system, such as an organization, in connection with its information processing needs.
- As a conceptual system (i.e. an abstraction of either of the above).

A more precise definition (based on [FVSV+98]) of the way we view the concept of information system system is:

Information system – A sub-system of an organizational system, comprising the conception of how the communication and information-oriented aspects of an organization 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 organizational system.

This definition refers in its term refers to the concept of organizational system, which is essentially a systemic view on an organization:

Organizational system – A special kind of system, being normally active and open, and comprising the conception of how an organization 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.

Using the definition of information system, we may specialize this to its computerized parts as follows:

Computerized information system – A sub-system of an information system, whereby all activities within that sub-system are performed by one or several computer(s).

In Chapter 2, we will provide more context to these definitions, making them more precise.

1.3.2 Information systems as work systems

Information systems are systems! To be more precise, 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 [FVSV+98] 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.

The list is, obviously, sheer endless. When we discuss information systems and their design, we consider it to be worthwhile to first look at *active* systems from a more general perspective, and then zoom in on information system specific properties when/if needed.

In civil and military architecture it has become an accepted practice to copy patterns from constructions in nature and use them in our own constructions. In line with this practice, it makes perfect sense to see if we can use patterns and properties of other systems, such as systems that

occur in nature, in the design of information systems. For instance, the way in which the development and design of software agents is approached, draws more and more on the way biological organisms interact and grow [Ode00a, Ode00b, WJK00]. In the field of complex-adaptive systems [HM95], even more inspiration may be gained on the development of information systems that are better equipped to deal with complexity and evolution of their environment.

Rechtin and Maier in [Rec91, MR02], also subscribe to the point of view that it is worthwhile to look at development of systems in general as it allows the development and evolution of specific classes of systems to benefit from each other's insights. In these observations lies our motivation to, whenever possible, to first look at systems in general before zooming in on information systems in particular.

In [Alt99, Alt02] 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 [Alt99].*

where *information systems* are to be regarded as special classes of work systems. In this text-book, we will indeed take the approach that information systems are work systems. Note, however, that in the current version of this text-book we still do so rather implicitly. In future versions of this text-book (and the DAVINCI Series in general) we will more explicitly work with the idea that we have the following hierarchy of systems:

1. Systems in general.
2. Subclass of systems: Open active systems.
3. Subclass of open active systems: Work systems.
4. Subclass of work systems: Organisational systems (i.e. organisations "as we know them").
5. Subclass of work systems and a sub-system of organisational systems: Information systems.
6. Subclass of work systems and a sub-system of information systems: Computerised information systems.

In adopting work systems as the common denominator of the kinds of systems considered in the DAVINCI Series, we will also slightly modify the original definition of work systems to:

> *A work system is an* open active *system in which actors perform processes using information, technologies, and other resources to produce products and/or services for internal or external customers.*

We purposely generalize "human participants and/or machines" to the notion of actors, in order to abstract from the fact wether these actors are of a biological, mechanical, chemical, electronical, or whichever, means. Actors are presumed to perform *activities* (work!) in order to achieve some *purpose*.

1.3.3 Information systems engineering

Most information systems, in particular computerized ones, do not appear out of the blue. They need to be developed using some development process. We view the development of active systems as involving four processes:

Definition process – A process aiming to identify all requirements that should be met by the system and the system description.

In literature this process may also be referred to as requirements engineering.

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.

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.

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

In these definitions, the term system means to refer to either an organizational system, information system or computerized information system. The definitions of the construction process and the installation process are conform the definitions used in [FV99]. Conform [FV99], the definition process and definition process are collectively referred to as the description process. However, in the context of this text-book, a clear distinction between the definition of the future system in terms of its requirements, and the actual system design is indeed needed. The combination of the construction and the installation process may also be referred to as the realization process.

The concept of system engineering may now be defined as:

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

This can be specialized to information systems as:

Information system engineering – A system engineering process pertaining to the creation or change of information systems.

Note that, the processes involved in (information) system engineering in no way need to be executed linearly. Most practical situations require a non-linear execution of these processes, e.g. an evolutionary or incremental approach. In this text-book we are mainly concerned with the definition and design processes.

1.4 Challenges for information system engineering

De overgang naar het digitale tijdperk brengt een aantal uitdagingen met zich mee voor de ontwikkeling van (gecomputeriseerde) informatiesystemen. Hieronder bespreken we een vijftal belangrijke uitdagingen.

1.4.1 Ambient technology

De eerste uitdaging betreft de verwevenheid van de producten van de informatietechnologie met ons dagelijks leven en werken. Als gevolg van de toenemende verwevenheid zullen de gevolgen van eventuele problemen met diezelfde informatietechnologie steeds groter worden. Tegelijkertijd maakt deze verwevenheid het moeilijk om, bij het ontwikkelen van grootschalige informatiesystemen, precies af te bakenen met welke andere systemen en belanghebbenden er rekening gehouden dient te worden. Traditioneel viel een informatiesysteem binnen de muren van één organisatieonderdeel van beperkte omvang. Bijvoorbeeld een afdeling of een filiaal. De grenzen van zo'n organisatieonderdeel vormden als het ware een natuurlijke begrenzing van het informatiesysteem. Dit is echter in veel situaties allang niet meer het geval. Op steeds meer plaatsen zien we informatiesystemen ontstaan die geen natuurlijke grenzen meer kennen. Het ultieme voorbeeld daarvan is het World-Wide-Web. Probeer maar eens van dat systeem de grenzen eenduidig af te bakenen. Maar ook op andere plekken zien we dit probleem terug. Denk eens aan de

informatievoorziening binnen éń productieketen, zoals de productieketen voor auto's. Het zou voor de aanstaande koper van een auto erg prettig zijn als hij tot op een paar dagen nauwkeurig kan weten wanneer zijn aanstaande heilige koe opgehaald kan worden. Theoretisch kan dit berekend worden op basis van de voorraden en doorlooptijden in de hele productieketen. Maar, waar moeten we beginnen om een dergelijk informatiesysteem af te bakenen? Een nachtmerrie voor een projectleider; een uitdaging voor een Informatiekundige.

1.4.2 Pluriformity of stakes

De tweede uitdaging heeft betrekking op de pluriformiteit van de belangen en de belanghebbenden van de systemen die ontwikkeld worden. Denk hierbij maar eens aan de eerdergenoemde organisatorische, menselijke, informationele en technologische aspecten die voor informatiesystemen onderkend kunnen worden. Informatiesysteemontwikkeling vindt in de praktijk plaats in situaties waarin sprake is van een ruime schakering aan belanghebbenden met vaak tegenstrijdige belangen. Neem als voorbeeld het OV chipkaart project . De hoofddoelstelling van dit project is de introductie van de chipkaart als vervoerbewijs bij het OV met als einddoel n kaart die overal in het land bruikbaar is voor de trein, metro, tram, bus en boot. Het zal duidelijk zijn dat de belangen van de diverse vervoerders, de overheid, de leveranciers van de benodigde infrastructuur en, oh ja, de reizigers, nogal sterk uiteen zullen lopen.

Bij het ontwikkelen van informatiesystemen kunnen en mogen we onze ogen niet sluiten voor deze pluriforme realiteit. De theorieën, methoden en technieken die we hiervoor gebruiken moeten dus plek bieden voor deze "dans der belangen".

De pluriformiteit van de belanghebbenden en hun belangen maakt ook dat de broodnodige afstemming tussen de eerdergenoemde organisatorische, menselijke, informationele en technologische aspecten van informatiesystemen van essentieel belang is. Hierbij moeten we ons er terdege van bewust zijn dat deze afstemming meer vergt dan het "aan elkaar praten" van een aantal aspectspecifieke modellen. Het gaat uiteindelijk om de afstemming van denkwerelden, belangen, motivaties, en de daadwerkelijke integratie van modellen.

1.4.3 Intangiable systems

De derde uitdaging wordt gevormd door de ongrijpbaarheid van informatiesystemen. Bij het ontwerpen van "iets" is het prettig als de belanghebbenden zich goed kunnen inleven in hoe het resultaat er uit zou kunnen zien. Zonder dat inlevingsvermogen wordt het vaak moeilijk voor belanghebbenden om zich een voorstelling te maken van de mogelijke invloed van het eindresultaat op hun belangen. Daar komt men dan vaak pas achteraf achter, met alle gevolgen van dien. Zolang een ontwerp slechts op de tekentafel bestaat blijft het voor veel belanghebbenden te abstract; ongrijpbaar.

Bij het ontwerpen van artefacten in de fysieke wereld, zoals een auto of een huis, zullen belanghebbenden zich van tevoren al een voorstelling kunnen maken van het te ontwerpen object. Op basis van analogieën met reeds bestaande fysieke objecten kan men zich voorstellen hoe het eindresultaat er uit zou kunnen zien. In die situaties wordt de ongrijpbaarheid dus wat verzacht. Dit is bij het ontwerpen van informatiesystemen veel minder het geval. Gecomputeriseerde informatiesystemen zijn moeilijk in te beelden dingen. Het enige fysieke dat aan hun bestaan raakt zijn de kasten waarin de computer hardware zich bevindt, de beeldschermen, en eventuele afdrukken op papier. Het informatiesysteem zelf blijft ongrijpbaar. Er is zelfs niet éń specifieke kast aan te wijzen waarin het informatiesysteem zich bevindt.

1.4.4 Evolution is a constant

De vierde uitdaging is de veranderlijkheid van de socio-economische en technologische context waarin informatiesystemen worden ontwikkeld. Evolutie is een constante. Die schijnbare tegenstelling is een bondige omschrijving van de condities waaronder veel organisaties tegenwoordig moeten opereren.

Informatietechnologie ontwikkelt zich snel. In de context van het World-Wide-Web wordt er wel eens gekscherend gesproken over "web-jaren" , als waren het hondenjaren. De ontwikkelingen op het World-Wide-Web lijken sneller te gaan dan in het ?echte? leven. Het uiteenspatten van de e-commerce zeepbel lijkt de duur van web-jaren weer wat meer in overeenstemming gebracht te hebben met de realiteit. Desondanks is de verwachting dat de informatietechnologische ontwikkelingen zich in hoog tempo zullen blijven voortzetten.

Het zijn echter zeker niet alleen de ontwikkelingen van de informatietechnologie die evolutie tot een constante maken. De liberalisering van markten, het verminderen van protectionisme, de privatisering van staatsbedrijven, de toenemende wereldwijde concurrentie en grensoverschrijdende bedrijfsfusies zijn allemaal aspecten die bijdragen aan de dynamiek van de huidige socio-economische context. Het is het samenspel tussen de socio-economische en de informatietechnologische ontwikkelingen die evolutie tot een constante maken. In deze dynamische omgeving moet de Informatiekunde helder krijgen wat de behoeften en belangen zijn ten aanzien van te ontwikkelen informatiesystemen. Het beantwoorden van die vraag verwordt al snel tot ?het schieten op een bewegend doelwit?.

The prevailing conditions under which most organizations currently operate have a tendency to evolve constantly. Reduced protectionism, de-monopolization of markets, deregulation of international trade, privatisation of state owned companies, increased global competition, cross-border merges, the emergence of new trade blocks, the introduction of common currencies, all contribute toward this increasingly dynamic business environment. Developments that are fuelled even more by the advances of eCommerce, Networked Business, Virtual Enterprises, etc. To improve their chances for survival, organizations need the ability to quickly adapt themselves to such socio-economic developments.

Organizations make use of (largely computerized) information systems to provide in their information processing needs. When an organization evolves, these information systems should be able to co-evolve in a natural way [Pro94]. In practice, this has proven to be a difficult task, in particular where it concerns the computerized (parts of the) information systems, i.e. information technology (information technology). Studies, such as the ones reported in [LS80, BP88, NP90] have shown that a large part of the costs associated to computerized information systems are spent on such modifications.

Ideally, information technology should empower an organization with the ability to go out and seek new challenges. However, one of the current dilemmas of information technology seems to be that in most cases it smothers an organization's ability to change rather than supporting it. While it is quite reasonable to say that advanced computerized information systems should lead to revolutionary improvements in the flexibility and effectiveness of organizations, organizations still find themselves anchored to their pre-existing information systems. Quite often, these systems are the embodiment of the prevailing cultures and structures of the organization's past. These systems tend to have an almost tangible monolithic nature that would be a feast to software archologists.

Example 1.4.1

Two concrete examples of such application domains with rapidly changing information needs are:

Taxes In most nations the (income) tax laws change quite often as they are used both to manage the economy, and to finance government policy.

Software firms developing and maintaining software for the calculation of, say, income taxes for company employees, have to change their software each time the government changes income tax. The change in information needs is caused by the changed laws.

Recent examples of changes in tax laws can be found in many of the nations of the European Community, due to the tax harmonizations brought about by the unification process.

Insurance Most insurance companies change the rules by which policy prices are calculated regularly. These changes are usually intended to improve the competitiveness of the policy pricing, for instance, the no-claims bonus for not claiming damages in the case of car insurances. As a result, the software for calculating the policy prices has to be changed to cope with any extra information required (history of damages claimed), as well as new formulas to perform the calculation. In this case, the changes in the information need are caused by marketing arguments.

An organization can deal with changes in their environment in a variety of ways. While some may try and continue their business *as usual*, others may choose to embrace the new developments and try to exploit their potential to their fullest. Neither approach is a guaranteed way to success or failure. Embracing new developments too early may lead to organizational chaos and decline, while waiting too long may result in missed business opportunities. The role of information technology can be characterized as a position between two undesirable extremes. On the one extreme, information technology can completely restrain organizational change. In the other extreme, it can (try to) bring about organizational changes in a pace which is far too high. The latter situation can be compared to putting a Ferrari engine in a VW Beetle, while the driver is used to easing along at maximum speeds of 80 to 100 km/h. In the first extreme, information technology appears to smother any organizational change. This seems to be one of the dilemmas of information technology. In the second extreme, information technology will drive an organization to a pace of change that goes beyond the speed its organizational fabrics can manage. Some organizations are just not ready to cope with the profound changes brought about by a technology push. For example, organizations that have only just reached a stage at which they are confident with the use of some basic information technology to automate the processing of orders might simply not survive a quick move towards electronic commerce.

Already in [Kee91] and [TC93], an elaborate discussion can be found on the changes in context and culture that are occurring inside organizations as well as in their environments as a result of different socio-economic changes in combination with technological developments in information technology. Tapscott [TC93] proposes an architectural approach as a solution to make the needed changes to the organizational structure and in particular information technology. These new demands on information technology in the new and rapidly evolving world, can be summed up by quoting [Tap96]:

> *In the past an architecture was really the design of a system that had been created to meet specific application needs. In the new business environment, organizations have little idea what their application needs will be in two, let alone five or ten years. Consequently, we need architectures that can enable the exploitation of unforeseen opportunities and meet unpredictable needs.*

Development of information systems in such rapidly evolving contexts becomes like shooting at a moving target [PW95a, PW94, PW95b]. This requires us to look at organizations and their information systems as evolving systems [Pro94] that are in a constant state of co-evolution.

Ideally, business strategists should be able to focus solely on the development of a business, while information technology plays the role of a catalyst. Tapscott [TC93] argues that organizations move between different levels of organizational development, improving their ability to cope with changes/evolution in their environment. information technology should act as one of the essential enablers of this process. In Figure 1.1, taken from [TC93], these levels have been

depicted. By redesigning their business processes, organizations will be able move to a situation in which teams can perform better. This development leads to high-performance business teams, where the focus is on the use of information technology to enable teams to perform business functions. This requires a shift away from a hierarchical view on organizations to a more team based view. The next shift involves the integration of the business teams to an integrated whole, leading to an integrated organization. The role of information technology in these cases, is increasingly one of being an enabler; from cost center to profit center. By linking their systems to other organizations, in particular customers and suppliers, an organization can finalize the paradigm shift, and become an extended enterprise.

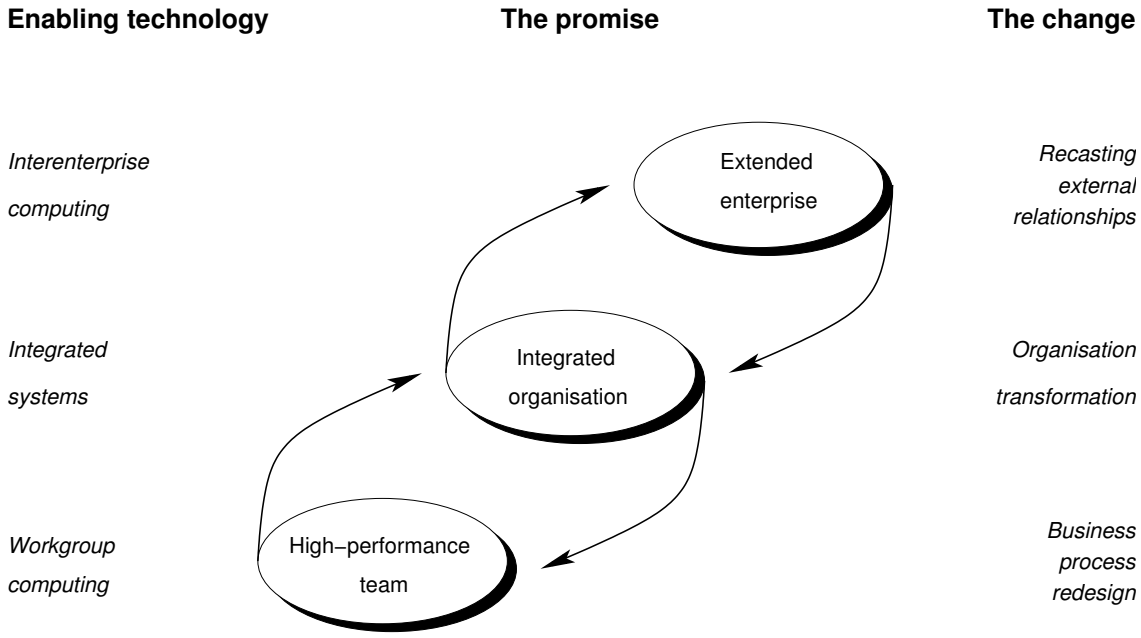


Figure 1.1: The enabling effect of information technology

All these shifts and developments pose new requirements on the information technology function. These shifts, however, will not come from information technology alone. As Michael Hammer argued in his seminal paper: *Re-engineering work: don't automate, obliterate* [Ham90], improving the flexibility and efficiency of business processes is more than just using information technology to make it go faster. information technology is only part of the answer, it should not be looked upon as the sole bringer of solutions, but rather as an enabler. Investments in a re-engineering of the business and better business-IT alignment will be needed just as well. However, organizational change should be business-driven, and not (solely) information technology-driven. The outcry for information technology that enables organizational change rather than inhibits it is clearly louder than ever. information technology, and associated application development, should therefore be driven by the needs of the business.

1.4.5 Complexity; the gravitational force of software construction

In architectural design of man-made constructions, a pivotal role is played by the struggle with *gravity*. In the software architectures for computerized information systems, this role seems to be played by the struggle with *complexity*. We can all see around us how the software systems we develop, start to break down under their own weight. The complexity of these systems has already reached a point where no single person is able to grasp all details of a systems working.

The increase of complexity is fuelled by our thirst for ever more functionality. We want our computerized information systems to do more and more work for us. We require them to increase their scopes from a single unified organization to cover entire coalitions of networked organizations, leading all sorts of interoperability problems. The constant pressure to evolve, as discussed above, makes this situation even more serious, leading to unreliable software, high maintenance costs, maintenance backlogs, etc.

The seriousness of software complexity has been reported in several sources. For example, in [Coc01], the following can be read:

> *What's the most important problem in computer science?*
>
> *Languages, tools, programmers?*
>
> *Well, according to a growing number of researchers and computer users, it's software complexity.* "We've known about this problem for 40 years," *says Alfred Spector, vice-president at IBM Research.*
>
> "This is probably the number one problem...It can't go on."

Brooks, in his Mythical Man-Month review [Bro95], mentions three distinctive concerns of software engineering, of which the last one reads: *How to maintain intellectual control over* **complexity** *in large doses.* Brooks, furthermore, writes:

> *The tar pit of software engineering will continue to be sticky for a long time to come. One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate of man's handiworks.*

1.5 Architecture-driven information systems engineering

1.5.1 Architecture

During the last decade, software architecture has received an increasing amount of attention in the software engineering community; both from research and from industry. The rationale behind this interest is that software architecture provides a number of important benefits [BCK98]:

- It is a vehicle for communication among stakeholders. A software architecture, often depicted graphically, can be communicated with end users, the client, designers, and so on.

  By developing scenarios of anticipated use, relevant quality aspects can be analyzed and trade-offs can be discussed with various stakeholders.
- It captures early design decisions, both functional aspects as well as quality aspects. In a software architecture, the global structure of the system has been decided upon, through the explicit assignment of functionality to components of the architecture.

  These early design decisions are important since their ramifications are felt in all subsequent phases. It is therefore paramount to assess their quality at the earliest possible moment. By evaluating the architecture, a first and global insight into important quality aspects can be obtained. The global structure decided upon at this stage also structures development: the work-breakdown structure may be based on the decomposition chosen at this stage, testing may be organized around this same decomposition, and so on.

These advantages are not only limited to software (as it may be found in computerized information systems), but equally well relate to most types of systems.

Several definitions of the concept of architecture, in the context of systems, exist. According to [Mer03], architecture is:

1 the art or science of building; *specifically*: the art or practice of designing and building structures and especially habitable ones,

2a formation or construction as or as if as the result of conscious act,

2b a unifying or coherent form or structure,

3 architectural product or work,

4 a method or style of building,

5 the manner in which the components of a computer or computer system are organized and integrated.

In the context of software-intensive systems, an IEEE working group [IEE00] has provided a definition of architecture. The resulting definition is, indeed, in line with the general definition of architecture (in particular interpretations 1 and 5), but specializes it to software-intensive systems. As (computerized information systems are generally software-intensive systems, we shall use this definition throughout this text-book:

Architecture – A model of which the system description, the so-called architectural description, is used during system engineering to:

- express the fundamental organization 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.

As an architecture provides guidelines for the (detailed) design and evolution of a system, it is a powerful *means* to approach evolution and complexity of organizations, their context and their information systems. Note that *approach* not necessarily mean *control* of every detail.

Architectures are usually expressed in terms of architectural descriptions, essentially design descriptions pertaining to a systems architecture. In [IEE00] the following potential uses of architectural descriptions are identified:

- Expression of the system and its (potential) evolution.
- Analysis of alternative architectures.
- Business planning for transition from a legacy architecture to a new architecture.
- Communications among organizations involved in the development, production, fielding, operation, and maintenance of a system.
- Communications between acquirers and developers as a part of contract negotiations
- Criteria for certifying conformance of implementations to the architecture.
- Development and maintenance documentation, including material for reuse repositories and training material.
- Input to subsequent system design and development activities.
- Input to system generation and analysis tools.
- Operational and infrastructure support; configuration management and repair; redesign and maintenance of systems, subsystems, and components.
- Planning and budget support.
- Preparation of acquisition documents (e.g., requests for proposal and statements of work).
- Review, analysis, and evaluation of the system across the life cycle.
- Specification for a group of systems sharing a common set of features, (e.g., product lines).

1.5.2 Alignment

In information systems development, another pivotal role is played by alignment. In literature this is usually referred to as business-IT alignment or as strategic alignment [TC93, Kee91, PB89, HV93]. According to [Mer03], alignment is:

1 the act of aligning or state of being aligned; *especially*: the proper positioning or state of adjustment of parts (as of a mechanical or electronic device) in relation to each other

2a a forming in line

2b the line thus formed

3 the ground plan (as of a railroad or highway) in distinction from the profile

4 an arrangement of groups or forces in relation to one another <new *alignments* within the political party>

The importance of a good alignment between business and information technology is discussed by several authors. For example, Tapscott & Caston [TC93], [Boa99] and Keen [Kee91], and they are certainly not the first in doing this, provide extensive motivations. In [PB89], Parker and Benson already discussed the need for strategic alignment between business and information technology strategies. They argued that information technology planning and strategic considerations are part of a circular process as depicted in Figure 1.2. In this process, a distinction is made between the business domain on the one side, and the technology domain on the other side. Business planning drives how an enterprise will be organized, which should on its turn drive the technology planning to support the business. Technology planning leads to the discovery of further opportunities for future uses of technology, which will influence further business planning and strategy.

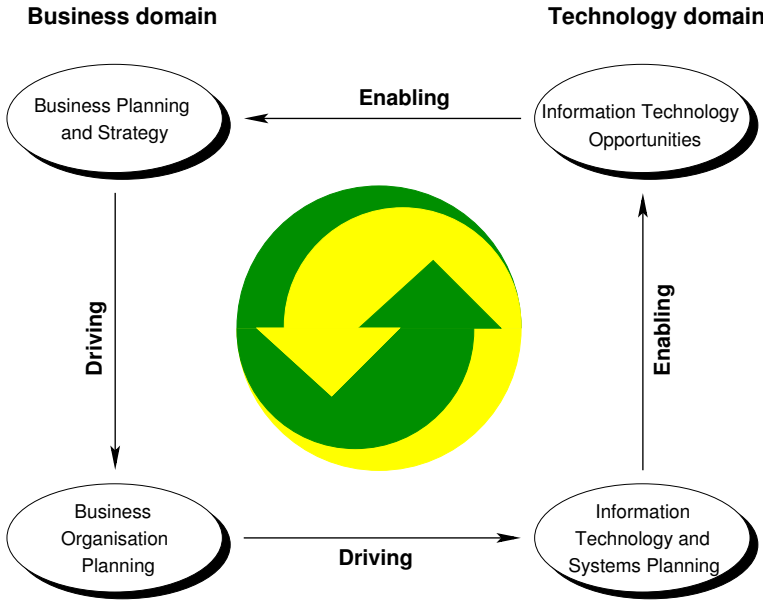


Figure 1.2: Strategic alignment cycle

Parker and Benson also recognized the fact that this cyclic process may not work on an organization-wide scale. Organizations usually do not operate in a way that supports a 'monolithic' view of their information systems. However, they also argue that these cycles can be specialized to a specific line-of-business, or a specific business unit. In other words, the cycle of Figure 1.2 can be applied to smaller, more focused, scopes within an organization. Scopes that may range from an entire value-chain, via business-units to teams and individual work places.

The views of Parker and Benson were refined further by Henderson and Venkatraman [HV93]. On the importance of alignment between business and information technology *strategies*, they argue:

> *We argue that the inability to realize value from information technology investments is, in part, due to the lack of alignment between business and information technology strategies of organizations.*

They also conclude:

> *Strategic alignment is not an event, but a process of continuous adaptation and change.*

Alignment, however, does not only play a role at a strategic level. Proper alignment between information systems and their organizational (and human!) context is just as important at a tactical and operational level. For example, at a tactical level one would be concerned with the selection of a specific portfolio of information systems to be developed on the shorter term and how they align to the organizational activities & goals on the shorter term. At the operational level, one would be concerned with the development of a specific information system and its direct organizational and human environment.

1.5.3 Architecture-driven information system engineering

This textbook takes the view that the use of *architecture* enables the alignment between an information system and its organizational and human context. An alignment that should range from the strategic levels to operational levels. We therefore take the perspective that proper alignment is at the heart of what we define to be *architecture-driven information systems engineering*:

Architecture-driven-system-engineering – System engineering, using architecture as a means to

- guide & control the design and evolution of the system,
- evaluate & compare different information system alternatives,
- negotiate the concerns of the information system's stakeholders,

with the aim of optimising the alignment of the resulting information system to its relevant, technological, organisation and human context (systems).

Architecture-driven information system engineering – Information system engineering, using architecture as a means to

- guide & control the design and evolution of the information system,
- evaluate & compare different information system alternatives,
- negotiate the concerns of the information system's stakeholders,

with the aim of optimising the alignment of the resulting information system to its relevant, technological, organisation and human context (systems).

This definition is a further elaboration of the definitions provided in [Pro98, PBHJ00]. As mentioned before, this text-book mainly focuses on the definition and design processes as these are influenced the most by the use of architecture.

1.6 A fundamental approach

Modeling is at the very heart of the field of information systems engineering as well as organizational engineering. Any course on the engineering of information systems should therefore also provide a fundamental understanding of modeling. 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 domain, they are likely to do so with different models of this domain in 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. In this text-book we aim to provide a fundamental discusion on at least some of these issues.

In developing our understanding of modeling (information system) domains, we will discuss several modeling languages for different aspects of such systems. When discussing these modeling languages, we will also discuss their syntax and semantics from a formal (mathematical) perspective. In [HW92, Hof93, HP98] three major reasons for a formal approach to the syntax and semantics of modeling techniques are given.

Even though in literature it has often been emphasized that modeling languages should have a rigorous formal basis (see e.g. [Coh89, TP91, Spi88, Jon86, HL89], somehow this need for formality has not been generally acknowledged in the field of information systems engineering and organization engineering. This has contributed greatly to the appearance of the "*Methodology Jungle*", a term introduced in [Avi95]. In [Bub86] it is estimated that during the past years, hundreds if not thousands of information system development methods have been introduced. Most organizations and research groups have defined their own methods. The techniques advocated in these methods usually do not have a formal foundation. In some cases their syntax is defined, but attention is hardly ever paid to their formal semantics. The discussion of numerous examples, mostly with the use of pictures, is a popular style for the "definition" of new concepts and their behavior. This has led to *fuzzy* and *artificial* concepts in information systems development methods (see also [Bub86]).

1.7 Structuring the domain

The aim of this text-book is to discuss several aspects from the domain of architecture-driven information system engineering. Even though the aim of this book is *not* to define a specific method for architecture-driven information system engineering, it makes sense to use a methodological framework to structure the contents of this text-book.

1.7.1 Methodological framework

At the end of the eighties of the last century, there was a growing need to compare different information system development methods. For example, in [SWS89, WH90, HW92] different frameworks and strategies are discussed to position development methods. Approaches to chop-down the so-called *methodology jungle*.

The methodological framework we will employ, is based on the framework originally presented in [WH90] and [SWS89], and refined further, with a way of communicating, in [PW94, PW95b] and [Pro94]. In the resulting framework, as depicted in Figure 1.3, a modeling method is dissected into the following six aspects:

Way of thinking – Articulates the assumptions on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* [Sol83, WAA85], *underlying perspective* [Mat81] or *philosophy* [Avi95].

Way of modeling – Identifies the *core concepts* of the language that may be used to denote, analyse, visualise and/or animate system descriptions.

Way of communicating – describes how the abstract concepts from the way of modeling 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 modeling and the way of working, it matches the abstract concepts of the way of modeling to the pragmatic needs of the way of working.

Note that it may very well be the case that different modeling techniques are based on the same way of modeling, yet use different notations.

Way of working – Structures (parts of) the way in which a system is developed. 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 controlling – The managerial aspects of system development. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [Ken84, Sol88]).

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 computerized tool (see for instance [McC89]).

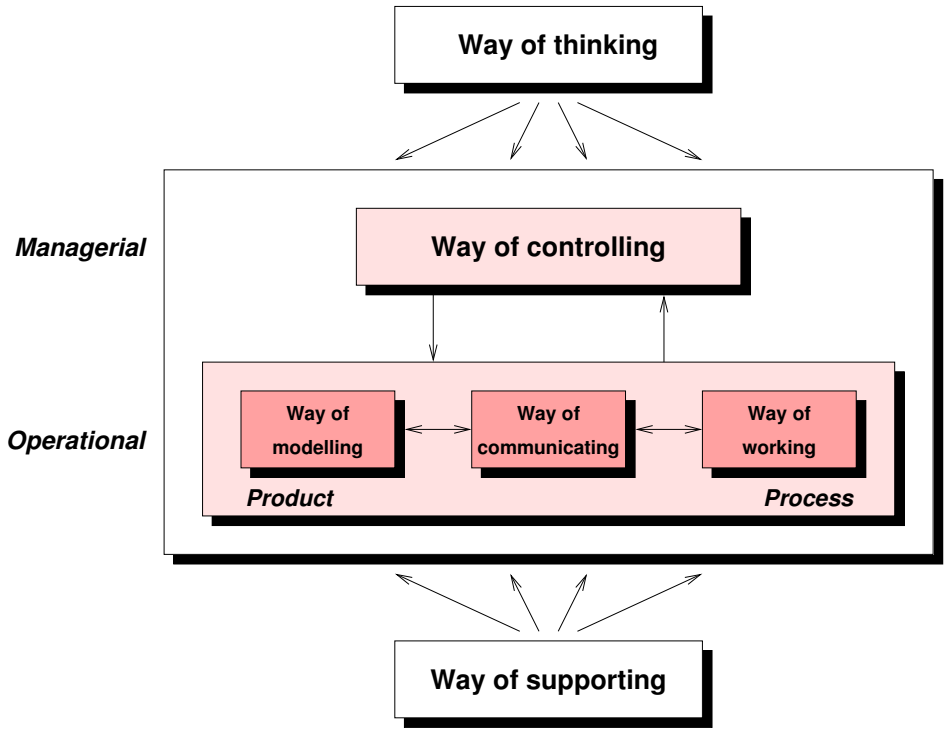


Figure 1.3: Aspects of a method

The arrows in Figure 1.3 should be interpreted as: if $x \rightarrow y$, then aspect x supports aspect y. The combination of a way of modeling and communicating is usually referred to as a "modeling technique".

Some methods may provide a very detailed way of working. For example, for the information modeling method as discussed in [Hal01], a very detailed way of working is presented. However:

A method should never become an excuse to stop thinking!

This text-book primarily focuses on a way of thinking, way of thinking and provides some aspects of a way of working and way of controlling for architecture-driven information system development.

For as far as this text-book portrays a complete *method* for the definition and design parts of architecture-driven development of information systems, its way of thinking is that the actual ways of modeling, working, controlling and support should be highly situational. In [WAA85], the authors state:

> *In a practical discipline, one must always distinguish between the* **ideal methodology** *as thought in text-books, and the realities of any situation which causes departure from the ideal in order to allow for the exigencies of the real world. Many design methodologies are prescriptive not only of what must be done but of the order in which it has to be done. In the real world, decisions are often made before all the facts have been gathered.*

In other words, let the reader be warned. The document you are currently reading is a text-book. A text-book that should inspire. It, however, will not provide all-inclusive answers that will fit all practical situations.

1.7.2 Structure of this text-book

This text-book is split into three themes:

- Information system modeling
- Information system engineering
- Architecture-driven information system engineering

Version: 04-04-05

Questions

1. What is an information system? Give some examples of an information system.
2. What is information system development? What does 'architecture-driven' add to this?
3. What are the consequences of an evolving environment on information systems?
4. Sometimes organisations use the introduction of IT as a way to force changes in organisations.

   Why is this likely to fail?
5. Why is complexity a challenge for information system development?

   What will happen if you add evolution to the equation?
6. Why is it important to use an 'architecture-driven' approach for the development of systems that are used by some organisation in achieving its goals?

   Which role is played by the environment of the organisation?

   What is the role of the possible evolution of the organisation and its environment?
7. What is the essence of architecture in an information systems context?
8. Explain, in your own words, the essence of alignment.
9. Why is it important to optimise the alignment of an information system to its context?

Recommended reading

[Neg96] N. Negroponte. *Being Digital*. Vintage Books, New York, New York, 1996. ISBN 0679762906

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

Chapter 1 in particular.

[Alt02] S. Alter. The work system method for understanding information systems and information system research. *Communications of the Association for Information Systems*, 9(9):90–104, 2002.
`http://cais.isworld.org/articles/default.asp?vol=9&art=6`

The work system method is a broadly applicable set of ideas that use the concept of "work system" as the focal point for understanding, analyzing, and improving systems in organizations, whether or not IT is involved. The premises underlying this method may be controversial in the IS community because they imply that the traditional jargon and concerns of IS practitioners and researchers address only part of the issues that should be covered and may discourage focusing on other core issues related to successful projects and systems.

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0632012168

This book offers an inspiring view on information system development. Even though the book may be regarded as dated, it should still provide ample inspiration to people who are new to the field of information system development.

The book presents a multi-view approach to the development of information systems, distinguishing five points of view – organisation's human activities, information, socio-technical, human-computer and technical.

[Ode00b] J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

This report discusses how the complex adaptive systems (or simply, complex systems) approach is applicable to developing multiagent applications. In particular, it discusses topics such as adaptation, emergence, "edge-of-chaos" phenomena, and the difference between agents and objects.

This makes it a useful introduction to the promise of using principles and patterns from the structure of other types of systems for the design of (computerized) information systems.

[Alt02] S. Alter. The work system method for understanding information systems and information system research. *Communications of the Association for Information Systems*, 9(9):90–104, 2002.
`http://cais.isworld.org/articles/default.asp?vol=9&art=6`

The work system method is a broadly applicable set of ideas that use the concept of "work system" as the focal point for understanding, analyzing, and improving systems in organizations, whether or not IT is involved. The premises underlying this method may be controversial in the IS community because they imply that the traditional jargon and concerns of IS practitioners and researchers address only part of the issues that should be covered and may discourage focusing on other core issues related to successful projects and systems.

Optional reading

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[Hor00] T.A. Horan. *Digital Places – Building our city of bits*. The Urban Land Institute (ULI), Washington DC, United States of America, 2000. ISBN 0874208459

[TC93] D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0070628572

[Kee91] P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372

[Tap96] D. Tapscott. *Digital Economy - Promise and peril in the age of networked intelligence*. McGraw-Hill, New York, New York, USA, 1996. ISBN 0070633428

[Boa99] B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[HV93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.

[PB89] M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.

Bibliography

[Alt99] S. Alter. A general, yet useful theory of information systems. *Communications of the Association for Information Systems*, 1(13), 1999.
`http://cais.isworld.org/articles/1-13/default.asp`

[Alt02] S. Alter. The work system method for understanding information systems and information system research. *Communications of the Association for Information Systems*, 9(9):90–104, 2002.
`http://cais.isworld.org/articles/default.asp?vol=9&art=6`

[Avi95] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, USA, 2nd edition, 1995. ISBN 0077092333

[BCK98] L. Bass, P.C. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Massachusetts, USA, 1998. ISBN 0201199300

[Boa99] B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376

[BP88] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *IEEE Transactions of Software Engineering*, 14(10):1462–1477, October 1988.

[Bro95] F. Brooks. *The Mythical Man-Month; anniversary edition*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201835959

[Bub86] J.A. Bubenko. Information System Methodologies - A Research View. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*, pages 289–318. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1986.

[Cas00] M. Castells. *The Information Age: Economy, Society and Culture*. Volume 1 – The Rise of the Network Society. Blackwell, Oxford, United Kingdom, EU, 2nd edition, 2000. ISBN 0631221409

[Coc01] S. Cochran. The rising cost of software complexity. *Dr. Dobb's Journal*, April 2001.

[Coh89] B. Cohen. Justification of formal methods for system specification. *Software Engineering Journal*, 4(1):26–35, January 1989.

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1558606726

[Ham90] M. Hammer. Re-engineering work: don't automate, obliterate. *Harvard Business Review*, 68(4):104–112, April 1990.

[HL89] I. van Horenbeek and J. Lewi. *Algebraic specifications in software engineering: an introduction*. Springer-Verlag, Berlin, Germany, 1989.

[HM95] J.H. Holland and H. Mimnaugh, editors. *Hidden Order : How Adaptation Builds Complexity*. Perseus Press, Cambridge, Massachusetts, 1995. ISBN 0201442302

[Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[Hor00] T.A. Horan. *Digital Places – Building our city of bits*. The Urban Land Institute (ULI), Washington DC, United States of America, 2000. ISBN 0874208459

[HP98] A.H.M. ter Hofstede and H.A. (Erik) Proper. How to Formalize It? Formalization Principles for Information Systems Development Methods. *Information and Software Technology*, 40(10):519–540, October 1998.

[HV93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.

[HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000. ISBN 0738125180
`http://www.ieee.org`

[Jon86] C.B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Kee91] P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372

[Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In T.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.

[Koe03] B.I. Koerner. What is smart dust, anyway? *Wired*, 11(6), June 2003. `http://www.wired.com/wired/archive/11.06/start.html?pg=10`

[LS80] B. Lientz and E. Swanson. *Software Maintenance Management – a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, Reading, Massachusetts, 1980. ISBN 0201042053

[Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. In Danish.

[McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003. `http://www.webster.com`

[MR02] M.W. Maier and R. Rechtin. *The Art of System Architecting*. CRC Press, Boca Raton, Florida, 2nd edition, 2002. ISBN 0849304407

[Neg96] N. Negroponte. *Being Digital*. Vintage Books, New York, New York, 1996. ISBN 0679762906

[NP90] J. Nosek and P. Palvia. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 3(2):157–174, 1990.

[Ode00a] J. Odell. Agents (part 1): Technology and usage. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[Ode00b] J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[PB89] M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.

[PBHJ00] H.A. (Erik) Proper, H. Bosma, S.J.B.A. Hoppenbrouwers, and R.D.T. Janssen. An Alignment Perspective on Architecture-driven Information Systems Engineering. In D.B.B. Rijsenbrij, editor, *Proceedings of the Second National Architecture Congres*, Amsterdam, The Netherlands, EU, November 2000.

[Pro94] H.A. (Erik) Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[Pro98] H.A. (Erik) Proper. Da Vinci – Architecture-Driven Business Solutions. Technical report, Origin, Amsterdam, The Netherlands, EU, Summer 1998.

[Pro01] H.A. (Erik) Proper, editor. *ISP for Large-scale Migrations*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU, 2001. ISBN 9076304882

[Pro03] H.A. (Erik) Proper. *Informatiekunde; Exacte vaagheid*. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU, November 2003. In Dutch. ISBN 9090172866

[PW94] H.A. (Erik) Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

[PW95a] H.A. (Erik) Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[PW95b] H.A. (Erik) Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0444867058

[Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[Spi88] J.M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge University Press, Cambridge, United Kingdom, EU, 1988.

[SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.

[Tap96] D. Tapscott. *Digital Economy - Promise and peril in the age of networked intelligence*. McGraw-Hill, New York, New York, USA, 1996. ISBN 0070633428

[TC93] D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0070628572

[TP91] T.H. Tse and L. Pong. An Examination of Requirements Specification Languages. *The Computer Journal*, 34(2):143–152, April 1991.

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0632012168

[WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag, Berlin, Germany, EU. ISBN 3540526250

[WJK00] M. Wooldridge, N.R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

Part I

Domain Modeling

Chapter 2

Work Systems

Version: 25-04-05

[1]In this chapter, we discuss our fundamental view on organizations, information systems and work systems. We will provide a definition of terms, which is based on a system theoretic [Ber01] foundation. In doing so, we also provide a fundamental discussion of the concepts of *system*, *organization* and *information*.

2.1 Exploring systems

Even though the notion of system is, in an IT context, often equated to "software system", the original sense of the word is much broader. The notion of system[2] 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*" [Iiv83] or just as: "*A set of objects with a set of links*" [Lan71]. 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 it may be found in [Mer03]:

> 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 that is 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 especially 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.

In the IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [IEE00] one can find a functionality-oriented perspective on systems:

[1]Parts of this chapter are based on work the author was doing with A.A. (Xander) Verrijn-Stuart on a revised edition on the FRISCO report [FVSV+98]. As Xander passed away unexpectedly, the revised edition was never finished.

[2]The term 'System' is derived from the Greek phrase 'Syn histanai' ($\sigma\upsilon\nu\ \acute{\iota}\sigma\tau\eta\nu\alpha\iota$): "to put together".

> 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 [FVSV+98] some examples are given of what we would, and would not, observe to be systems in our daily life:

Example 2.1.1

One can regard an organization or a bicycle as systems. Also a Hitchcock film recorded 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 raindrop.

However, when looking 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.

In [Rop99] it is argued that, strictly speaking, there exist three different interpretations of systems:

Structural – The structural interpretation is known best. According to this interpretation, a system includes a set of elements and a set of links between these elements. This interpretation complies with the ancient definition of the *holon* by Aristotle.

Functional – The functional interpretation views a system as an entity, sometimes called black box, which transforms inputs into outputs. Depending on specific internal states; the kind of transformation is called a function (in the descriptive meaning of the word).

Hierarchical – The structural interpretation turns into the hierarchical interpretation, if (some of) the constituting elements are regarded as subsystems. Concluding by analogy, the original system may be considered as a subsystem of a more extensive supersystem.

2.2 Observing systems

The aim of this section is to define the concept of system. In terms of this definition, we can then continue to define the concept of organization more fundamentally as well.

2.2.1 Subjectivity

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 a car 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 behavior 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 Faraday cage 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 a business enterprise, an institution or any other kind of organization as a system – an organizational system – we have a domain which is much more complicated than a car and driver. Furthermore, the number of possible views of an organization 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 expresses it, there must be a describer/observer who conceives or thinks about a part of the world as a system [Che81]. In other words, it is important, that there is a *viewer* who can see a purpose in regarding something "*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 something "*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 [FVSV+98].

2.2.2 Observing the universe

Let us start by considering what happens if some viewer observes 'the universe'. It is our assumption, based on the work of C.S. Peirce [Pei69a, Pei69b, Pei69c, Pei69d], that *viewers* perceive a *universe*, leading to a *perception* of this universe and then produce a *conception* of that part they deem relevant. 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.

Viewer – An actor perceiving and conceiving (part of) a domain.

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.

Conception – That what results, in the mind of a viewer, when they interpret a perception of a domain.

In general, people tend to think of the universe as consisting of elements. In [FVSV+98] this approach is indeed taken. In our view, presuming that the universe consists of a set of elements constitutes a subjective choice, which essentially depends on the viewer observing the universe. Nevertheless, taking this assumption has proven to be a sensible assumption, in particular in the context of systems. However, we do *not* presume the *universe* itself to consist of elements, but *rather* the *conception* of a universe. This makes the identification of elements relative to a viewer.

The conceptions harbored by a viewer are impossible to communicate and discuss with other viewers unless they are articulated somehow. In other words, a conception needs to be *described* somehow in terms of a description:[3]

Description – The result of a viewer denoting a conception, using some language to express themselves.

The resulting situation is illustrated in Figure 2.1. Descriptions may be:

- Formal or informal
- Complete or incomplete
- More refined/less refined

The underlying relationships between viewer, universe, conception and description can be expressed in terms of the so-called FRISCO tetrahedron [FVSV+98], as depicted in Figure 2.2[4].

[3] In [FVSV+98] the term *representation* is used rather than the term *description* as it is used here. We have chosen to favor the term *description*, as it is the term of choice of [IEE00].

[4] The original FRISCO tetrahedron uses 'domain' where we use 'universe'. This difference is due to the above, more refined, discussion on the subjectivity of viewing the universe as a set of elements.

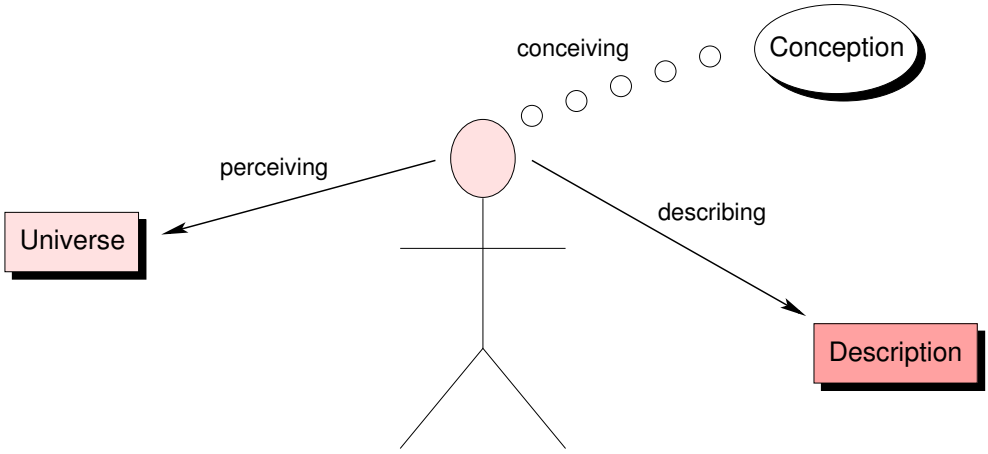


Figure 2.1: A viewer, having a conception of the universe, and describing this in terms of a description.

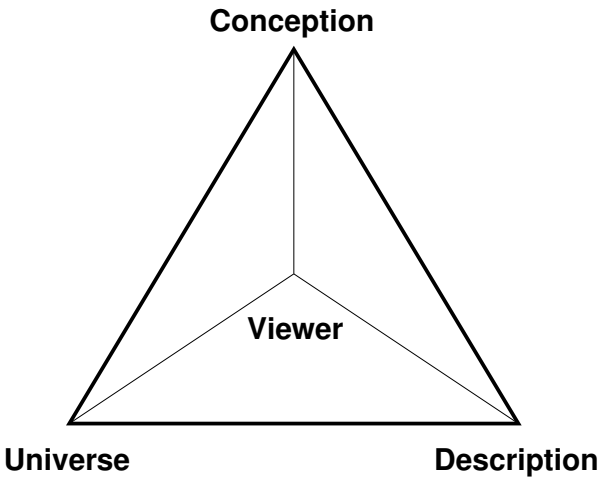


Figure 2.2: The (revised) FRISCO tetrahedron.

2.2.3 Conceptions

To express the above discussions more formally, let $\mathcal{UN}$[5] be the set of universes that may be observed, and let $\mathcal{VW}$[6] be the set of all possible viewers. Let furthermore, $\mathcal{EL}$ be the set of elements that may be part of conceptions. These basic sets should be disjoint:

[S1] $\mathcal{VW}, \mathcal{UN}$ and $\mathcal{EL}$ are mutually disjoint.

Let $\_ \models^{c} \_ \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL})$ be the links expressing which conception is held by which viewer. The fact that a viewer $v \in \mathcal{VW}$ harbors a conception $C \subseteq \wp(\mathcal{EL})$ for universe $U \in \mathcal{UN}$ can be expressed as $U \models^{c}_{v} C$. This situation is depicted more graphically in Figure 2.3. 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". At the moment the only constructs we presume to exist are *elements*. Note: the fact that viewers can change their conception over time is ignored for the moment.

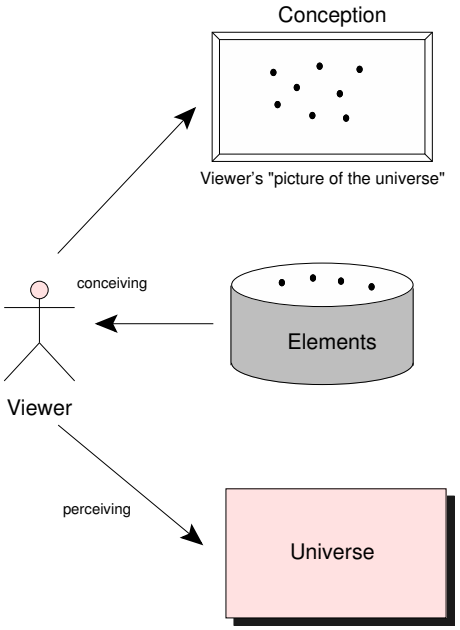


Figure 2.3: Painting a picture of the universe

Domains and their environments

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 [Ber01]. Even more, the very definition of a system depends on our ability to distinguish it from its environment. This is illustrated in Figure 2.4.

[5]The reader is advised that appendix A provides an overview of the mathematical notations/conventions used in this textbook.

[6]One should actually regard this set as the set of *states* a viewer may have. A viewer may for example have, in differing states, different interests with which they conceive the universe. The elements from $\mathcal{VW}$ should really be regarded as the states of these viewers.

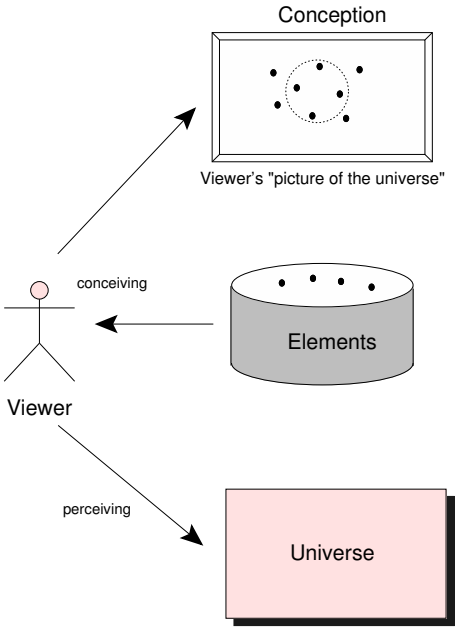


Figure 2.4: Identifying a domain and its environment

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. Formally, a domain D can be regarded as a subset of a conception C, in other words a sub-conception $D \subseteq C$. If $U \models^{\mathsf{c}}_{v} C$ and $D \subset C$ is some domain within C, then the environment of D will not generally be $C - D$ as a whole, but rather a subset $E \subseteq C - D$. For E to be a sensible environment of domain D, the elements in E must have some links to the elements in D. In order to more precisely define the notion of environment, we should therefore first refine our notion of elements of a conceptions. There are really two types of elements: concepts and links connecting the concepts. We will define these notions as follows:

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

Concept – Any element from a conception that is not a links.

Link – Any element from a conception that relates two concepts.

The distinction between a link and an 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.

Let $\mathcal{CO} \subseteq \mathcal{EL}$ be the set of concepts and let $\mathcal{LI} \subseteq \mathcal{EL}$ be the set of links. These sets should form a partition of $\mathcal{EL}$:

[S2] $\mathcal{CO} \cap \mathcal{LI} = \emptyset$ and $\mathcal{EL} = \mathcal{CO} \cup \mathcal{LI}$.

In terms of Figure 2.4, our viewer can now select from two classes of elements: concepts and links. This is depicted in Figure 2.5. In the next chapter, we will provide an even more refined view on the classes of elements we identify.

Figure 2.6 provides a model, our ontology, of the classes of elements which a conception may consist of and their mutual relationships. The notation we have used there is the ORM (Object-Role Modeling) [Hal01][7] notation, which is the same notation as used in the *Domain Modeling* course.

[7]We have used the extension introduced in [HP95] to signify that Entity and Relationship are really self-defining subtypes, i.e. not requiring a dedicated subtype defining rule.

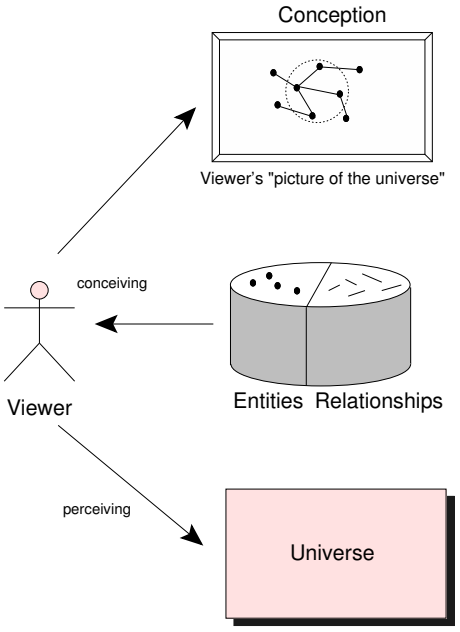


Figure 2.5: Painting a more refined picture of the observed domain

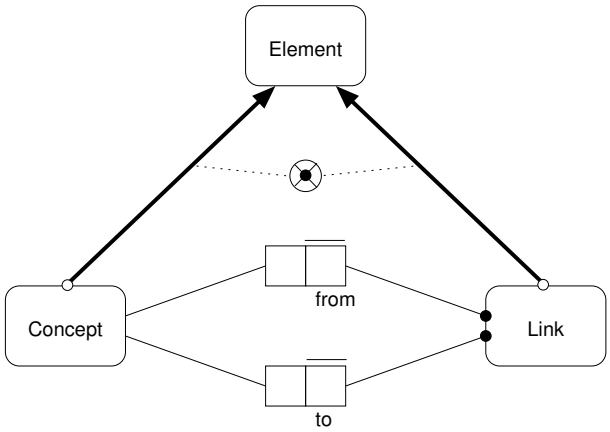


Figure 2.6: The ontology of a viewer's conception

If $X \subseteq \mathcal{EL}$, then we will use the following abbreviations:

$$\mathcal{CO}_X \triangleq X \cap \mathcal{CO} \text{ and } \mathcal{LI}_X \triangleq X \cap \mathcal{LI}$$

Links run between concepts. In other words, we can presume functions $\mathsf{From} : \mathcal{LI} \rightarrow \mathcal{CO}$ and $\mathsf{To} : \mathcal{LI} \rightarrow \mathcal{CO}$ to exist, providing the source and destination of these links respectively. As an abbreviation we will use $\mathsf{Involved}(r) \triangleq \{\mathsf{From}(r), \mathsf{To}(r)\}$. A conception C is considered to be closed iff:

$$\forall_{r \in \mathcal{LI}_C} [\mathsf{Involved}(r) \subseteq C]$$

Conceptions of viewers should be closed:

[S3] If $U \models^{\mathsf{c}}_v C$, then C is closed.

A conception C which is closed under $\mathcal{LI}$ can essentially be regarded as a graph:

$$\langle \mathcal{CO}_C, \mathcal{LI}_C, \mathsf{From}, \mathsf{To} \rangle$$

A conception C is called connected iff the associated graph is connected. A conception is required to be a connected graph:

[S4] If $U \models^{\mathsf{c}}_v C$, then C is connected.

Note: if a conception would not be a connected graph, it would be a conception of multiple universes.

We generalise From and To to sets of *relations* as follows:

$$\begin{aligned} \mathsf{From}(R) &\triangleq \left\{ \mathsf{From}(r) \;\middle|\; r \in R \right\} \\ \mathsf{To}(R) &\triangleq \left\{ \mathsf{To}(r) \;\middle|\; r \in R \right\} \end{aligned}$$

We are now in a position to properly define the environment of a domain:

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.

Formally, we view a domain D and an environment E as being a subset of a conception C, in other words a sub-conception $D, E \subseteq C$. Let $\_ \models^{\mathsf{d}}_{\_} \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$ now be the relation expressing which domain and environment a viewer conceives. The fact that a viewer v harbors a conception of domain D and environment E for universe U can be expressed as $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$. This link should limit itself to the conceptions held by viewer v:

[S5] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then $U \models^{\mathsf{c}}_v C$ and $E, D \subseteq C$.

A domain and its environment should not overlap:

[S6] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then $E \cap D = \emptyset$.

The domain and environment should be closed:

[S7] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then D is closed.

[S8] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then E is closed.

The combination of a domain and its environment is closed as well:

Corollary 2.2.1
If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then $E \cup D$ is closed.

Proof:
Left as an exercise to the reader.

A domain should be connected:

[S9] If $U \models^{\mathrm{d}}_{v} \langle C : E : D \rangle$, then D is connected.

The combination of a domain and its environment is connected as well:

[S10] If $U \models^{\mathrm{d}}_{v} \langle C : E : D \rangle$, then $D \cup E$ is connected.

Note that an environment does not have to be connected!

Concepts in the environment are related somehow to concepts in the domain:

Lemma 2.2.1
Let $U \models^{\mathrm{d}}_{v} \langle C : E : D \rangle$ with a non-empty environment E. If P is *a* maximum[8] subset of E such that it is connected, then:

$$\exists_{e \in \mathcal{CO}_P} \exists_{r \in \mathcal{LI}_C, d \in \mathcal{CO}_D} [\{e, d\} = \mathsf{Involved}(r)]$$

Proof:
Left as an exercise to the reader.

The authors of [FVSV^{+}98] 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.

In the remainder of this book, we will use the phrase: *a viewer v observing a domain (of interest) D (with environment E)* as an abbreviation for: *a viewer v having a conception of the universe, zooming in on domain D (with environment E).*

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 2.7.

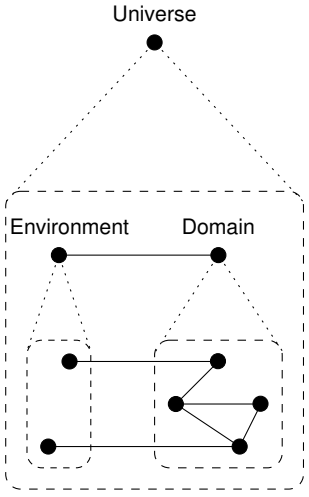


Figure 2.7: Decomposition of the universe

[8] In other words, there is no P' such that $P \subset P' \subseteq E$ while P' is still connected.

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 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 2.8.

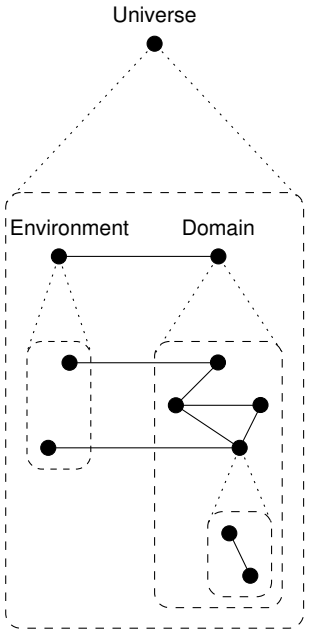


Figure 2.8: Decomposition of a part of a domain

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 2.9. This really implies we need to identify a specific class of links called decomposers. Let us therefore presume we have a set: $\mathcal{DC} \subseteq \mathcal{LI}$ of links.

To more easily reason about decompositions within a conception, we will introduce the derived relationship $\_ \rightarrow_{\_} \_ \subseteq \mathcal{CO} \times \wp(\mathcal{EL}) \times \mathcal{CO}$. If $x \rightarrow_C y$, the concept x in conception C is decomposed into (possibly amongst others) concept y. This relationship is defined (precisely) by the three following (recursive) derivation rules:

$$\begin{array}{lll} 1: & \exists_{d \in \mathcal{DC}_C} [x = \mathsf{From}(d) \wedge y = \mathsf{To}(d)] & \vdash \quad x \rightarrow_C y \\ 2: & x \rightarrow_C y \wedge y \rightarrow_C z & \vdash \quad x \rightarrow_C z \end{array}$$

The decompositions should be acyclic:

[S11] $x \rightarrow_C y \Rightarrow x \neq y$

As an abbreviation we introduce: $x \rightarrow\!\!\!\!=_C y \triangleq x = y \vee x \rightarrow_C y$.

To enforce the fact that a viewer's conception of a universe consists of one "top" concept representing the universe as a whole, we require:

[S12] If $U \models^{\mathsf{c}}_v C$, then: $\exists_{u \in \mathcal{CO}_C} \forall_{x \in \mathcal{CO}_C} [u \rightarrow\!\!\!\!=_C x]$

Even more, the $u \in C$ is unique:

Corollary 2.2.2
If $U \models^{\mathsf{c}}_v C$, then: $\exists!_{u \in \mathcal{CO}_C} \forall_{x \in \mathcal{CO}_C} [u \rightarrow\!\!\!\!=_C x]$

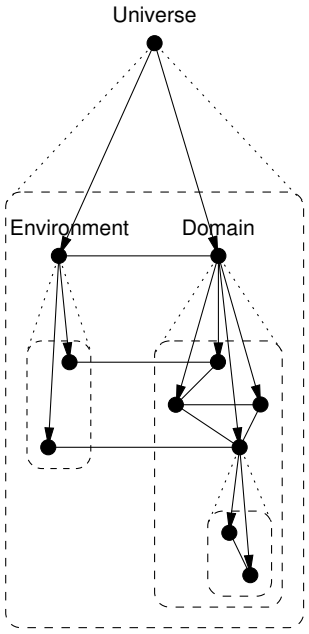


Figure 2.9: Decomposer relationships

Proof:
Left as an exercise to the reader.

Similarly, for domains and environments we have:

[S13] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then: $\exists_{d \in D} \forall_{x \in \mathcal{CC}_D} [d \rightrightarrows_D x]$

[S14] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then: $\exists_{e \in E} \forall_{x \in \mathcal{CC}_E} [e \rightrightarrows_E x]$

Corollary 2.2.2 applies to each of these as well. So for C, D and E there are unique tops in the hierarchies. This unique *top* elements will be referred to as $\mathsf{Top}(C)$, $\mathsf{Top}(D)$ and $\mathsf{Top}(E)$ respectively. For these tops, we should have:

[S15] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then: $\exists!_{d \in \mathcal{DC}_C} [\mathsf{Top}(C) = \mathsf{From}(d) \wedge \mathsf{Top}(E) = \mathsf{To}(d)]$

[S16] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then: $\exists!_{d \in \mathcal{DC}_C} [\mathsf{Top}(C) = \mathsf{From}(d) \wedge \mathsf{Top}(D) = \mathsf{To}(d)]$

[S17] If $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then: $\exists!_{r \in \mathcal{LI}_C - \mathcal{DC}_C} [\mathsf{Top}(D) = \mathsf{From}(d) \wedge \mathsf{Top}(E) = \mathsf{To}(d)]$

These three axioms require the top part of a conception to have the structure as depicted in Figure 2.9.

By adding the set of decomposers, we have now enriched our ontology to the situation as depicted in figure 2.10. Note that the asterisk (*) attached to the is decomposed into relationship signifies this to be a derived relationship.

2.2.4 Model

In the context of organizations, 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 semantical domain, such as logic.

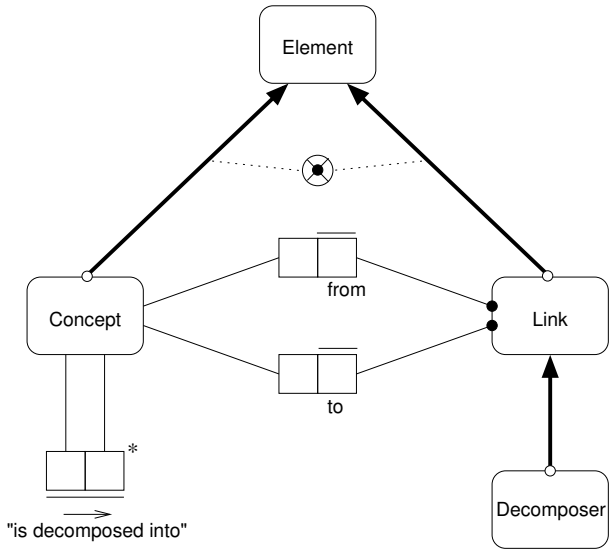


Figure 2.10: Ontology refined with decomposers

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*:

Modeling – 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 *modeling* to also include the activities involved in the *description* of the model by means of some language and medium.

To represent the fact that some viewer produces a model in an environment when they observe some part of the universe, we introduce the relation:

$$\_ \models^{\mathsf{m}}_{\_} \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$$

Formally, the fact that a viewer v views the universe U and in doing so has a conception C including a model M with environment E is represented by: $U \models^{\mathsf{m}}_{v} \langle C : E : M \rangle$. Every model is a conception. We should therefore have:

[S18] If $U \models^{\mathsf{m}}_{v} \langle C : E : M \rangle$, then $U \models^{\mathsf{d}}_{v} \langle C : E : M \rangle$.

Note that not all conceptions of a domain produce models. So the reverse of the above axiom does *not* hold! As an abbreviation we also introduce:

$$U \models^{\mathsf{m}}_{v} M \;\triangleq\; \exists_{C,E} \left[U \models^{\mathsf{m}}_{v} \langle C : E : M \rangle \right]$$

2.2.5 System

Using the above general definitions, we can, in line with [FVSV+98], 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 properties) which, in general, cannot be attributed exclusively to any of its components.

System description – The description of a system.

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 [FVSV+98], 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 they together have certain systemic properties.*

To represent the fact that some viewer "sees" a system in an environment when they observe some part of the universe, we introduce the relation:

$$\_ \models^{\mathsf{s}}_{\_} \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$$

Formally, the fact that a viewer v views the universe U and in doing so has a conception C including a system S with environment E is represented by: $U \models^{\mathsf{s}}_{v} \langle C : E : S \rangle$. Every system is a model. We should therefore have:

[S19] If $U \models^{\mathsf{s}}_{v} \langle C : E : M \rangle$, then $U \models^{\mathsf{m}}_{v} \langle C : E : M \rangle$.

Note that not all conceptions of a domain produce models. So the reverse of the above axiom does *not* hold! As an abbreviation we also introduce:

$$U \models^{\mathsf{s}}_{v} S \triangleq \exists_{C,E} [U \models^{\mathsf{s}}_{v} \langle C : E : S \rangle]$$

In our informal exploration of the concept of system, we already discussed that there are three major ways of viewing systems [Rop99]: *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 behavior of the system, while when taking the white-box approach one will see the internal structure/behavior of the system as well.

2.3 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.

2.3.1 Sub-systems

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. [Sim62]. In this textbook, for example, information systems will be positioned as sub-systems of organizational 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 to have 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 components 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 sub-system 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 [FVSV$^+$98].)

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 organizational-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.

A sub-system may, in line with [FVSV$^+$98], defined as:

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.

Formally, this can be expressed as:

$$U \models^{s}_{v} S' \subset S \quad \triangleq \quad U \models^{s}_{v} S, \quad U \models^{s}_{v} S' \text{ and } S' \subset S$$

Corollary 2.3.1
If $U \models^{s}_{v} S' \subset S$, then: $\exists_{s \in S'} \forall_{x \in \mathcal{CO}_{S'}} [s \rightrightarrows_{S'} x]$

Proof:
Left as an exercise to the reader.

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.

Formally:

$$U \models^{s}_{v} S' \subset_c S \quad \triangleq \quad U \models^{s}_{v} S' \subset S \text{ and } (S' \cap \mathcal{CO}) \subset S$$

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.

Formally:

$$U \models^{s}_{v} S' \subset_a S \quad \triangleq \quad U \models^{s}_{v} S' \subset S \text{ and } (S' \cap \mathcal{LI}) \subset S$$

Note that some authors, for example [Vel92, Bem98], 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 [FVSV+98]), 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.

2.3.2 Describing systems

When a system developer in a system viewing/modeling 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 properties, 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 interest 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 / modeler 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 it is 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 [FVSV+98]. A semi-formal description of it based on an example is presented in [Lin92].

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.

2.3.3 Open-active systems

A work system, an information system as well as an organizational system 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.

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 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 behavior 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 behavior (for details see [Ack71]). 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) organizations can be regarded as autonomous systems.

2.4 Information system

To be able to more precisely define the concept of information systems, we first need a better understanding of the concept of information.

2.4.1 Knowledge, information and data

In defining what we mean by information, we first need to more clearly define what we mean by the concepts of knowledge, communication and messages:

Knowledge – A relatively stable, and usually mostly consistent, set of conceptions posessed by a single (possibly composed) actor.

In more popular terms: "an actor's picture of the world".

Information is defined in terms of the knowledge increment brought forward when exchanging messages:

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.

where data and message are defined as follows:

Data – Any representation in some language. Data is therefore simply a collection of symbols that may, or may not, have some meaning to some actor.

Message – Data that is transmitted from one actor (the sender) to another actor (the receiver).

A message may actually be 'routed' via several actors before reaching its actual receiver. For example, when human actor exchange messages, they usually need to make use of some other actor playing the role of a medium (for example, vibrations in the air, or an e-mail system).

To make this list of definitions complete, we also provide the related notion of communication as:

Communication – An exchange of messages, i.e. a sequence of mutual and alternating message transfers between at least two human actors, called communication partners, whereby these messages represent some knowledge and are expressed in languages understood by all communication partners, and whereby some amount of knowledge about the domain of communication and about the action context and the goal of the communication is made present in all communication partners.

Using the above definition of information, we can now also define the notions of information system and computerized information system more specifically:

Information system – A sub-system of an organizational system, comprising the conception of how the communication and information-oriented aspects of an organization 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 organizational system.

Computerized information system – A sub-system of an information system, whereby all activities within that sub-system are performed by one or several computer(s).

Needless to say that in modern-day information intensive organizations information systems, and in particular the computerized parts of these systems, play an ever increasing role.

2.5 Dealing with evolution of conceptions

Before concluding this chapter, there is one final issue to deal with. An organizational 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 write $U \models^{c}_{v} C$ it 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?*.

Several strategies exist to deal with evolution [Pro94]. 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 2.11. 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 2.12.

Based on the approach taken in [Pro94], the evolution of the elements in a viewer's conception is treated as a set of (partial[9]) functions over time. At each point in time, a specific element (a

[9]For a discussion on the difference between total and partial functions, see appendix A.

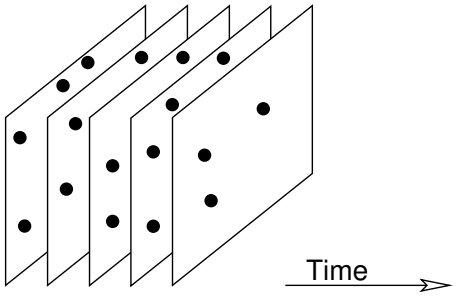


Figure 2.11: Modeling evolution by snapshots

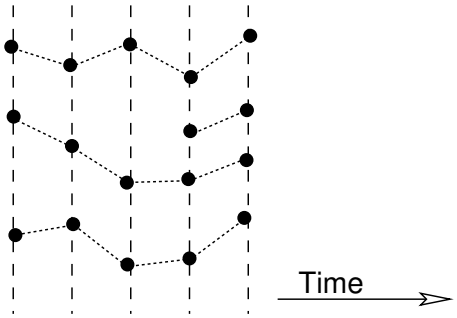


Figure 2.12: Modeling evolution by functions in time

version) may be associated to such a function. This means (as also illustrated in figure 2.13), the situation depicted in figure 2.11 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.

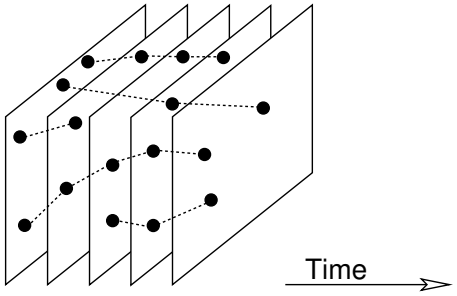


Figure 2.13: Deriving snapshots

In order to introduce the notion of evolution formally, we first need to define a time axes. We presume all viewers to agree that the time axes (at least) consists of:

- A set of points in time $\mathcal{TI}$.
- A complete and total order $< \subseteq \mathcal{TI} \times \mathcal{TI}$.

We also define: $t_1 \leq t_2 \triangleq t_1 < t_2 \vee t_1 = t_2$.

Since we have a complete order on the points in time, we can define the relation $\rhd \subseteq \mathcal{TI} \times \mathcal{TI}$ with intended intuition: if $t_1 \rhd t_2$, then t_2 is the next point in time immediately after t_2. Formally, this relation is defined as:

$$t_1 \rhd t_2 \triangleq t_1 < t_2 \wedge \neg \exists_s [t_1 < s < t_2]$$

As, $<$ provides a *complete* and *total* order, for a given t_1 there is always at most one t_2 such that $t_1 \rhd t_2$:

Corollary 2.5.1
$t_1 \rhd t_2 \wedge t_1 \rhd t_3 \Rightarrow t_2 = t_3$

Proof:
Left as an exercise to the reader.

As a result, we can actually view $\rhd$ as a function: $\rhd : \mathcal{TI} \rightarrow \mathcal{TI}$ and write $\rhd t$ as an abbreviation for: the unique t' such that $t \rhd t'$.

The evolution over time of an element from a conception, can now indeed be modelled formally as a function: $h : \mathcal{TI} \rightarrowtail \mathcal{EL}$. By means of $h(t)$ we obtain the "version" of the element's evolution as described by h at point of time t, while $h(\rhd t)$ would yield the version at the next point in time. These functions, which represent an element's evolution, will be referred to as *element evolutions*. This also requires us to think of the elements in $\mathcal{EL}$ as the possible *versions* an element evolution may take on. Whenever we need to emphasize this, we shall therefore use the term *element version*.

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. We could, for example, have a situation where: h is an element evolution, $t_1 < t_2 < t_3$ are points in time, while: $h{\downarrow}t_t \wedge \neg\, h{\downarrow}t_2 \wedge h{\downarrow}t_3$[10].

The set of all element evolutions is defined as: $\mathcal{EE} \triangleq \mathcal{TI} \rightarrowtail \mathcal{EL}$, in other words the set of partial functions from the time axes to the possible element versions. The evolution of an entire conception can then be represented formally as a set of element evolutions. In other words: $\mathcal{CE} \triangleq \wp(\mathcal{EE})$. To formally express the fact that $H \in \mathcal{CE}$ is a conception of viewer v for universe U, we will write $U \models^{\mathsf{c}}_{v} H$.

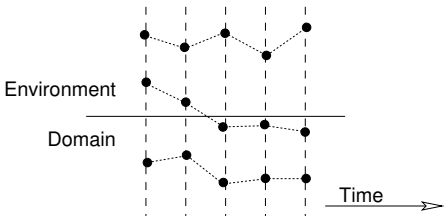


Figure 2.14: An element evolution migrating from the environment to the domain

If $H \in \mathcal{CE}$, then we will use as abbreviation: $H(t) \triangleq \{h(t) \mid h \in H\}$, yielding the entire "state" of H at time t. If H is a conception of some viewer, then each of the states should be a valid conception:

[S20] If $U \models^{\mathsf{c}}_{v} H$, then: $\forall_{t \in \mathcal{TI}} [U \models^{\mathsf{c}}_{v} H(t)]$.

Similarly, to $\_ \models^{\mathsf{c}}_{\_} \_$ we want to extend $\_ \models^{\mathsf{d}}_{\_} \langle \_ : \_ : \_ \rangle$, $\_ \models^{\mathsf{m}}_{\_} \langle \_ : \_ : \_ \rangle$ and $\_ \models^{\mathsf{s}}_{\_} \langle \_ : \_ : \_ \rangle$ to deal with evolution as well. We might, for example, view $U \models^{\mathsf{d}}_{v} \langle H_C : H_E : H_D \rangle$ to means: viewer v has a conception evolution H_C, environment evolution H_E and domain evolution H_D of universe U, where $H_D, H_E \subseteq H_C$. The difficulty with this is that an element evolution might start out as being in the environment, but might evolve *into* the domain, or vice versa. This is illustrated in Figure 2.14. To properly deal with such evolution, we will need to regard H_E and H_D as a classification of the element evolutions from H_C at each point in time. In other words as functions:

$$H_E, H_D : \mathcal{TI} \rightarrow \wp(H_C)$$

[10] Please refer to appendix A for an explanation of the notion used.

yielding the set of conception evolutions from H_C that are part of the environment/domain respectively at a given point in time. This means that $H_E(t), H_D(t) \subseteq H_C$ are sets of element evolutions, while $H_E(t)(t)$ and $H_D(t)(t)$ yield the actual environment and domain as it holds at t. For this we have:

Corollary 2.5.2

If $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then: $\forall_{t \in \mathcal{TI}} [H_E(t)(t) \subseteq H_C(t)]$ and $\forall_{t \in \mathcal{TI}} [H_D(t)(t) \subseteq H_C(t)]$.

Proof:

Left as an exercise to the reader.

Note: the same would hold for $U \models^{\mathsf{m}}_v \langle H_C : H_E : H_D \rangle$ and $U \models^{\mathsf{s}}_v \langle H_C : H_E : H_D \rangle$. In each case, the states of the evolutions should be valid conceptions/environments/domains as well:

[S21] If $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then $\forall_{t \in \mathcal{TI}} \left[U \models^{\mathsf{d}}_v \langle H_C(t) : H_E(t)(t) : H_D(t)(t) \rangle \right]$

[S22] If $U \models^{\mathsf{m}}_v \langle H_C : H_E : H_D \rangle$, then $\forall_{t \in \mathcal{TI}} [U \models^{\mathsf{m}}_v \langle H_C(t) : H_E(t)(t) : H_D(t)(t) \rangle]$

[S23] If $U \models^{\mathsf{s}}_v \langle H_C : H_E : H_D \rangle$, then $\forall_{t \in \mathcal{TI}} [U \models^{\mathsf{s}}_v \langle H_C(t) : H_E(t)(t) : H_D(t)(t) \rangle]$

Adding evolution of conceptions to our ontology from figure 2.10, leads to the refinement of our ontology to the situation as depicted in figure 2.15.

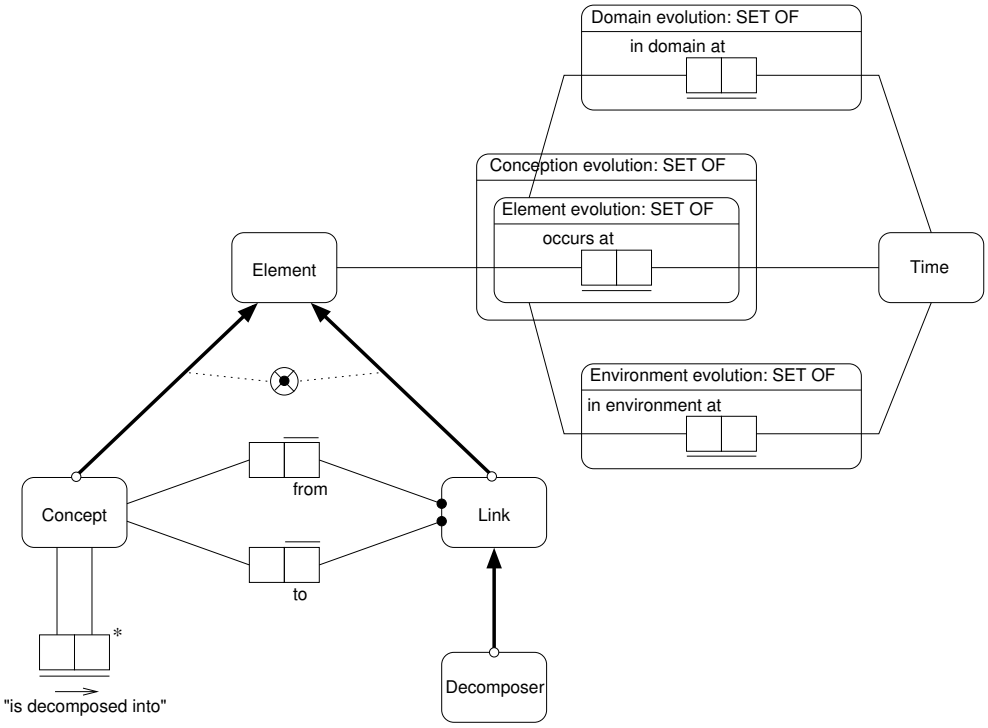


Figure 2.15: Ontology with evolution added

Mainly due to Axiom **S6** (page 53), we have:

Corollary 2.5.3

If $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then: $\forall_{t \in \mathcal{TI}} [H_E(t)(t) \cap H_D(t)(t) = \emptyset]$

Proof:

Left as an exercise to the reader.

Even more specifically, we have:

Lemma 2.5.1

If $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then: $\forall_{t \in \mathcal{TI}} [H_E(t) \cap H_D(t) = \emptyset]$

Proof:

Left as an exercise to the reader.

2.6 Conclusion

In this chapter we have taken a highly fundamental and formal outlook on information systems, organizations and work systems, and the way they are modelled, including their decompositions and evolutions.

Version: 16-03-05

Questions

1. How are the terms 'organization', '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 the formal language given in this chapter.
2. Proof Corollary 2.2.1 (page 53).
3. Proof Lemma 2.2.1 (page 54).
4. Not all conceptions of a domain produce models. Why not?
5. 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).
6. From a modeling point of view, organizations can be considered as systems containing a.o. entities and relations.
   - Why is it important to be aware of the aspect of subjectivity when creating models?
   - What view does an information system developer have when modeling organizations?
   - Why would an information system developer want to start by creating a model of an organization, instead of directly focusing on modeling an information system?
7. Proof Corollary 2.2.2 (page 55).
8. Suppose you are requested by a large organization (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:
   (a) Where would you start modeling?
   (b) What would you model?
   (c) Why model that?
9. Proof Corollary 2.3.1 (page 60).
10. Consider a home cinema set.
    (a) Describe the systems elements.
    (b) Distinguish proper sub-systems.
    (c) Can you derive typical aspect systems and component systems?

    Explain your answers.
11. Consider a travel agency.
    (a) Describe the most important system characteristics and exposition characteristics.
    (b) Describe its behavior in terms of internal and external functions.
12. Describe why *information* systems contain *data*bases. Use the descriptions of the terms data, information and knowledge as described in Chapter 2 in your description.
13. Consider our definition of information intensive organizations.

(a) Give some examples of:
   i. Information intensive organizations
   ii. Non-information-intensive organizations

(b) How would you describe the distinction between information intensive organizations and non-information organizations?

14. Describe, in your own words, the differences between knowledge, information and data.
15. Proof Corollary 2.5.1 (page 66).
16. Proof Corollary 2.5.2 (page 67).
17. Proof Corollary 2.5.3 (page 67).
18. Proof Lemma 2.5.1 (page 67).

Bibliography

[Ack71] R.L. Ackoff. Towards a system of system concepts. *Management Science*, 17, July 1971.

[Bem98] T.M.A. Bemelmans. *Bestuurlijke Informatiesystemen en Automatisering*. Kluwer, Deventer, The Netherlands, EU, 7th edition, 1998. In Dutch. ISBN 9026727984

[Ber01] L. von Bertalanffy. *General Systems Theory – Foundations, Development, Applications*. George Braziller, New York, New York, USA, revised edition, 2001. ISBN 0807604534

[Che81] P. Checkland. *Systems thinking, systems practice*. John Wiley & Sons, New York, New York, USA, 1981. ISBN 0471279110

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1558606726

[HP95] T.A. Halpin and H.A. (Erik) Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000. ISBN 0738125180
`http://www.ieee.org`

[Iiv83] J. Iivari. Contributions to the theoretical foundations of systemeering research and the pioco model. Technical Report 150, University of Oulu, Oulu, Finland, EU, 1983. ISBN 9514215435

[Lan71] B. Langefors. *Editorial notes to: Computer Aided Information Systems Analysis and Design*. Studentlitteratur, Lund, Sweden, EU, 1971.

[Lin92] P. Lindgreen. A General Framework for Understanding Semantic Structures. In E.D. Falkenberg, C. Rolland, and E.N. El Sayed, editors, *Information System Concepts: Improving the understanding – Proceedings of the second IFIP WG8.1 working conference (ISCO-2)*, Alexandria, Egypt, April 1992. North Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU. ISBN 0444895078

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003. http://www.webster.com

[Pei69a] C.S. Peirce. *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 0674138007

[Pei69b] C.S. Peirce. *Volumes III and IV – Exact Logic and The Simplest Mathematics*. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA, 1969. ISBN 0674138005

[Pei69c] C.S. Peirce. *Volumes V and VI – Pragmatism and Pragmaticism and Scientific Metaphysics*. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA, 1969. ISBN 0674138023

[Pei69d] C.S. Peirce. *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 0674138031

[Pro94] H.A. (Erik) Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[Rop99] G. Ropohl. Philosophy of socio-technical systems. *In Society for Philosophy and Technology*, 4(3), 1999.

[Sim62] H.A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, 1962.

[Vel92] J. in 't Veld. *Analyse van organisatieproblemen – Een toepassing van denken in systemen en processen*. Stenfert Kroese, Leiden, The Netherlands, EU, 1992. In Dutch. ISBN 9020722816

Chapter 3

Basic Object-Role Modeling

Version: 03-06-05

This is where the reading of these lecture notes gets bumpy. The previous chapters already have a reasonable amount of text explaining the theory. This chapter is still in an outline (i.e. close to powerpoint style) state.

The previous chapter did (see Figure 2.1) 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 modeling approach from *Domain Modeling*. In the next chapter, we will enrich this modeling approach with constructs that allow us to model work systems.

3.1 Natural language grounding of modeling

Not an uncommon approach:

- ORM [Hal01],
- NIAM [Win90, NH89],
- UML use cases [BRJ99],
- DEMO [RMD99],
- KISS [Kri94] and
- OOSA [EKW92].

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) organizations 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 modeling endeavors on the language we use most to talk about organizations, i.e. natural language.

3.2 The logbook heuristic

Natural language based modeling 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 3.1. The left hand viewer tells the right hand viewer 'what they see'.

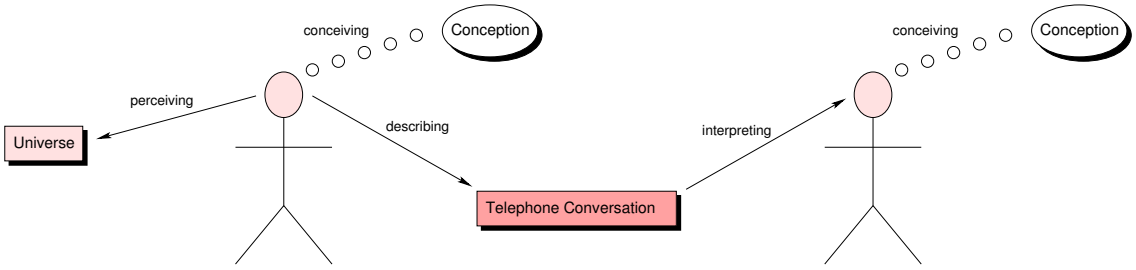


Figure 3.1: The telephone heuristic

In this chapter we are interested in having a "transcript" of the telephone conversation from Figure 3.1. More specifically, we want to maintain a *logbook* of this telephone conversation, leading to the situation as depicted in Figure 3.2.

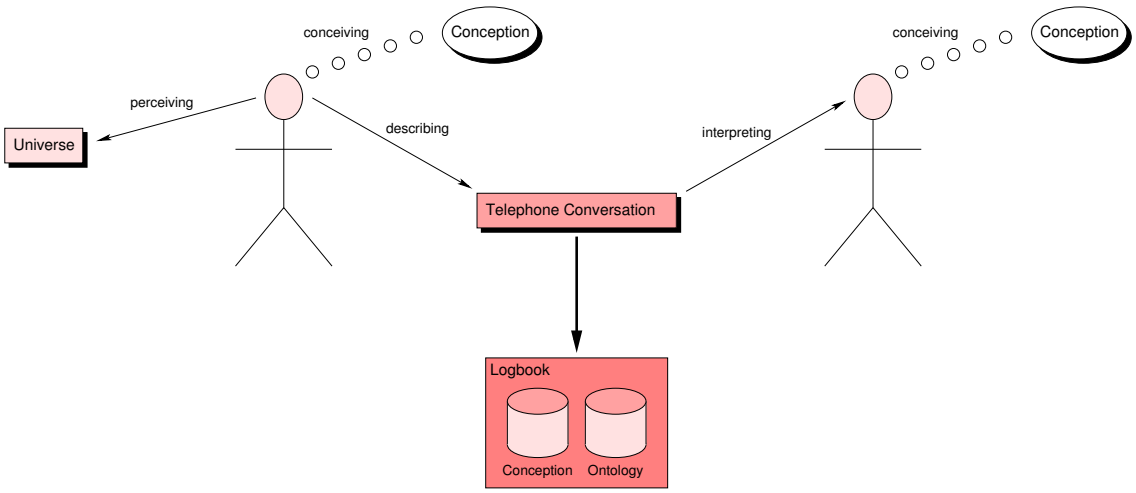


Figure 3.2: Logging the telephone conversation

Even more, we could actually replace the second person from Figure 3.1, leaving only the original viewer and the logbook to maintain the transcript. This leads to the *logbook heuristic* as depicted in Figure 3.3.

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 2.15 (page 67).

Let λ be a logbook, logging the transcript as produced by a viewer leads to a situation where the logbook has its own conception of the observed universe (by way of the viewer). In the case of a viewer observing a system, we will denote this as: $U \models^{\mathbf{s}}_{\lambda} \langle H_C : H_E : H_S \rangle$.

All of the **S** axioms apply to the conception held by logbook λ.

In the remainder of this text book, we will further refine the ontology as presented in Figure 2.15 (page 67)

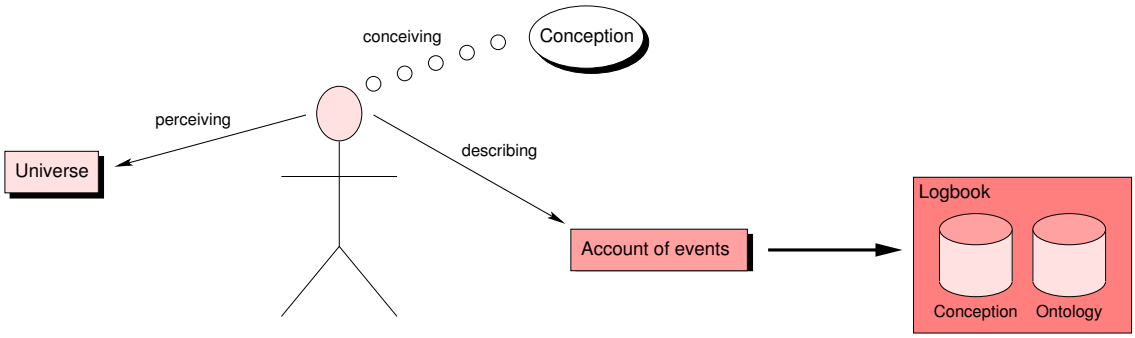


Figure 3.3: The logbook heuristic

3.3 Verbalizing conceptions

We presume the transcriptions that are entered into the logbook to refer to events 'in the life' of specific element evolutions. 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	22-05-1967
Person #001 received name Erik Proper	23-05-1967
Person #001 lives at address: Koperwiekstraat 6, Rheden, The Netherlands, EU	23-05-1967
Person #001 lives at address: 3/26 Rylatt Street, Brisbane, Australia	28-06-1994
Person #002 lives at address: Koperwiekstraat 6, Rheden, The Netherlands, EU	29-06-1994
Person #001 works for employer: University of Queensland	28-06-1994
Person #003 works for employer: University of Queensland	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.

3.4 Elementary facts

Similarly to *Domain Modeling*, we require the facts in the transcripts to be *elementary*, in other words, no logical connectors like *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 *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 links in a conception can be treated as a relationship over concepts:

$$x\ \mathsf{LinkedTo}\ y \ \triangleq\ \exists_{l \in \mathcal{LI}}\, [\mathsf{From}(l) = x \wedge \mathsf{To}(l) = y]$$

The roles involved in a fact are linked to the *players* of these roles and to the *facts* in which these roles take part by means of the relationships:

$$\mathsf{Player}, \mathsf{Fact} \subseteq \mathsf{LinkedTo}$$

respectively, with intuition:

$$\begin{array}{lcl} r\ \mathsf{Player}\ x & = & \text{role } r \text{ is played by } x \\ r\ \mathsf{Fact}\ x & = & \text{role } r \text{ is involved in fact } x \end{array}$$

The set of roles is defined as:

$$\mathcal{RO} \ \triangleq\ \left\{ r \ \middle|\ \exists_x\, [r\ \mathsf{Player}\ x \vee r\ \mathsf{Fact}\ x] \right\}$$

The Player and Fact relationships are exclusive:

[S24] $\mathsf{Player} \cap \mathsf{Fact} = \emptyset$

Even more, they behave as total functions from roles to concepts:

[S25] $\mathsf{Player}, \mathsf{Fact} \in \mathcal{RO} \rightarrow \mathcal{CO}$

This allows us to write $\mathsf{Player}(r)$ and $\mathsf{Fact}(r)$. We generalize these functions to sets of roles as follows:

$$\begin{array}{rcl} \mathsf{Player}(R) & \triangleq & \left\{ \mathsf{Player}(r) \ \middle|\ r \in R \right\} \\ \mathsf{Fact}(R) & \triangleq & \left\{ \mathsf{Fact}(r) \ \middle|\ r \in R \right\} \end{array}$$

With this we can define the set of facts and players as:

$$\begin{array}{rcl} \mathcal{FC} & \triangleq & \mathsf{Fact}(\mathcal{RO}) \\ \mathcal{PL} & \triangleq & \mathsf{Player}(\mathcal{RO}) \end{array}$$

When taking 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 set of objects is defined as:

$$\mathcal{OB} \triangleq \mathcal{FC} \cup \mathcal{PL}$$

Objects and roles are disjunct classes of concepts:

[S26] $\mathcal{RO} \cap \mathcal{OB} = \emptyset$

The set of roles played by an object is defined as:

$$\mathsf{Plays}(x) \triangleq \{ r \mid \mathsf{Player}(r) = x \}$$

while the set of roles involved in a fact is defined as:

$$\mathsf{RolesOf}(f) \triangleq \{ r \mid \mathsf{Fact}(r) = f \}$$

Facts are to be regarded as complex objects. In other words, the roles and players involved in a fact are part of its decomposition:

[S27] $\forall_{r \in \mathsf{RolesOf}(f)} [r \ \mathsf{DecompOf} \ f]$

[S28] $r \in \mathcal{RO} \wedge r \ \mathsf{DecompOf} \ f \Rightarrow \mathsf{Player}(r) \ \mathsf{DecompOf} \ f$

At this moment, we have actually refined our ontology from Figure 2.15 (page 67) to the situation as depicted in Figure 3.4 (where we have omitted the aspects pertaining to the evolution of conceptions for reasons of compactness).

Our formal considerations take place in the context of some logbook λ. In other words, we have: $U \overset{\mathsf{s}}{\models}_{\lambda} \langle H_C : H_E : H_S \rangle$. Given a H_C, the set of elements of the conception at some point in time t is given by: $H_C(t)$. For any subset $X \subseteq \mathcal{EL}$ of elements we introduce the abbreviation:

$$X_t \triangleq X \cap H_C(t)$$

A conception version should be closed with regards to the roles included in it:

[S29] $r \in \mathcal{RO}_t \Leftrightarrow \mathsf{Fact}(r) \in \mathcal{FC}_t$ and $r \in \mathcal{RO}_t \Leftrightarrow \mathsf{Player}(r) \in \mathcal{PL}_t$

As a direct consequence we have:

Corollary 3.4.1
$\mathcal{FC}_t = \mathsf{Fact}(\mathcal{RO}_t)$ and $\mathcal{PL}_t = \mathsf{Player}(\mathcal{RO}_t)$.

Proof:
Left as an exercise to the reader.

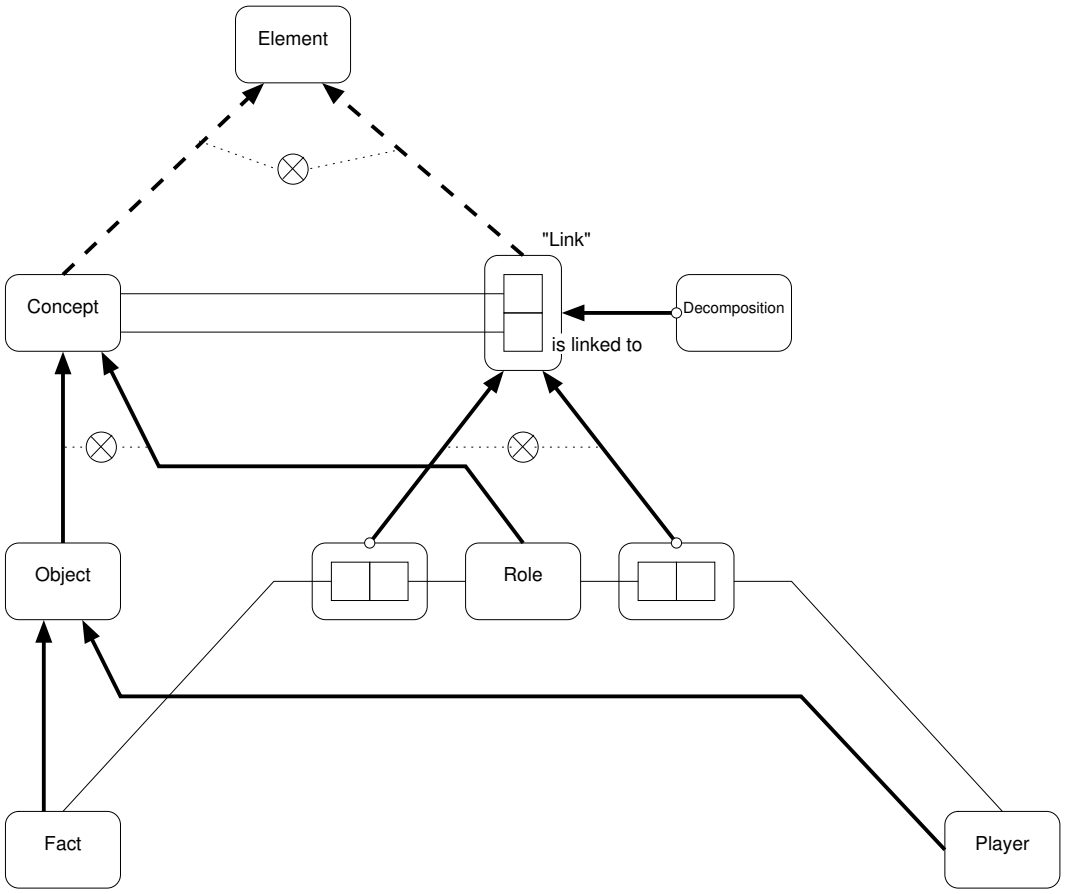


Figure 3.4: Our ontology refined with facts and participants

3.5 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 Modeling*, 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: $\mathsf{HasType} \subseteq \mathsf{DecompOf}$ where:

$$x \,\mathsf{DecompOf}\, y \;\triangleq\; \exists_{l \in \mathcal{DC}} \left[\mathsf{From}(l) = y \wedge \mathsf{To}(l) = x\right]$$

The set of instances ($\mathcal{IN}$) and the set of types ($\mathcal{TP}$) can be defined as:

$$\begin{array}{rcl} \mathcal{TP} & \triangleq & \left\{ y \;\middle|\; \exists_x \left[x \,\mathsf{HasType}\, y\right] \right\} \\ \mathcal{IN} & \triangleq & \left\{ x \;\middle|\; \exists_y \left[x \,\mathsf{HasType}\, y\right] \right\} \end{array}$$

Types and instances form a partition of the set of concepts:

[S30] $\mathcal{TP} \cap \mathcal{IN} = \emptyset$ and $\mathcal{TP} \cup \mathcal{IN} = \mathcal{CO}$.

All instances have some type:

Corollary 3.5.1
$\forall_{x \in \mathcal{IN}} \exists_{y \in \mathcal{TP}} \left[x \,\mathsf{HasType}\, y\right]$

Proof:
Left as an exercise to the reader.

Note that the reverse does not hold. In other words, we do not generally have:

$$\forall_{y \in \mathcal{TP}} \exists_{x \in \mathcal{IN}} \left[x \,\mathsf{HasType}\, y\right]$$

Some types may have an empty population.

We actually require Corollary 3.5.1 to hold at each point in time:

[S31] Let $t \in \mathcal{TI}$, then: $\forall_{x \in \mathcal{IN}} \exists_{y \in \mathcal{TP}} \left[x \,\mathsf{HasType}_t\, y\right]$

Sets such as $\mathcal{FC}$, $\mathcal{PL}$, etc, contain both types and instances. To be able to refer to the types and instances respectively, we introduce:

$$\hat{X} \;\triangleq\; X \cap \mathcal{TP} \quad \text{and} \quad \check{X} \;\triangleq\; X \cap \mathcal{IN}$$

for any set $X \subseteq \mathcal{EL}$ of elements. As a direct consequence we have:

Corollary 3.5.2

$$\begin{array}{rclcrcl} \hat{\mathcal{PL}} & \triangleq & \mathsf{Player}(\hat{\mathcal{RO}}) & & \check{\mathcal{PL}} & \triangleq & \mathsf{Player}(\check{\mathcal{RO}}) \\ \hat{\mathcal{FC}} & \triangleq & \mathsf{Fact}(\hat{\mathcal{RO}}) & & \check{\mathcal{FC}} & \triangleq & \mathsf{Fact}(\check{\mathcal{RO}}) \end{array}$$

Proof:
Left as an exercise to the reader.

As abbreviations we will also use:

$$\begin{aligned}
\mathsf{Types}_t(x) &\triangleq \{y \mid x\ \mathsf{HasType}_t\ y\} \\
\mathsf{Types}_t(X) &\triangleq \bigcup_{x \in X} \mathsf{Types}_t(x) \\
\mathsf{Pop}_t(y) &\triangleq \{x \mid x\ \mathsf{HasType}_t\ y\} \\
\mathsf{Pop}_t(Y) &\triangleq \bigcup_{y \in Y} \mathsf{Pop}_t(y)
\end{aligned}$$

The *extra temporal* versions are defined as:

$$\begin{aligned}
x\ \mathsf{HasType}\ y &\triangleq \exists_{t \in \mathcal{TI}}\ [x\ \mathsf{HasType}_t\ y] \\
\mathsf{Types}(x) &\triangleq \bigcup_{t \in \mathcal{TI}} \mathsf{Types}_t(x) \\
\mathsf{Types}(X) &\triangleq \bigcup_{t \in \mathcal{TI}} \mathsf{Types}_t(X) \\
\mathsf{Pop}(y) &\triangleq \bigcup_{t \in \mathcal{TI}} \mathsf{Pop}_t(y) \\
\mathsf{Pop}(Y) &\triangleq \bigcup_{t \in \mathcal{TI}} \mathsf{Pop}_t(Y)
\end{aligned}$$

Typing should adhere to the classification in our ontology. In other words:

[S32] For all $X \in \{\mathcal{FC}, \mathcal{RO}, \mathcal{PL}\}$ we have:

$$x\ \mathsf{HasType}\ y \Rightarrow (x \in X \Leftrightarrow y \in X)$$

The Fact and Player functions should never cross the type/instance level:

[S33] $\forall_{r \in \mathcal{RO}}\ [r \in \mathcal{TP} \Leftrightarrow \mathsf{Fact}(r) \in \mathcal{TP}]$

[S34] $\forall_{r \in \mathcal{RO}}\ [r \in \mathcal{TP} \Leftrightarrow \mathsf{Player}(r) \in \mathcal{TP}]$

All non-typing forms of decomposition should also not cross the type/instance level:

[S35] If $x\ \mathsf{DecompOf}\ y$, then:

$$x\ \mathsf{HasType}\ y \vee x, y \in \mathcal{TP} \vee x, y \in \mathcal{IN}$$

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

[S36] If $r \in \hat{\mathcal{RO}}$, then: $\mathsf{Player}(\mathsf{Pop}(r)) \subseteq \mathsf{Pop}(\mathsf{Player}(r))$

The same applies to facts:

[S37] If $r \in \hat{\mathcal{RO}}$, then: $\mathsf{Fact}(\mathsf{Pop}(r)) \subseteq \mathsf{Pop}(\mathsf{Fact}(r))$

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

[S38] If $r \in \hat{\mathcal{RO}}$, then: $\mathsf{Fact}(\mathsf{Pop}(r)) \supseteq \mathsf{Pop}(\mathsf{Fact}(r))$

As an immediate result we have:

Corollary 3.5.3
If $f \in \hat{\mathcal{FC}}$, then:

$$\mathsf{Pop}(f) = \mathsf{Fact}(\mathsf{Pop}(\mathsf{Involved}(f)))$$

Proof:
Left as an exercise to the reader.

We can express this at the type level as:

Corollary 3.5.4
Let $f \in \hat{\mathcal{FC}}$, then:

$$\mathsf{Types}(\mathsf{RolesOf}(f)) = \mathsf{RolesOf}(\mathsf{Types}(f))$$

Axiom **S38** does not have a pendent for players. In other words, we do not generally have:

$$\mathsf{Player}(\mathsf{Pop}(r)) \supseteq \mathsf{Pop}(\mathsf{Player}(r))$$

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:

[S39] If $p \in \hat{\mathcal{PL}}$, then:

$$\mathsf{Pop}(p) \subseteq \mathsf{Player}(\mathsf{Pop}(\mathsf{Plays}(p)))$$

In other words, instances of a player type *must* be active in one of the associated roles. With Axiom **S36** we have:

Corollary 3.5.5
If $p \in \hat{\mathcal{PL}}$, then:

$$\mathsf{Pop}(p) = \mathsf{Player}(\mathsf{Pop}(\mathsf{Plays}(p)))$$

Proof:
Left as an exercise to the reader.

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

[S40] Let $r \in \hat{\mathcal{RO}}$ and $s_1, s_2 \in \mathsf{Pop}(r)$, then:

$$\mathsf{Fact}(s_1) = \mathsf{Fact}(s_2) \Rightarrow s_1 = s_2$$

This axiom allows us, for any fact instance $f \in \hat{\mathcal{FC}}$, to define the *partial* function $\vec{f} : \hat{\mathcal{RO}} \rightarrowtail \check{\mathcal{OB}}$ as:

$$\vec{f} \triangleq \{\langle r, \mathsf{Player}(s)\rangle \mid s \in \mathsf{Pop}(r) \wedge \mathsf{Fact}(s) = f\}$$

If f is some fact and $R \subseteq \mathsf{RolesOf}(\mathsf{Types}(f))$, then we define the *total* function $\vec{f}[R] : R \rightarrow \check{\mathcal{OB}}$ as:

$$\vec{f}[R] \triangleq \{\langle r, \vec{f}(r)\rangle \mid r \in R\}$$

The ontology resulting after this refinement is depicted in Figure 3.5.

3.6 Standard constraints

Let $R \subseteq \mathsf{Involved}(f)$ for some fact type $f \in \hat{\mathsf{Fact}}$, then:

$$\begin{aligned}\mathsf{Unique}_t(f : R) &\triangleq \forall_{g_1,g_2 \in \mathsf{Pop}_t(f)}\,[g_1[R] = g_2[R] \Rightarrow g_1 = g_2]\\ \mathsf{Unique}(f : R) &\triangleq \forall_{t \in \mathcal{TI}}\,[\mathsf{Unique}_t(f : R)]\end{aligned}$$

Let $R \subseteq \hat{\mathcal{RO}}$ such that $\forall_{r,s \in R}\,[\mathsf{Player}(r) = \mathsf{Player}(s)]$, then:

$$\begin{aligned}\mathsf{Total}_t(R) &\triangleq \bigcup_{r \in R} \mathsf{Pop}_t(\mathsf{Player}(r)) \subseteq \bigcup_{r \in R} \{f(r) \mid f \in \mathsf{Pop}_t(\mathsf{Fact}(r))\}\\ \mathsf{Total}(R) &\triangleq \forall_{t \in \mathcal{TI}}\,[\mathsf{Total}_t(R)]\end{aligned}$$

Inter-facttype constraints can, as usual, best be defined as constraints on derived facttypes.

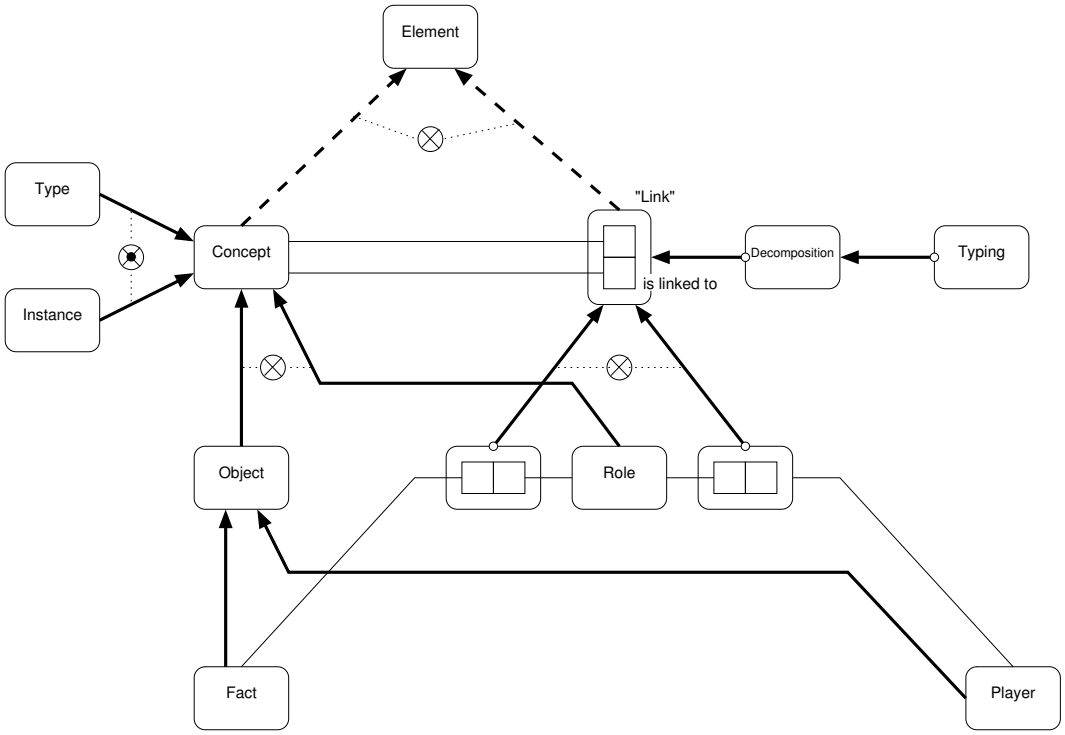


Figure 3.5: Our ontology refined with typing

3.7 Temporal ordering

Now consider the following verbalizations (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 3.6.

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 3.7.

But also consider the situation as depicted in Figure 3.8. What is the semantic difference?

As a graphical abbreviation, we will use the notation as depicted in Figure 3.9 and Figure 3.10 respectively.

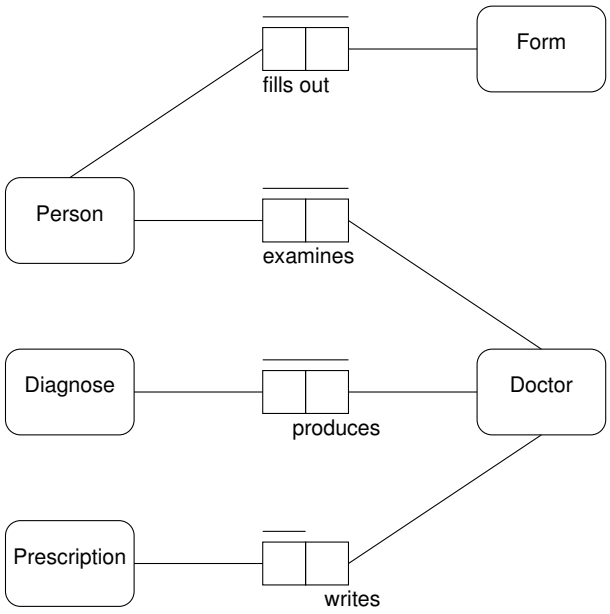


Figure 3.6: Basic model of a visit to a Doctor

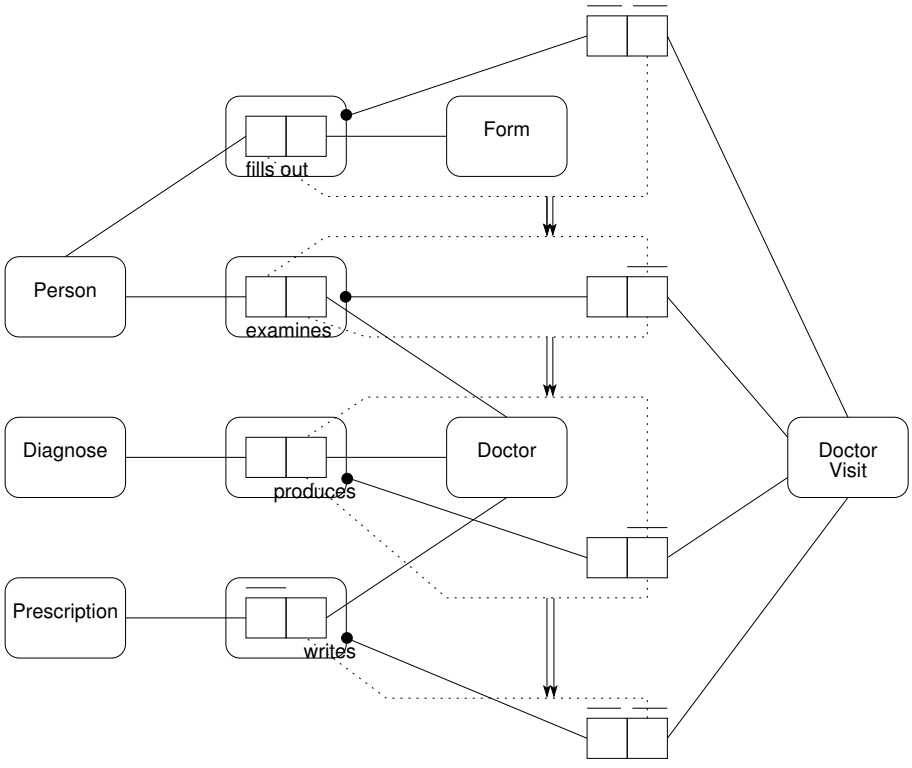


Figure 3.7: Model of a visit to a Doctor with explicit entity

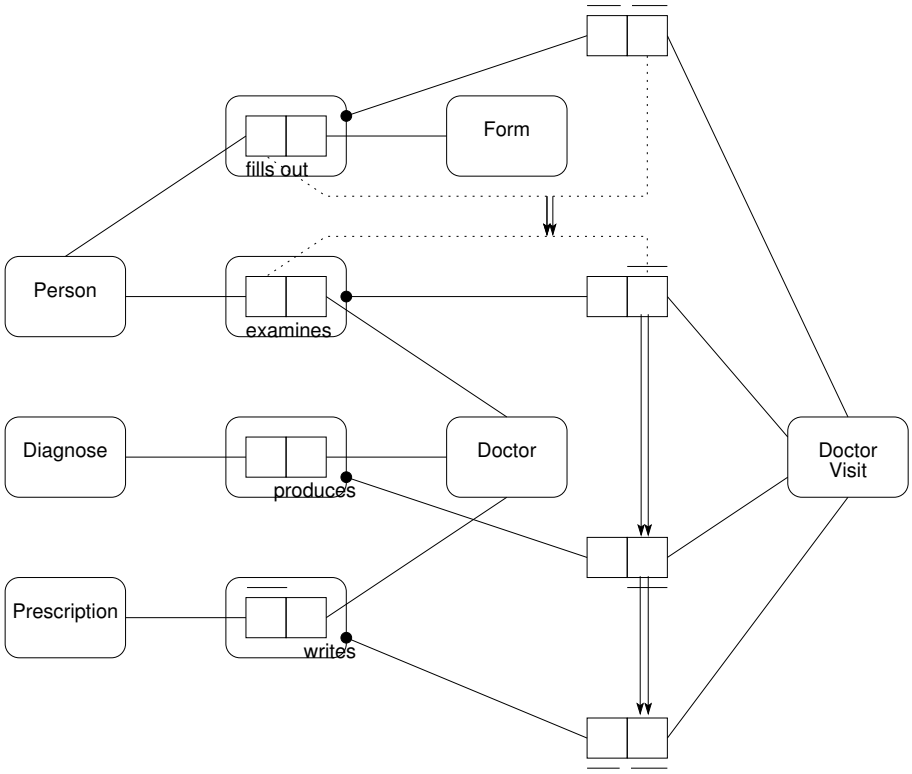


Figure 3.8: Model of a visit to a Doctor with alternative semantics

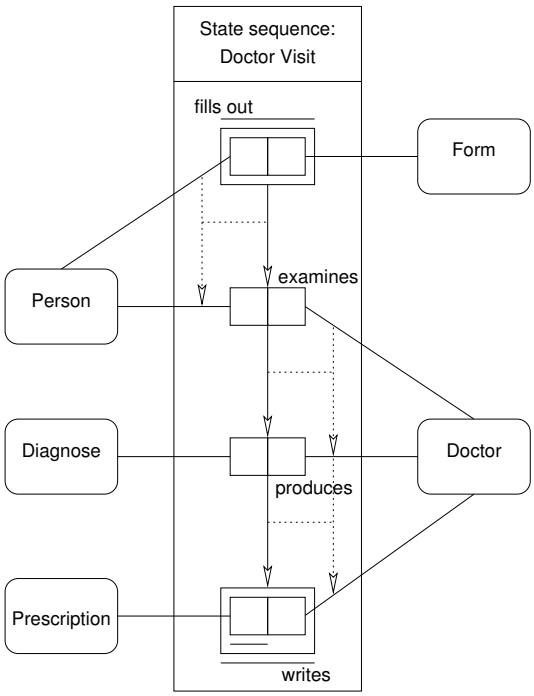


Figure 3.9: Compact model of a visit to a Doctor

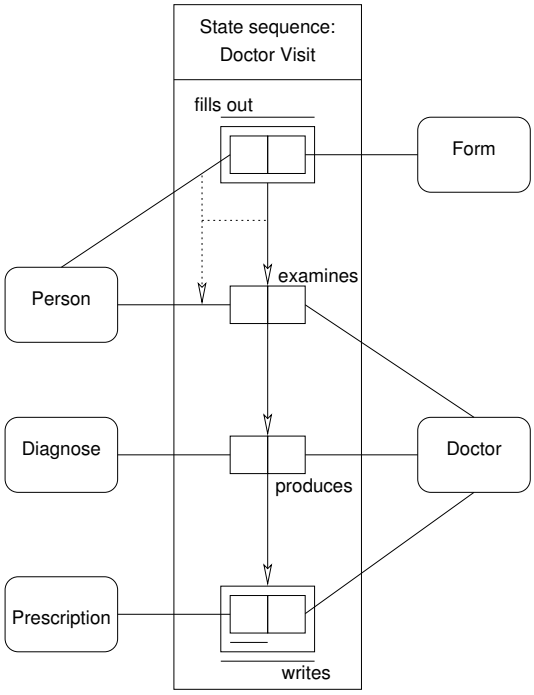


Figure 3.10: Compact model of a visit to a Doctor with alternative semantics

Formally, temporal dependency constraints can be defined as follows. Let $f \in \hat{\mathcal{FC}}$, $R \in \mathsf{RolesOf}(f)^+$ and $x \in \mathsf{Pop}(\mathsf{RolesOf}(f))^+$ such that $|R| = |x|$, then:

$$\begin{aligned}
\mathsf{Matches}(g, x, R) &\triangleq \forall_{1 \leq i \leq |x|} \left[\vec{g}(R[i]) = x[i]\right] \\
\mathsf{Ended}_t(x, R) &\triangleq \exists_{g \in \mathsf{Pop}_t(f) - \mathsf{Pop}_{\rhd\, t}(f)} \left[\mathsf{Matches}(g, x, R)\right] \\
\mathsf{Started}_t(x, R) &\triangleq \exists_{g \in \mathsf{Pop}_{\rhd\, t}(f) - \mathsf{Pop}_t(f)} \left[\mathsf{Matches}(g, x, R)\right]
\end{aligned}$$

Let $f, g \in \hat{\mathcal{FC}}$, $R \in \mathsf{RolesOf}(f)^+$, $S \in \mathsf{RolesOf}(g)^+$ and $x \in \mathcal{EL}^+$ such that $|R| = |S| = |x|$ and $\forall_{1 \leq i \leq |x|} \left[\mathsf{Player}(R[i]) \sim \mathsf{Player}(S[i])\right]$, then:

$$\begin{aligned}
\mathsf{Before}_t(x, R, S) &\triangleq \mathsf{Ended}_t(x, R) \Leftrightarrow \mathsf{Started}_t(x, S) \\
\mathsf{Before}_t(R, S) &\triangleq \forall_{x \in \mathcal{CO}^+} \left[|x| = |R| \Rightarrow \mathsf{Before}_t(x, R, S)\right]
\end{aligned}$$

Inter-facttype versions can, again, best be defined as constraints on derived facttypes.

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. Stel je maakt een ontwerp voor een geldautomaat. Wat zijn voor dat domein de belangrijkste concepten en hun onderlinge links? Hoe werken ze samen?
3. Proof Corollary 3.4.1 (page 75).

4. Proof Corollary 3.5.1 (page 77).
5. Proof Corollary 3.5.2 (page 77).
6. Proof Corollary 3.5.5 (page 79).
7. Proof Corollary 3.5.4 (page 79).
8. Consider the following case:

   Een onderneming produceert en verkoopt een tiental soorten gevulde chocoladeartikelen. 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 €0,10 per stuk. Op de artikelen met een prijs tot €5,- komt één, op de overige artikelen (tussen €5,- en €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.

Bibliography

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999. ISBN 0201571684

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-Oriented Systems Analysis – A model-driven approach*. Yourdon Press, Englewood Cliffs, New Jersey, USA, 1992. ASIN 0136299733

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1558606726

[Kri94] G. Kristen. *Object Orientation – The KISS Method, From Information Architecture to Information System*. Addison-Wesley, Reading, Massachusetts, USA, 1994. ISBN 0201422999

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630

[RMD99] V.E. van Reijswoud, J.B.F Mulder, and J.L.G. Dietz. Commucation Action Based Business Process and Information Modelling with DEMO. *The Information Systems Journal*, 9(2):117–138, 1999.

[Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.

Chapter 4

Advanced Object-Role Modeling

Version:
12-05-05

This chapter is mainly based on the work reported in [HW93, BBMP95, HP95, CP96].

4.1 Subtyping

Sub-typing is an important feature of fact-based modeling. Figure 4.1 shows an example of sub-typing in terms of a specialization hierarchy.

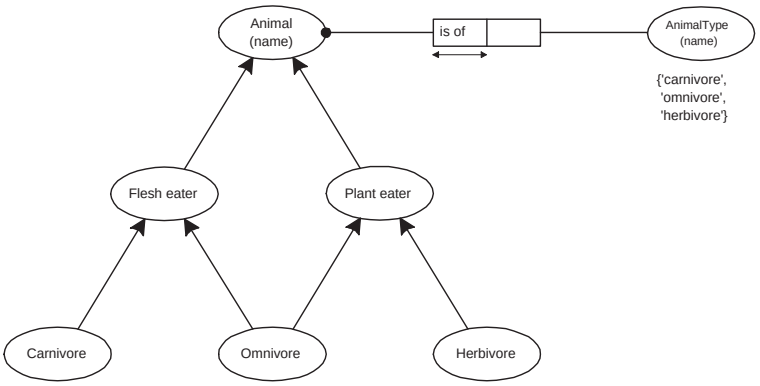


Figure 4.1: Example of a specialization hierarchy

In general, sub-typing involves the identification of a sub-set of the population of some super-type. For example, in the situation depicted in Figure 4.1 *flesh eater* is a specific sub-set of *animals*. In different versions of ORM, different rules apply to sub-typing [HW93, HP95, Hal01]. In this textbook we present a rather generic interpretation.

Formally, sub-typing is captured as a relationship $\mathsf{Sub} \subseteq \mathsf{LinkedTo}$ with intuition: if $x \,\mathsf{Sub}\, y$, then type x is considered to be a subtype of y. We will also refer to x as the *sub-type* and y as the *super-type*. Sub-typing is a relationship over object types:

[S41] $\mathsf{Sub} \subseteq \hat{\mathcal{OB}} \times \hat{\mathcal{OB}}$

The sub-typing relationship should be transitive and acyclic:

[S42] $x \,\mathsf{Sub}\, y \,\mathsf{Sub}\, z \Rightarrow x \,\mathsf{Sub}\, z$

[S43] $\neg x \,\mathsf{Sub}\, x$

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

[S44] $x \,\mathsf{Sub}\, y \Rightarrow \mathsf{Pop}(x) \subseteq \mathsf{Pop}(y)$

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, if $x \,\mathsf{Sub}\, y$ then the population of x is bounded by: $\emptyset \subseteq \mathsf{Pop}(x) \subseteq \mathsf{Pop}(y)$. Graphically, this leads to the situation i) as depicted in Figure 4.2.

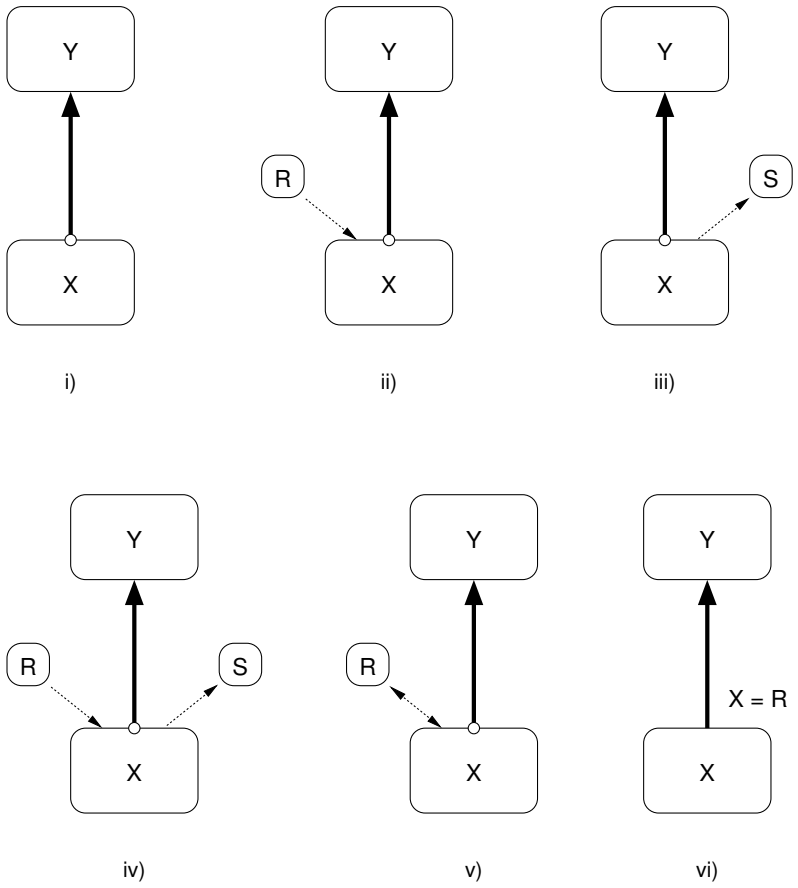


Figure 4.2: Restricting subtyping

The population of a subtyping 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 4.2 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:

$$\mathsf{SubSet}(x, R) \;\triangleq\; \Sigma[R](\mathsf{Pop}_t) \subseteq \mathsf{Pop}_t(x)$$

where $\Sigma[R](\mathsf{Pop}_t)$ provides the set of instances in Pop_t that match rule R. Note: for R to be a sensible rule, we should at least have:

$$\forall_y \left[x \,\mathsf{Sub}\, y \Rightarrow \Sigma[R](\mathsf{Pop}_t) \subseteq \mathsf{Pop}_t(y)\right]$$

otherwise $\mathsf{SubSet}(x, R)$ would lead to inconsistent set of constraints on the population.

While the situation from ii) provides a minimum population for x, the situation depicted in iii) does the reverse by demanding a maximimum population. The population of x is limited to those instances of x's *supertypes* that match rule S.

$$\mathsf{SuperSet}(x, S) \triangleq \mathsf{Pop}_t(x) \subseteq \Sigma[S](\mathsf{Pop}_t)$$

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:

$$\Sigma[R](\mathsf{Pop}_t) \subseteq \mathsf{Pop}_t(x) \subseteq \Sigma[R](\mathsf{Pop}_t)$$

in other words:

$$\mathsf{Pop}_t(x) = \Sigma[R](\mathsf{Pop}_t)$$

As a graphical abbreviation we will use the notation provided in situation vi), which corresponds to the traditional notion of *specialization* from ORM [Hal01].

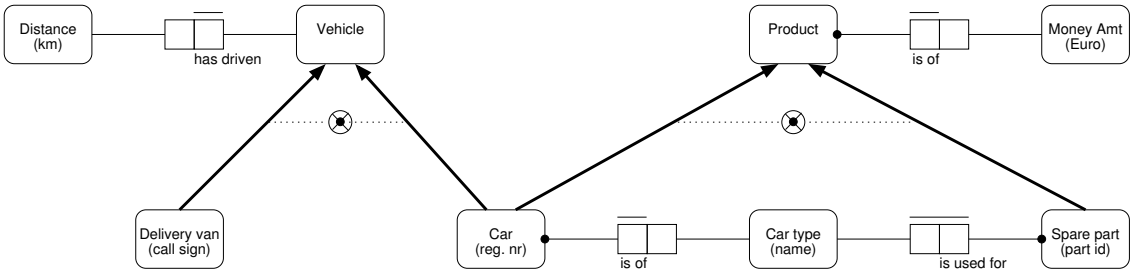


Figure 4.3: Example of a multi-rooted specialization hierarchy

A further interesting case of specialization is depicted in Figure 4.3. It illustrates how specialization hierarchies can have multiple roots. This example also introduces two important classes of constraints: exclusiveness and totality of specializations. If T is a set of types, then we can define:

$$\mathsf{Exclusive}(T) \triangleq \forall_{x,y \in T} [x \neq y \Rightarrow \mathsf{Pop}(x) \cap \mathsf{Pop}(y) = \emptyset]$$

If T is a set of types with a common supertype s, such that $\forall_{t \in T} [t \,\mathsf{Sub}\, s]$, then we can define:

$$\mathsf{Total}(s : T) \triangleq \mathsf{Pop}(s) = \bigcup_{t \in T} \mathsf{Pop}(t)$$

To express the semantics of totality in general we first need to identify all common supertypes:

$$\mathsf{CommonSuper}(T) \triangleq \{ y \mid \forall_{x \in T} [x \,\mathsf{Sub}\, y] \}$$

If T is a set of types such that there is some common supertype ($\mathsf{CommonSuper}(T) \neq \emptyset$), then we can define:

$$\mathsf{Total}(T) \triangleq \forall_{s \in \mathsf{CommonSuper}(T)} [\mathsf{Total}(s : T)]$$

Based on [HW93] a graphical abbreviation can be used for this kind of sub-typing. This is depicted in Figure 4.4. The right-hand side provides an abbreviation for the situation depicted in the left-hand side.

4.2 Overlap of populations

When considering the ORM schema as depicted in Figure 4.3, one would expect the populations of Vehicle and Distance to be disjoint, while the populations of Vehicle and Product are expected to overlap.

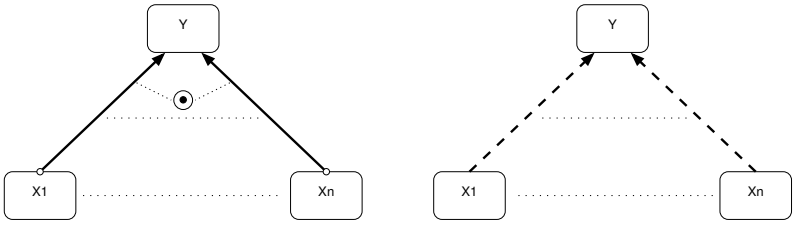


Figure 4.4: Generalization

Thus far, we have not introduced any formal mechanism to enforce this type of behavior other that the inclusion of populations for sub-types. To properly formalize this, we first define the family (as it is 'alive' at some point in time) of an object type based on the sub-typing hierarchy as follows:

$$\mathsf{Family}_t(x) \triangleq \{y \in \hat{\mathcal{OB}}_t \mid y \,\mathsf{Sub}\, x \vee y = x\}$$

Two object types are deemed *type related* iff their families overlap:

$$x \sim_t y \triangleq \mathsf{Family}(x) \cap \mathsf{Family}(y) = \emptyset$$

If the population of object types overlaps, then they mus be type related:

[S45] $\mathsf{Pop}_t(x) \cap \mathsf{Pop}_t(y) \neq \emptyset \Rightarrow x \sim_t y$

Note that the converse does not necessarily hold.

4.3 Abstraction

To introduce abstraction (*schema decomposition*), we start with an example domain taken from [CP96].

For our example domain, we consider a bank. Figure 4.5 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 bank is identified by a name. Basically, the use of the (name) suffix is a graphical abbreviation of the schema fragment depicted in Figure 4.6. 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).

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 4.7. 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

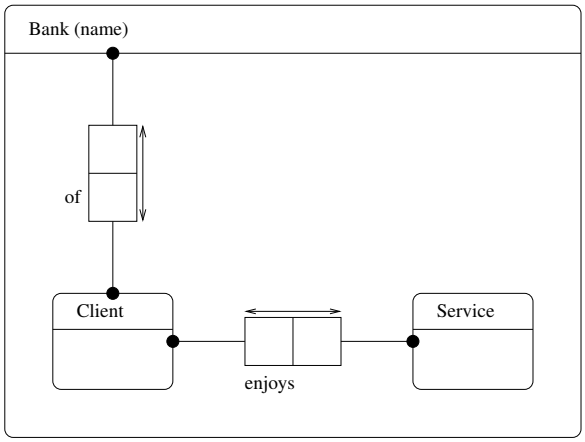


Figure 4.5: The top diagram of the Bank domain

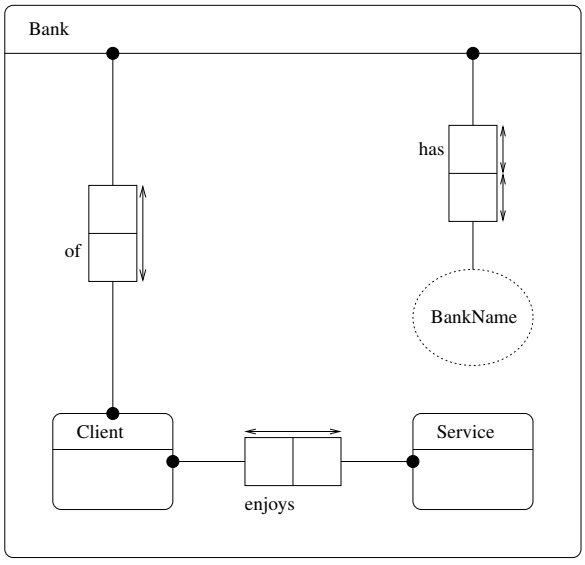


Figure 4.6: Fully detailed top diagram

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.

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.

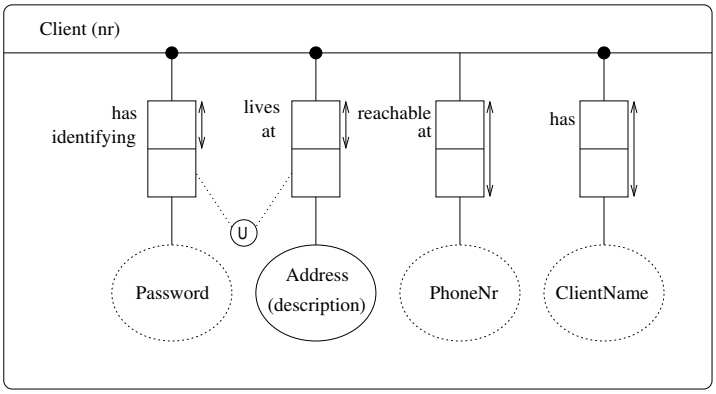


Figure 4.7: Refinement of the client type

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 4.8. 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 modeling 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 representation, 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.

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

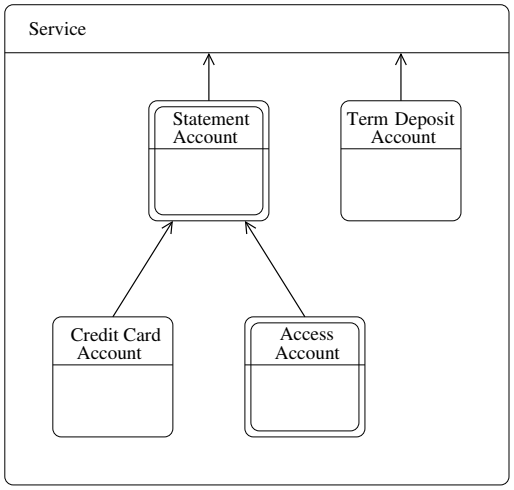


Figure 4.8: Refinement of the service type

are shown in Figure 4.9. 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.

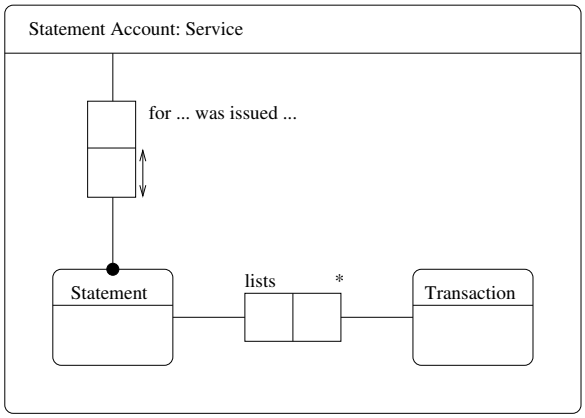


Figure 4.9: Refinement of statement account

One of the key features of the fact based modeling 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, properties 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 4.10. 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.

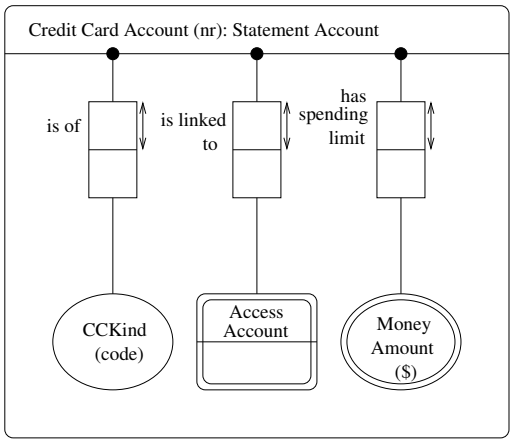


Figure 4.10: Refinement of the credit card type

For Access Account, the details are shown in Figure 4.11. 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.

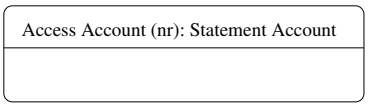


Figure 4.11: Refinement of an access account

Figure 4.12 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 contextual 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.

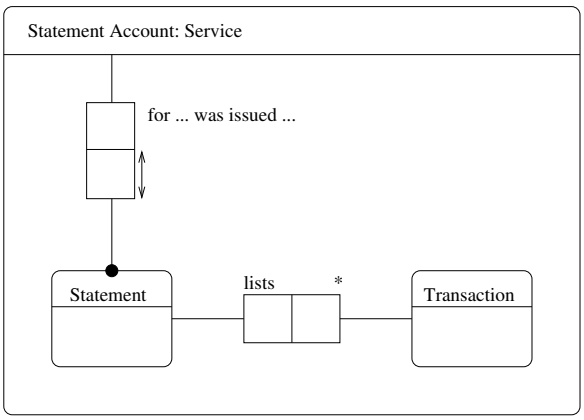


Figure 4.12: Refinement of a statement

The refined view on a transaction is shown in Figure 4.13. 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.

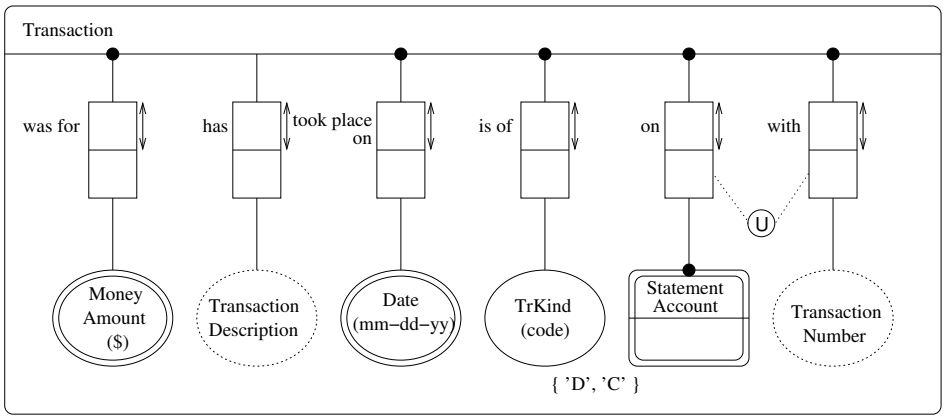


Figure 4.13: Refinement of a transaction

Term deposits form a world on their own. This is elaborated in Figure 4.14. 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 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 4.15. A Long Term Deposit is a term deposit with a duration of more than 60 days. In Figure 4.16 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 4.17.

This completes the schema of the example domain. When modeling 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 4.18 by uniting all clusters. This is, however, still not the 'lowest' level at which an ORM diagram can be displayed, since we have used the standard abbreviations for simple identifications and the short notation for objectifications. Objectification is a concept we have not yet discussed in this paper.

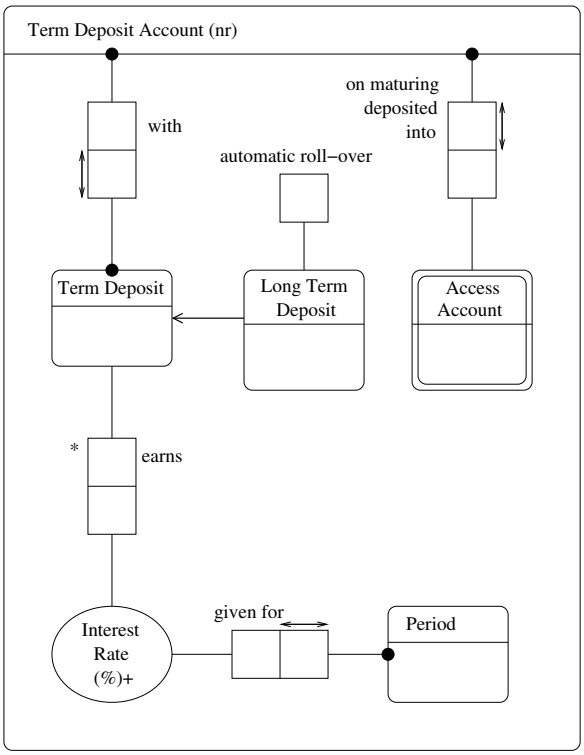


Figure 4.14: Refinement of a term deposit account

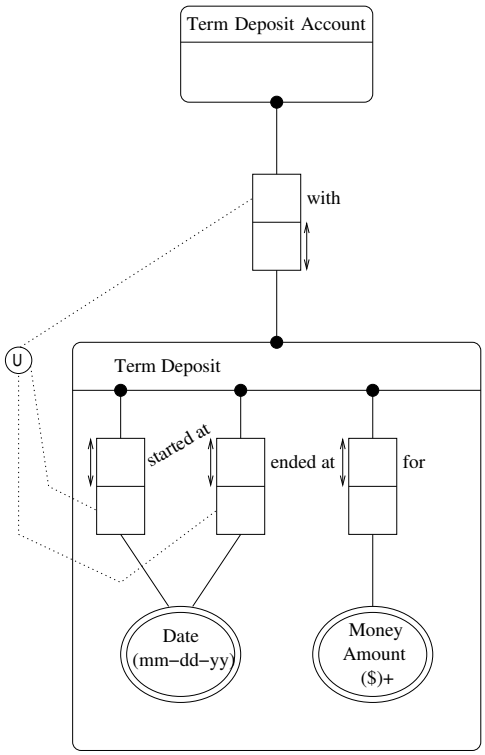
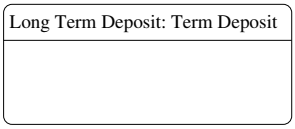


Figure 4.15: Refinement of a term deposit



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

Figure 4.16: Refinement of a long term deposit

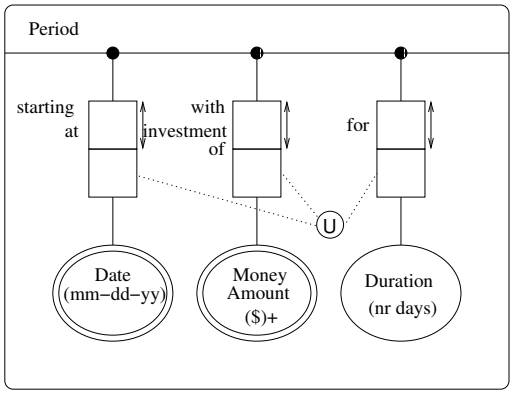


Figure 4.17: Refinement of periods

Also when looking at a design procedure for ORM schemas as presented in [Hal95] the decision to model a Transaction, say, as an objectification or a flat entity type is based on considerations of abstraction. When, for the modeling 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 [HW94, HW97] 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 4.18 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.

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 4.19. The version we used there is based on the one discussed in [BCN92]. Differing extended ER versions use different notations for this concept [EWH85, EN94, EGH$^+$92]. The names for attributes in this diagram are simply based on the verbalizations given in the ORM schema. For most ER modelers, the concept of using elaborate verbalizations is new. One could allow for the specification of specific attribute names to, for example, abbreviate with minimum deposit of MoneyAmount ($)+ to MinDeposit. In this article we do not discuss naming conventions in detail but rather focus on the underlying conceptual issues. In [BBMP95] 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

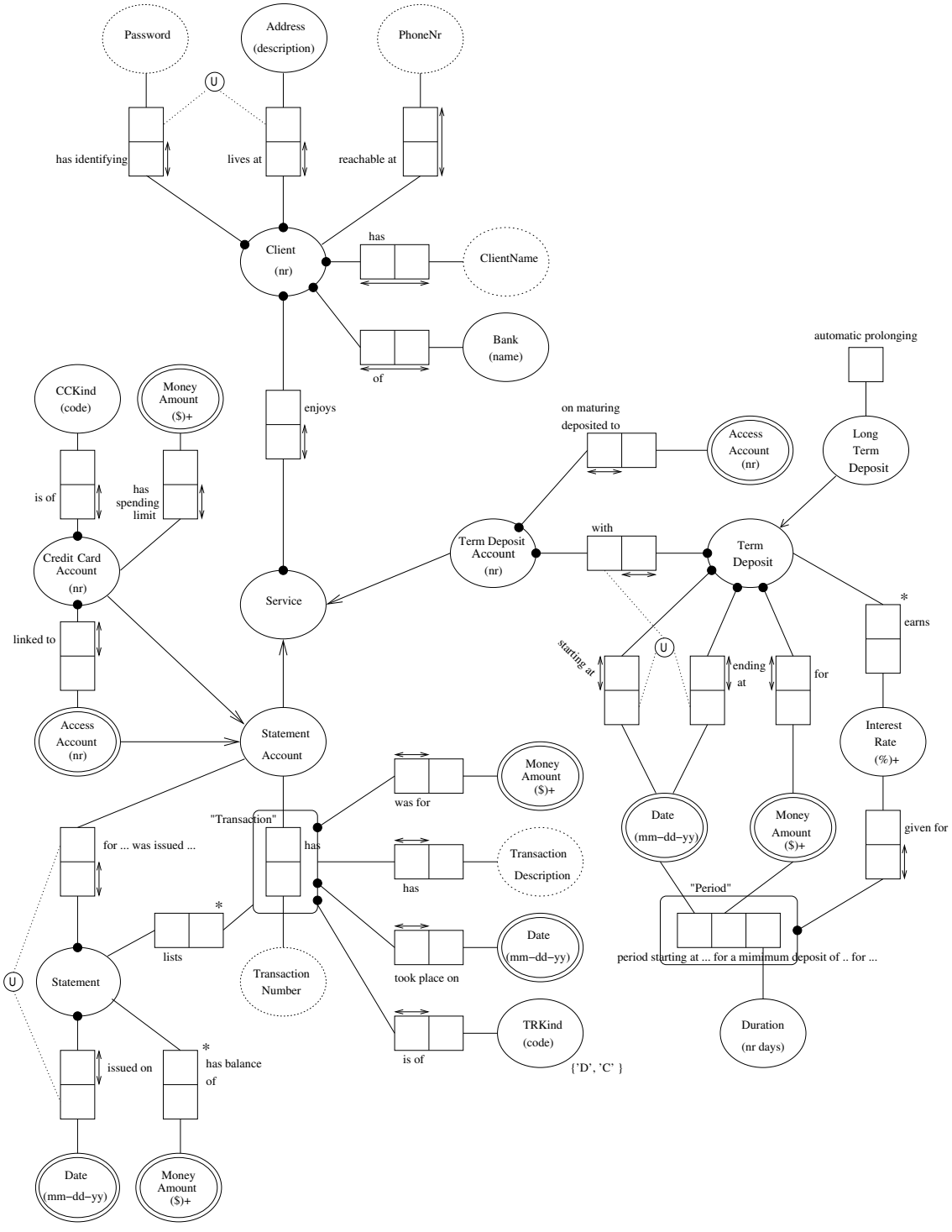


Figure 4.18: Complete diagram of the Bank domain

different concepts underlying these modeling techniques are related, together with a mapping of the (graphical) concepts between the two classes of data modeling techniques.

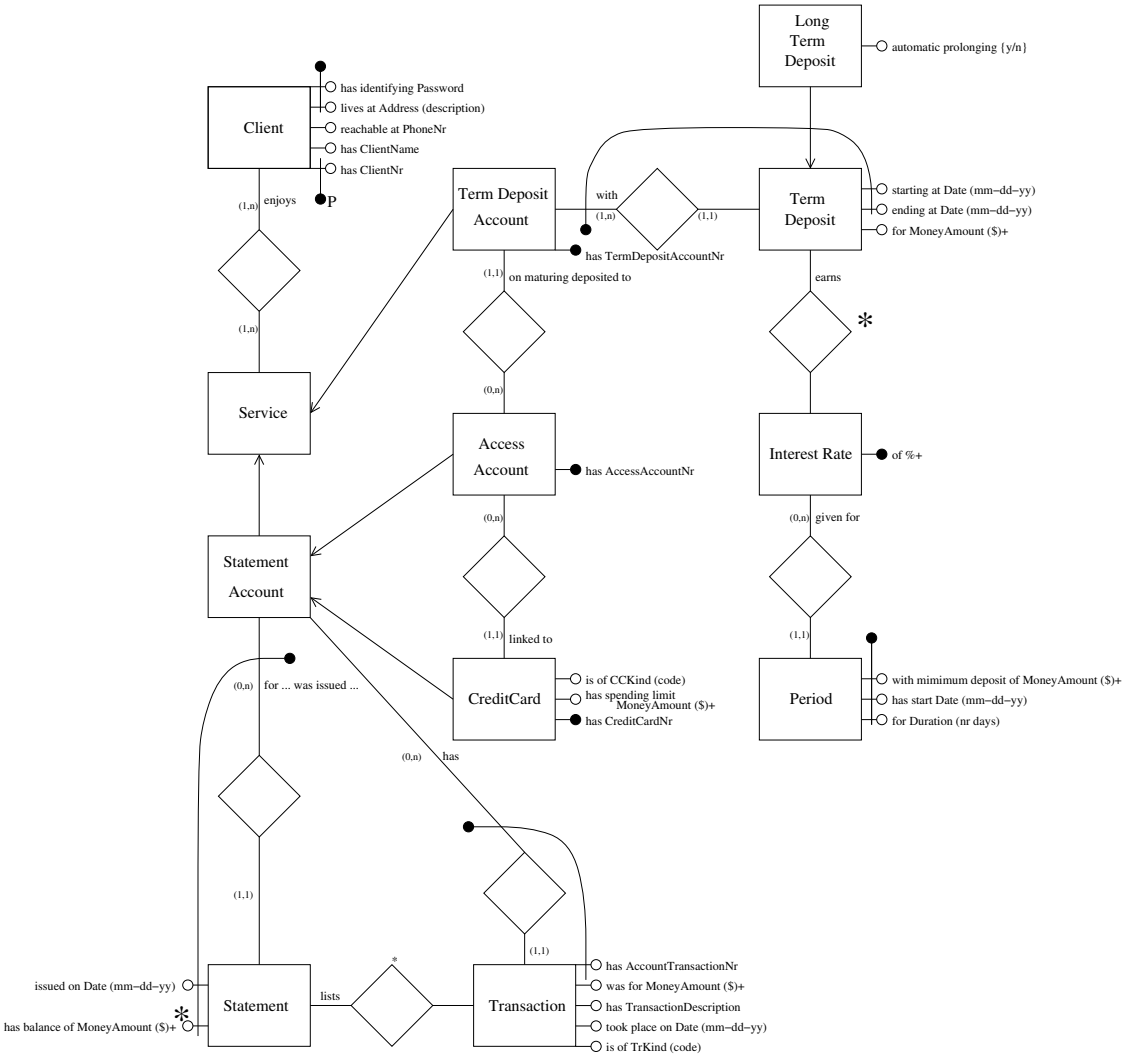


Figure 4.19: Complete ER diagram of the Bank domain

4.4 Set types

In set theory we use $\wp(X)$ to denote the set consisting of all subsets of X (see for instance [Lev79]). When modeling 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 [AH87]). An illustrative example, taken from [HW94], is shown in Figure 4.20. 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:

$$\forall_i \left[i \in X \Leftrightarrow i \in Y \right] \Rightarrow X = Y$$

In Figure 4.20 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.

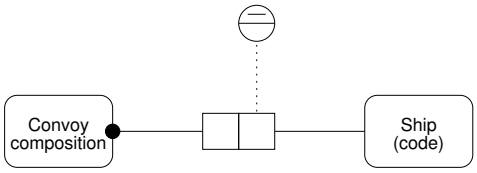


Figure 4.20: Convoy of ships

Formally, we introduce existential uniqueness by first identifying the variety an instance of a fact type may have with regards to a set of roles. Let $f \in \hat{\mathcal{FC}}$ be a fact type and $R \subseteq \mathsf{RolesOf}(f)$ be a set of roles involved in this fact type. The variety of instances $i \in \mathsf{Pop}(f)$ with regard to their roles R is defined as:

$$\mathsf{Variety}(i, f, R) \triangleq \{ j[R] \mid j \in \mathsf{Pop}(f) \wedge j[\,\overline{R}\,] = i[\,\overline{R}\,] \}$$

For a set of roles R of some fact type f, we can now express the existantial uniqueness constraint as:

$$\mathsf{ExtUnique}(f : R) \triangleq \forall_{i,j \in \mathsf{Pop}(f)} \left[\forall_e \left[e \in \mathsf{Variety}(i, f, R) \Leftrightarrow e \in \mathsf{Variety}(j, f, R) \right] \Rightarrow j[\,\overline{R}\,] = i[\,\overline{R}\,] \right]$$

where $\overline{R} = \mathsf{RolesOf}(f) - R$. Note the correspondance to existantiality from set theory. A more compact form (which we are allowed to use due to the existantiality axiom from set theory) would be:

$$\mathsf{ExtUnique}(f : R) \triangleq \forall_{i,j \in \mathsf{Pop}(f)} \left[\mathsf{Variety}(i, f, R) = \mathsf{Variety}(j, f, R) \Rightarrow j[\,\overline{R}\,] = i[\,\overline{R}\,] \right]$$

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 4.21.

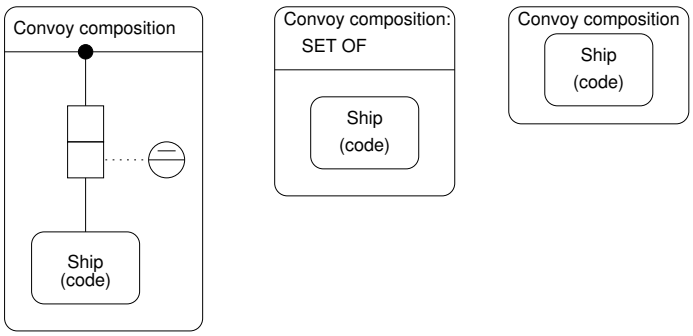


Figure 4.21: Shorthand notations for convoys of ships

4.5 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 4.22. In the depicted domain, a train composition class is defined as a multi-set of types 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 4.23.

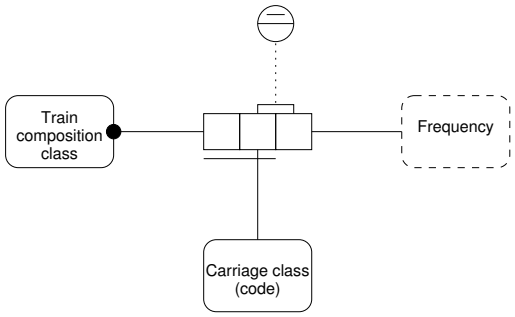


Figure 4.22: Train composition classes

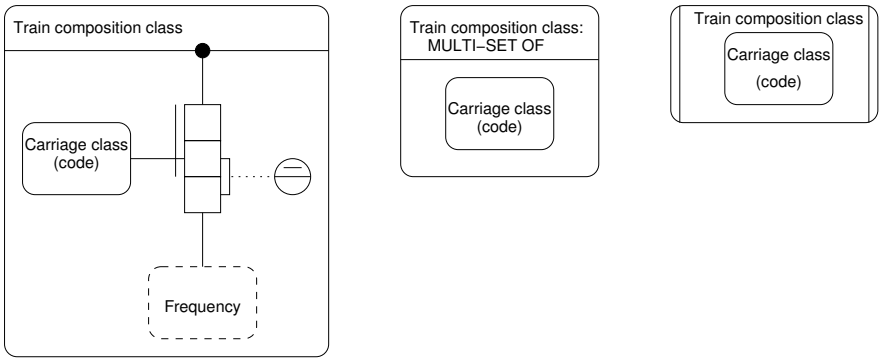


Figure 4.23: Shorthand notations for train composition classes

4.6 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 4.24.

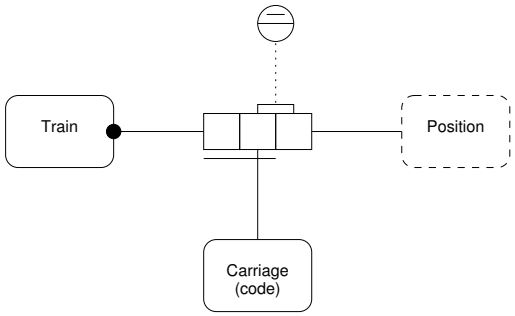


Figure 4.24: Train as a sequence of carriages

Using the abstraction mechanism, we are again able to more introduce a number of shorthand notations for sequence types. These are depicted in Figure 4.25.

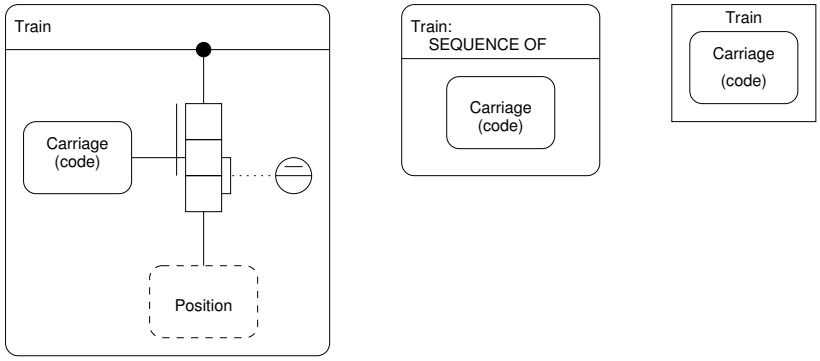


Figure 4.25: Shorthand notations for trains as sequences of carriages

4.7 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 4.26.

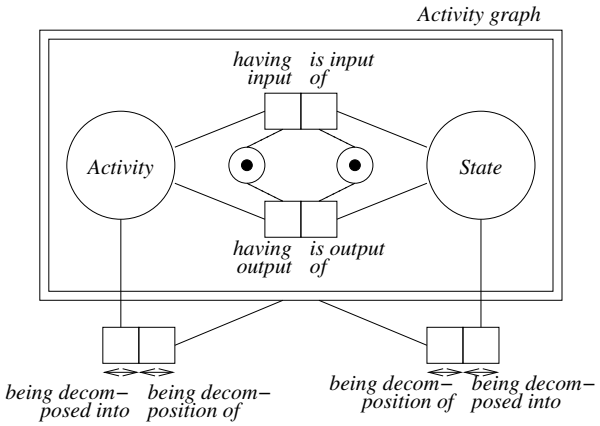


Figure 4.26: Activity graphs usign a schema type

Using the abstraction mechanism, we are again able to more introduce a number of shorthand notations for schema types. These are depicted in Figure 4.27.

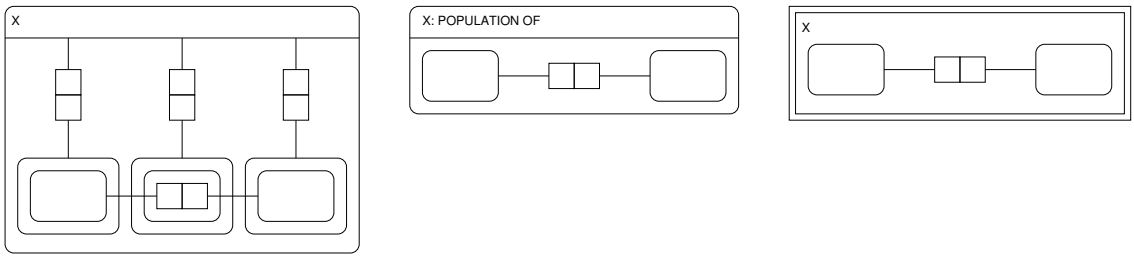


Figure 4.27: Shorthand notations for schema types

Questions

1. 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?
2. 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.

Bibliography

[AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[BBMP95] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, and H.A. (Erik) Proper. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3):213–235, 1995.

[BCN92] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin Cummings, Redwood City, California, 1992.

[CP96] P.N. Creasy and H.A. (Erik) Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering*, 20(2):119–162, 1996.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.

[EN94] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, Redwood City, California, 1994. Second Edition.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1:75–116, 1985.

[Hal95] T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition, 1995.

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1558606726

[HP95] T.A. Halpin and H.A. (Erik) Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[HW94] A.H.M. ter Hofstede and Th.P. van der Weide. Fact Orientation in Complex Object Role Modelling Techniques. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 45–59, Townsville, Australia, July 1994.

[HW97] A.H.M. ter Hofstede and Th.P. van der Weide. Deriving Identity from Extensionality. *International Journal of Software Engineering and Knowledge Engineering*, 8(2):189–221, June 1997.

[Lev79] A.Y. Levy. *Basic Set Theory*. Springer-Verlag, Berlin, Germany, 1979.

Chapter 5

The Act of Modelling

Version: 13-04-05

In this chapter, which is based on the work reported in [PBH04, HBP05], we turn to the question *how to model a domain*; a question to which there is no simple, one-size-fits-all answer.

5.1 What to model?

- Modeling goal
- Intended audience
- Viewpoint

5.2 The modeling challenge

5.2.1 Goal-bounded and communication-driven

Some modeling approaches, such as NIAM [NH89] and ORM [Hal01], suggest or prescribe a detailed procedure. Practice shows, however, that experienced modelers frequently deviate from such procedures [Ver93]:

> 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 modeling procedure should be done with some caution. While a predefined modeling procedure should never become "an excuse to stop thinking", situational specificity should not become an excuse for taking an ad-hoc approach to the modeling effort. A more stable anchor is needed upon which modelers can base themselves when making decisions during the modeling process. We believe that domain modeling requires a goal-bounded and communication-driven approach. With goal-bounded we hint at the fact that when modeling a domain, a modeler is confronted with a plethora of modeling decisions. These decisions range from the modeling 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 modeled as a relationship, or is it an object on its own?

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

In most situations where a domain needs to be modeled, the modeler cannot merely passively observe the domain. Modelers 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, modelers 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 modeling 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 modeling process can thus be seen as a communication-driven process [FW04b, VHP03]. The principles of natural language driven modeling approaches [NH89, EKW92, Kri94, Hal01] can be used as a basis for shaping the communication process between informer and modeler.

5.2.2 Aspects of a method

When considering a modeling approach or method, several aspects thereof can be discerned [SWS89, WH90]. An important distinction to be made is that between a product oriented perspective and a process oriented perspective. In terms of the framework presented in [SWS89, WH90] these are referred to as the *way of modeling* and *way of working*, respectively:

Way of modeling – The way of modelling provides an abstract description of the underlying modelling concepts together with their interrelationships and properties. It structures the models which can be used in the information system development, i.e. it provides an abstract language in which to express the models.

Way of working – The way of working structures the way in which an information system is developed. 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.

In the case of domain modeling, the *way of working* represents the process followed when modeling a domain. In the following sections, we will mainly elaborate on this aspect. The *way of modeling* used for domain modeling is likely to be prescribed by a diagramming technique such as ORM diagrams [Hal01], ER diagrams [Che76] or UML class diagrams [BRJ99].

5.2.3 The process of modeling

In general, the goals underlying (business) domain modeling are [BPH04]-

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

Based on the results reported in [Hop03], we consider domain modeling 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)
- 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 *conceptualization processes* [Hop03]. When, as in the context of software development, conceptualization is performed deliberately, as a specific task and with a specific goal in mind, it is referred to as an *explicit conceptualization process*. The above mentioned stream of activities called concept specification is such an explicit conceptualization process. In [Hop03, BPH04] a reference model for conceptualization processes is provided. This reference model distinguishes five streams of activities or *phases*:

Assess domain and acquire raw material – Domain modeling 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.[1]

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

[1] In an earlier version of this framework, this was referred to as *scoping the universe of discourse*.

5.3 Ambition levels for modeling

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

Singular – This level of ambition corresponds to the modeling approaches as described in e.g. NIAM [NH89] and ORM [Hal01]. It involves the modeling of a *single* environment of discourse based on complete input; usually in terms of a *complete* verbalization of (*only*) the relevant parts of the domain.

Elusive – At this level of ambition, modelers need to cope with the unavoidable iterative nature of the modeling process. As a modeling 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 [PH04].

Evolving – The final ambition level recognizes the fact that domains themselves are not stable; they evolve over time [PH04]. 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 modeling at the *singular*, *elusive* and *pluriform* levels of ambition. The *evolving* level is omitted for now.

5.4 Meeting the challenge

This section aims to discuss the domain modeling 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 modeling as introduced in the previous section.

5.4.1 Modeling a singular domain

At this level of ambition we are only interested in the modeling of a single environment of discourse based on complete input. In terms of the above framework for domain modeling, 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 modeling approaches like NIAM [NH89] and ORM [Hal01] concern elaborately described ways of executing a domain modeling process at this ambition level. For example, the modeling procedure as described in ORM [Hal01] identifies the following steps:

Step 1 – Transform familiar examples into elementary facts This step involves the verbalization 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. sales-price = retail-price + 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 modeling 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 modelers.

Even though the above order is very explicit, and therefore well suited for educational purposes, a goal-bounded approach to domain modeling requires a more refined view. The key question concerns the *goal* for which a domain is modeled. 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, modelers are likely to skip steps 6 and 7. The modeling procedure as discussed in [Hal01], also requires modelers 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 modelers 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. [Ver93, page 161].

> The order in which the various modeling tasks are performed differs to a large extent. A clear distinction exists between prescribed modeling knowledge and applied modeling knowledge, in this respect. Whereas an almost strictly *linear* order of performing modeling 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 modeling 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 [EKW92] the Object-Oriented Systems Analysis method is presented. It uses a natural-language based approach to produce an Object-Relationship Model (accidently 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 modeling*, however, has a more sketchy nature and has been worked out to a lesser degree. The KISS approach, as reported in [Kri94], 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 modeling concepts are introduced (*way of modeling*) covering a wide range of diagraming techniques (not unlike the UML [BRJ99]).

Independent of the approach used, a modeling process always needs to be flanked by a continuous communication process with the stakeholders [VHP03]. Communication brings along the aspect of documentation. Modeling 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 [FW02, Fre97]. In this respect we could argue that any modeling 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

Systems Modeling

Chapter 6

Natural-Language Foundations of Information-Systems Modeling

Version: 12-05-05

In this chapter, we will essentially use the ORM philosophy to model key aspects of information-systems. This requires the immediate introduction of some new concepts into our ontology, such as: agentive, experiencing , circumstantial and predicative role, action, predication, agent, subject and context elements. These concepts allow us to reason about such things as: *when* does something happens (triggering), *what* happens (action), *who/what* makes it happen (agentr), *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 modeling approach from the *Domain Modeling* course. Based on these annotated ORM models, we will be able to mechanically derive process models in a modeling notation (ArchiMate [Lo05]) that is particularly suited for the modeling of business processes. This will be the focus of the *next* chapter.

6.1 Classes of roles

We consider organizational systems, i.e. open active systemor work-systems:

- So there is activity going on.
- In other words, there are active elements.
- A specific class of concepts should therefore be: actions.
- Even more, these actions are performed by *agents*, and are performed on *subjects*. We want to be able to 'talk' about these these agents and subjects.
- In other words, we actually need three additional classes of concepts: actions, agents and subjects.

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. All players in the facts describing the above domain are players in this domain. What are the actions, agents and subjects? Several degrees of activeness exist with regards to the role player plays in a fact/domain:

- Some roles will be more active than others.
- 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.

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 takes place.

Four 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 where the player is regarded as being a predicate on some other player.

In the example domain: the name Erik.

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*, *context elements*, and *predicators* respectively.

In the example domain, the writing of the letter by the person can be regarded as a key activity in the domain. In other words, writing is an action, while the person is the agent and the letter is the subject. We may regard the cat and the loved one as a subject as well. What about the pen, the name Erik, the desk, etc? They are really players in *predications* over the other players. The fact that a pen is used by the person to write the letter is a *predication* of the writing *action*.

If we were to zoom in on a sub-domain of the above sketched domain, we could actually find that what was a subject in the super-domain is an agent in the sub-domain. Consider, for example, the sub-domain:

[The writing of this letter by Erik], is done while [the cat] is watching

When considered in isolation, one may quite easily argue that the primary action here is the watching, which is something that is being done by the cat. This really makes the cat into an agent rather than the subject, while the thing that is being watched (the writing) becomes the subject. This really means that our notions of agent, subject, action and predication are really to be taken *relative* to the domain under consideration, which is in line with the subjective approach to modeling as taken in this textbook.

Formally, we presume the existance of sets $\mathcal{AR}, \mathcal{ER}, \mathcal{CR}, \mathcal{PR} \subseteq \mathcal{RO}$ with agent and subject *roles* respectively. These classes of roles form a partition of the roles:

[S46] $\mathcal{AR}$, $\mathcal{ER}$, $\mathcal{CR}$ and $\mathcal{PR}$ are a partition of $\mathcal{RO}$.

With this we can also define the sets of actions and predications, respectively, as:

$$\begin{array}{lcl} \mathcal{AN} & \triangleq & \{\mathsf{Fact}(r) \mid r \in \mathcal{AR}\} \\ \mathcal{PN} & \triangleq & \mathcal{FC} - \mathcal{AN} \end{array}$$

In other words, all facts that have an agentive role are regarded as actions. The other facts are (pure) predications.

Typing should adhere our refined ontology. In other words:

[S47] For all $X \in \{\mathcal{FC}, \mathcal{RO}, \mathcal{PC}\}$ we have:

$$x \,\mathsf{HasType}\, y \Rightarrow (x \in X \Leftrightarrow y \in X)$$

All experiencing roles must be involved in some action:

[S48] $\forall_{d \in \mathcal{ER}} [\mathsf{Fact}(d) \in \mathcal{AN}]$

For the example given above, we would have:

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

Finally, a word of warning:

- Natural language also harbors 'conceptual prejudice'
- Most, if not all, Indo-European languages are rather state oriented: 'Charles the bold' stands on 'the floor'
- Some North-American native languages are activity oriented: 'Dances with wolves' stands on 'what supports us when walking'
- Imagine the impact on the way we view & design systems/organizations: Is nature state oriented or a continuous flow of activities?

6.2 Activity types

Based on the above discussions regarding actions and predications, we can identify which roles are of which class, and mark this in the ORM model. For the example from Figure 3.10, we have the situation as depictedin Figure 6.1.

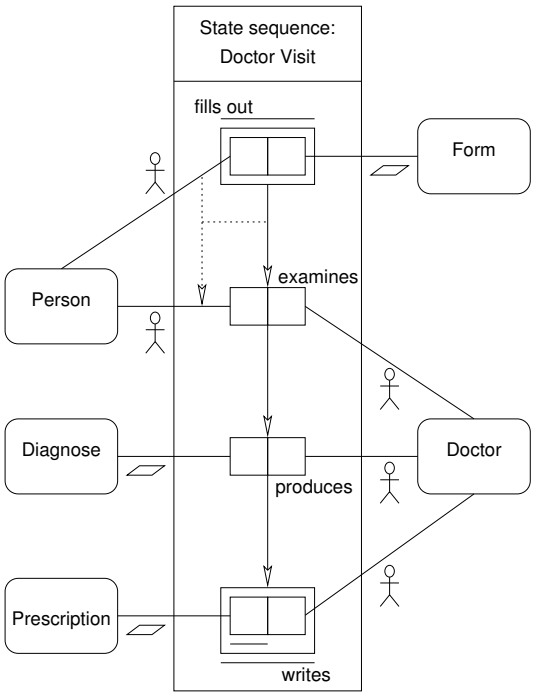


Figure 6.1: Compact model of a visit to a Doctor with alternative semantics

Those state-sequence types which comprise a number of action types (or other activity types) are considered to be (complex) *activity* types. Action types on their own are (atomic) *activity* types.

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 entities and relationships depicting this domain.

2. Consider the following domain:

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

   Wat zijn hier de entiteiten en de relaties? Wat zijn de acties, actoren, actanden en predications.?

3. Stel je maakt een ontwerp voor een geldautomaat. Wat zijn voor dat domein de belangrijkste systeem entiteiten en hun onderlinge relaties? Hoe werken ze samen? Wat zijn hier de entiteiten en de relaties? Wat zijn de acties, actoren, actanden en predications.?

4. Proof Corollary 3.4.1 (page 75).

5. Proof Corollary 3.5.1 (page 77).

6. Consider the case from Question 3.8.

   Answer the following questions:

   (a) (Re)produce elementary facts for this domain.
   (b) What are the actions, actors and actands?

Bibliography

[Lo05] M.M. Lankhorst and others. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, Berlin, Germany, EU, 2005. ISBN 3540243712

Chapter 7

Activity Modeling

Version: 14-04-05

Suitability! The situation as depicted in Figure 3.9 and Figure 3.10 may be highly explicit, but one may wonder if the resulting diagrams are still communicatable. Do they still convey the message?

7.1 Introduction

What do we model?

- Who are involved: Actor domain
- What activities do they perform: Activity domain
- What (information) objects are used/created: Actand domain
- AND: the coherence between these three domains

Modeling domains:

- Actor domain
  - Who is involved?
  - What are possible interactions?
  - Roles, resources, business functions, ...
- Activity domain
  - Activities
  - Interactions
  - Relationships
  - Structure/decomposition
- Actand domain
  - See ORM/DM
  - Advanced/complex types

Core concepts:

Use-case: An activity which is triggered by an external trigger.

"Paying the first month of Mr. Jones' pension."

Trigger: A change in the environment, that initiates a use-case.

"Mr. Jones from Bristol turns 65 years of age."

Use-case type: A class of similar use-cases.

"Paying the first month of people's pension."

Trigger type: A class of business triggers with similar treatment.

"People turning 65 years of age."

Actor An organizational entity that is capable of performing behavior

Delivers services to its environment

"The NIII providing courses to students"

Collaboration A temporary configuration of two or more actors resulting in a collective behavior

"Insured person and insurance company submitting/receiving a claim"

Can be regarded as a temporary of two role-playing actors into an ad-hoc actor, that plays a role providing the behavior of the collaboration

7.2 Basic modeling language

The notation is primarily inspired by ArchiMate [Lo05]. 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 [AH05].

The ArchiMate language is actually set up such that 'local' specializations can be made quite easily. In our case. a specialization is called for that honors our natural language and conception based approach.

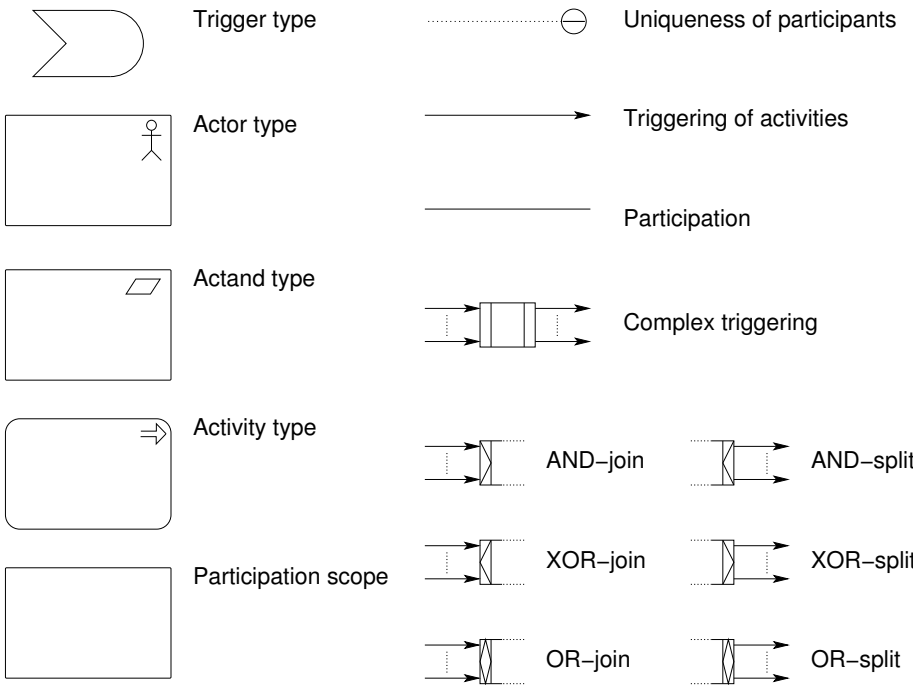


Figure 7.1: Activity modeling notation

A legend of the symbols used is depicted in Figure 7.1. When taking the example from Figure 3.10 and 6.1 and using this new notation, we obtain the situation as depicted in Figure 7.2 and 7.3 respectively.

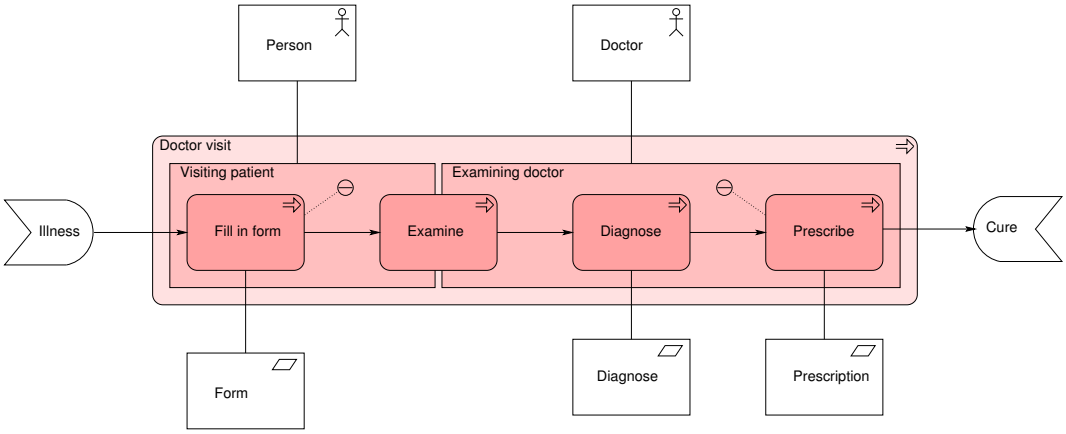


Figure 7.2: Visit to a Doctor in activity modeling notation

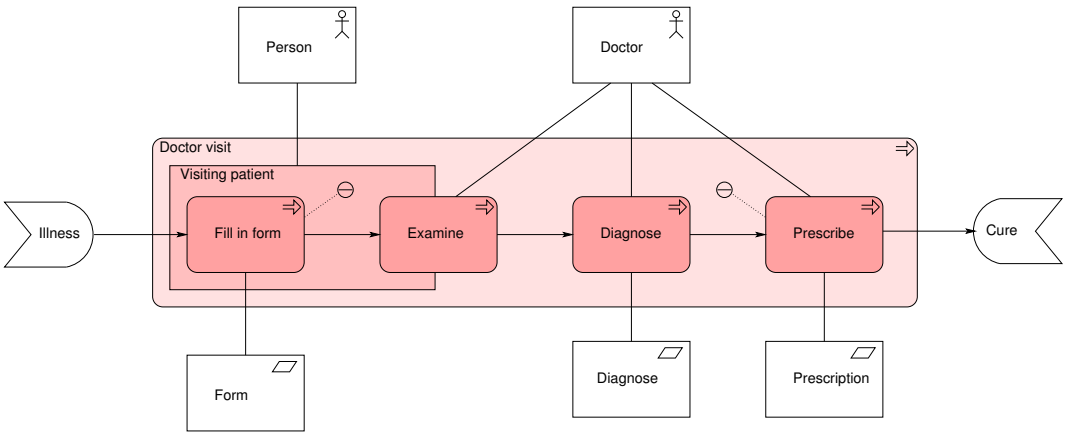


Figure 7.3: Visit to a Doctor with alternative semantics

7.3 Composed activities

Activities may be composed of yet other activities. This is illustrated in Figure 7.4.

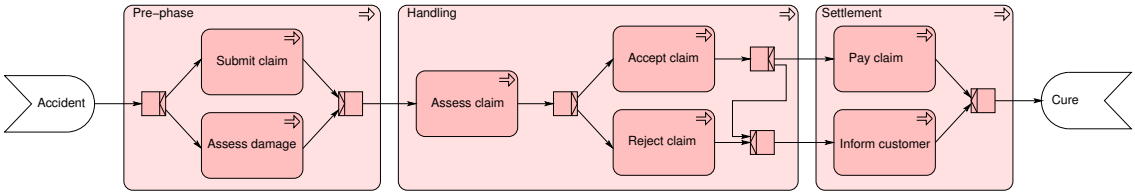


Figure 7.4: Claim handling

7.4 Petri-net based semantics

In order to gain a more precise understanding of activity 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 [AH05] for a more detailed formalization.

In Figure 7.5 the working of petri-nets are illustrated.

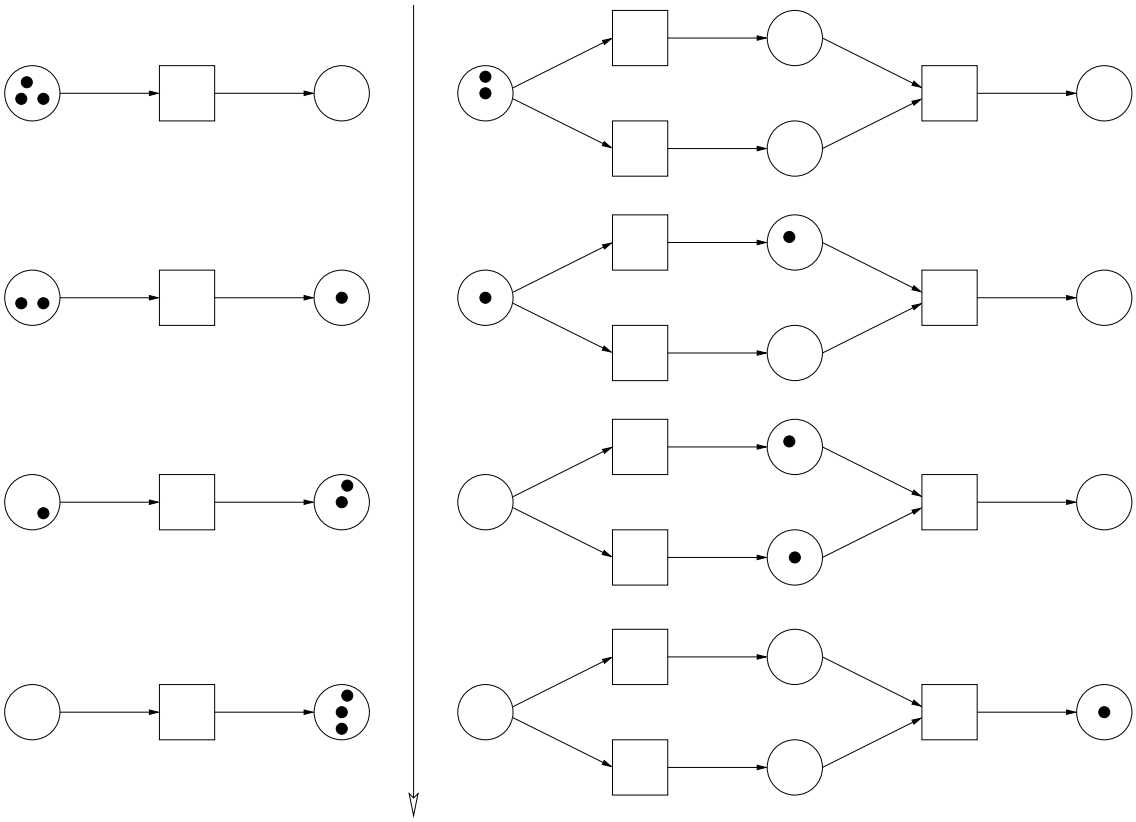


Figure 7.5: Petri-nets

In Figure 7.6, the mapping from complex triggering in activity models to petri nets is given. Note that the resulting petri-net will only be applicable for *one* run of the activity.

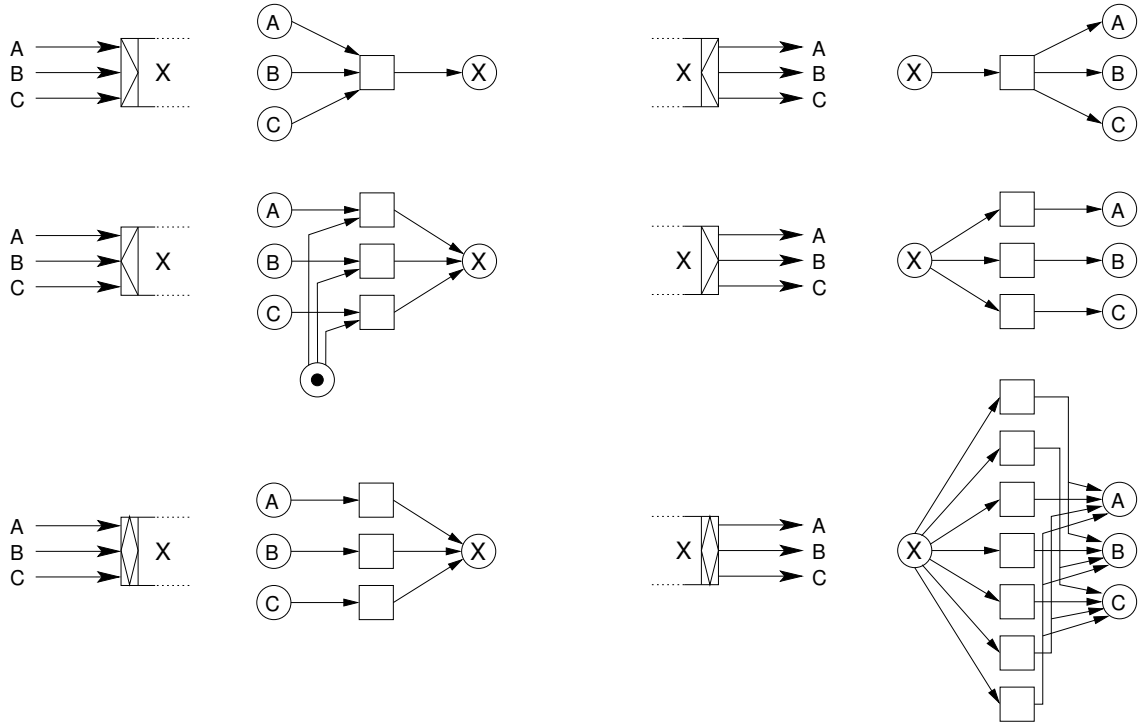


Figure 7.6: Mapping activity models to Petri-nets

An example translation is given in Figure 7.7.

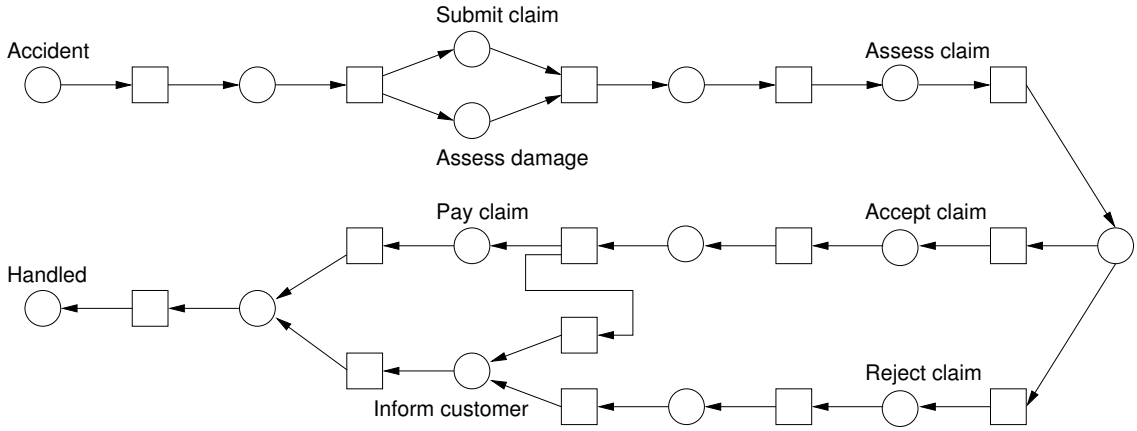


Figure 7.7: Example mapping of activity models to Petri-nets

Mapping the semantics of participations of participants (be it actors, actands, or others) to petri-nets can be done as illustrated in Figure 7.8 and 7.9. In this mapping we have actually used an extended version of petri-nets, called *colored* petri-nets. Note the subtle difference between the two versions with regards to the size of the scope and its impact on the mapping.

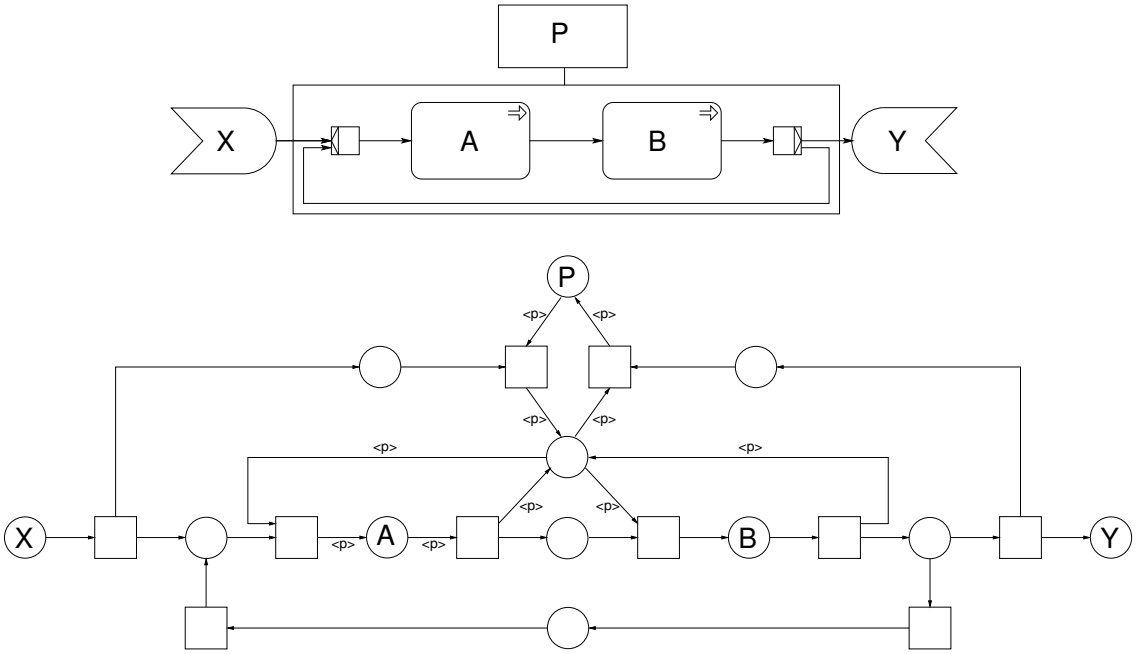


Figure 7.8: Mapping participation scopes

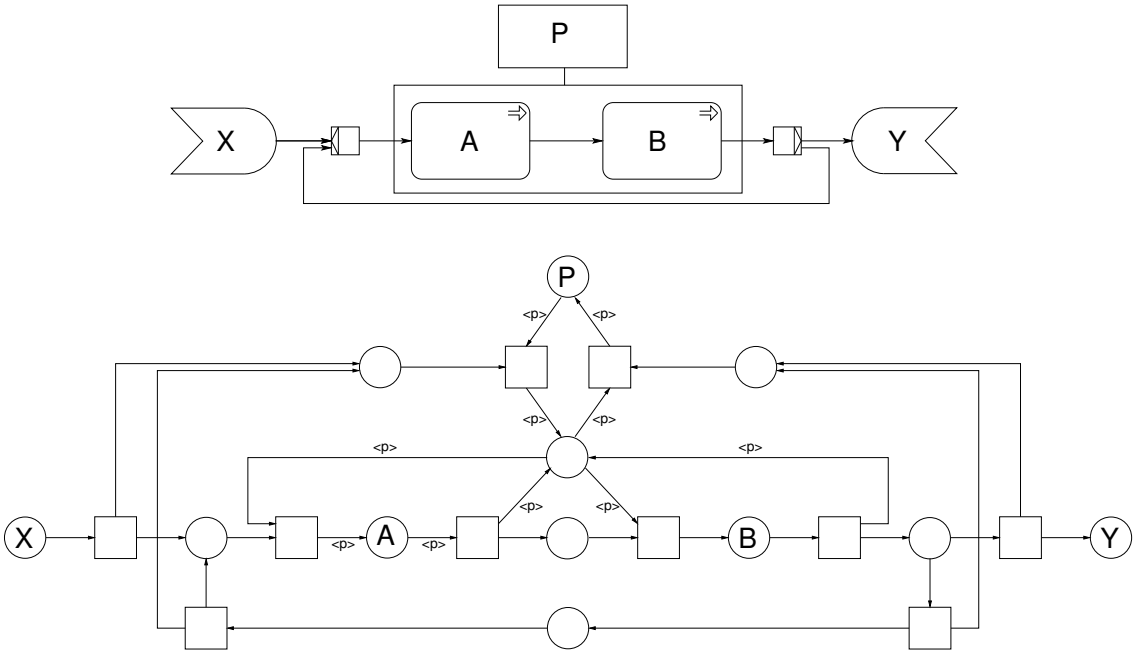


Figure 7.9: Looping and participation scopes

7.5 Quantative semantics

The petri-net based semantics of activity models focuses on the flow of specific cases through the activity. In addition to the petri-net based semantics, we identify a quantitative semantics. This is illustrated in Figure 7.10. The earlier used claim handling example can be made quantitative as depicted in Figure 7.11.

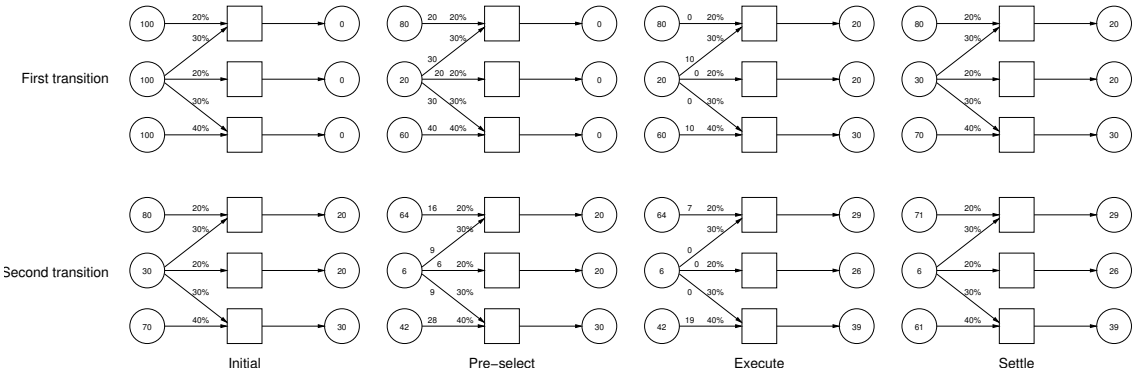


Figure 7.10: Quantitative analysis of activities

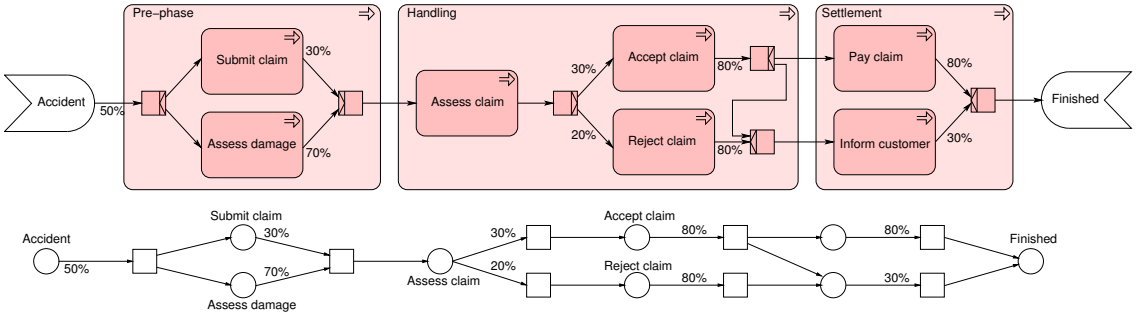


Figure 7.11: Claim handling with quantitative information

Uses for quantative analysis:

- Performance analysis.
- Optimization of design.
- Critical path analysis.

7.6 Mapping to UML 2.0 activity diagrams

In this section we look at the relationship between the activity modeling using the (enriched) ArchiMate notation and the activity diagrams from UML 2.0 [OMG03]. 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 7.4 can be represented by means of UML activity diagrams as illustrated in Figure 7.12.

The doctor visit example from Figure 7.2 and 7.3 respectively can be represented as the activity diagrams depicted in Figure 7.13 and 7.14

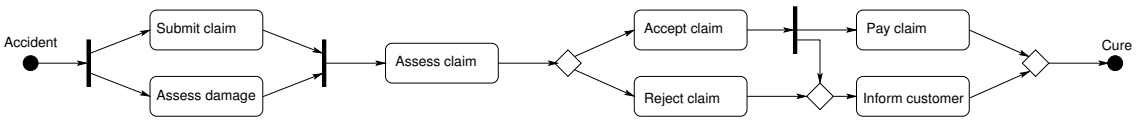


Figure 7.12: Claim handling as an UML 2.0 activity diagram

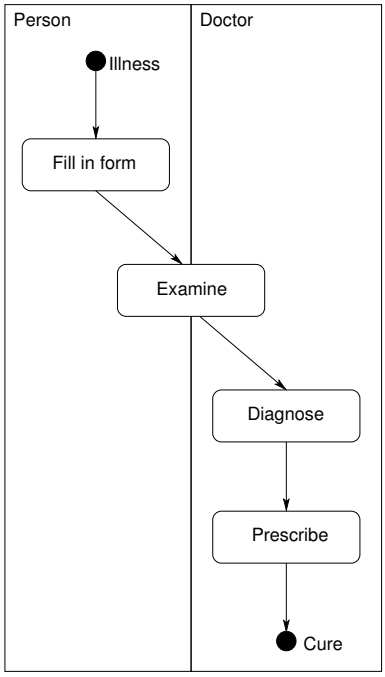


Figure 7.13: Doctor visit as an UML 2.0 activity diagram

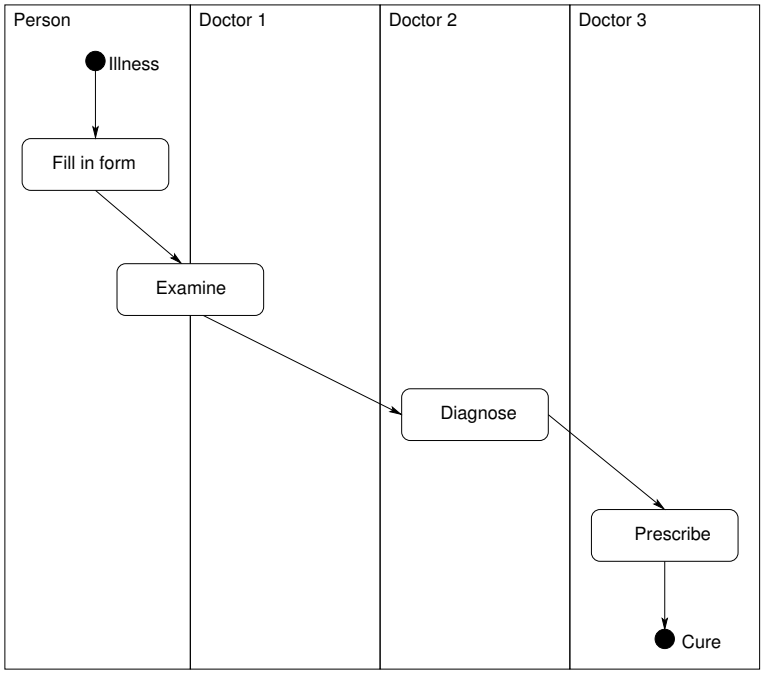


Figure 7.14: Doctor visit as an UML 2.0 activity diagram

7.7 Modeling approach

Outline

1. Identify key use cases
2. Describe key use cases
3. Compose initial model
4. Detail model
5. Re-examine models
6. Identify phases of activity

7.7.1 Identify key use cases

1. Determine the key entity types in UoD
2. Define for each of the key entity types, the key use case

7.7.2 Describe key use cases

1. Verbalise each use case
2. Single-out: predications, actions, actors, actands and activities.

7.7.3 Compose initial model

1. Relate actions, actors and actands. Use graphical notation.
2. Introduce activities as composed actions when needed.

7.7.4 Detail model

1. Select areas of intest
2. Gather detailed verbalisations
3. Compose detailed diagrams

7.7.5 Re-examine models

Re-examine the models in order to:

1. Identify and resolve unclear parts
2. Restructure processes/actors to better resemble the UoD

7.7.6 Identify phases of activity

1. Improve clarity of the models by introducing a higher level perspective on the domain.

Questions

1. 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 activity model voor.

2. Consider the domain as discussed in Question 3.8. Produce an activity model for this domain.

3. 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 gentroduceerde clubcard waarmee klanten kortingen bij alle aangesloten bedrijven kunnen krijgen. Inmiddels is er al een groot aantal klanten met een clubcard.

   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 credit card) 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 clubcard heeft en adviseert op basis van de klantbehoefte een auto uit een bepaalde tariefgroep. Hij controleert daarbij ook of er zoń 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 accoord 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 bijvoorbeeld een afkoop eigen risicio of een inzittenden verzekering. Dit wordt ook in de offerte opgenomen. Wanneer de klant ingaat op de offerte, dan maakt de verkoper een huurcontract nadat hij de credietwaardigheid 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 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 copie 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 accoord 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 zo'n 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.

Analyseer nu het hierboven beschreven domein en produceer hier een activity model voor.

4. Consider the claim handling process with quantative information as depicted in Figure 7.11. 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.

Bibliography

[AH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[Lo05] M.M. Lankhorst and others. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, Berlin, Germany, EU, 2005. ISBN 3540243712

[OMG03] OMG. UML 2.0 Superstructure Specification – OMG Draft Adopted Specification. Technical Report ptc/03-08-02, August 2003.
`http://www.omg.org`

Chapter 8

Resource Modeling

8.1 Actor modeling

As discussed before, we have the following modeling domains:

- Actor domain
  - Who is involved?
  - What are possible interactions?
  - Roles, resources, business functions, ...
- Activity domain
  - Activities
  - Interactions
  - Relationships
  - Structure/decomposition
- Actand domain
  - See ORM/DM
  - Advanced/complex types

The previous chapter focussed on the activity domain. This chapter promissed to discuss the actor and actand domains. Thus far, however, we have been "sharpening our axe" by discussing complex types. Actors and actands are usually complex types from a domain modeling perspective.

For the actor domain, we (re-)introduce the following core concepts:

Actor – An entity that performs behavior. I.e. it is actively involved in an activity.

Actor type – An entity that performs behavior. I.e. it is actively involved in an activity.

Use-case (instance) – An activity which is triggered by an external trigger.

Use-case type – A class of similar use-cases.

Process – Synonym for use-case type.

Business use-case (instance) – A use-case which an organisation/business executes to fulfill a service to the outside world.

Business use-case type – A class of similar business use-cases.

Business process – Synonym for business use-case type.

Business role (type) – A named specific contribution that an actor (type) may provide to a business use-case (type).

Is a special kind of role (type). Such a role is usually defined by functional responsibilities.

(Business) role-player (type) – An actor (type) playing a (business) role (type).

Business function – A virtual or idealised organisation within a business/organisation.

Identification can be based on different criteria: knowledge/skills required, use of resources, type of functional behaviour, etc.

(Business) collaboration (type) A temporary configuration of two or more actors (types) resulting in a collective behavior.

"Insured person and insurance company submitting/receiving a claim"

Can be regarded as a temporary of two role-playing actors into an ad-hoc actor, that plays a role providing the behavior of the collaboration

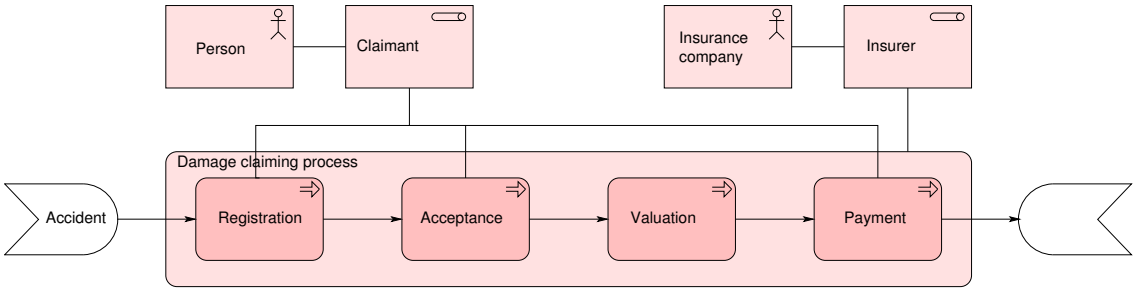


Figure 8.1: Claim handling process with business-roles

Figure 8.1 shows an example activity model with associated actor types and role types. The underlying ORM domain model is depicted in Figure 8.2.

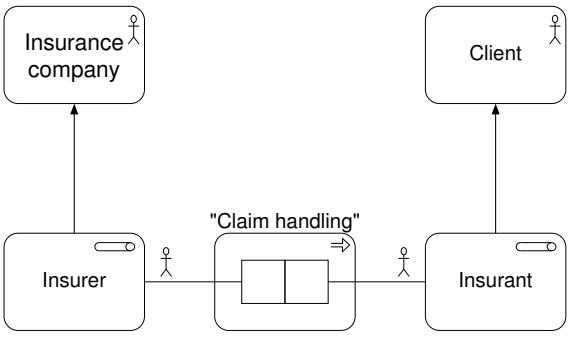


Figure 8.2: Claim handling process with business roles mapped to ORM

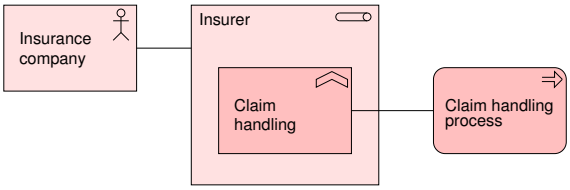


Figure 8.3: Claim handling process with business-functions

Figure 8.3 shows an example activity model where the business-function within the insurer is shown wich is responsible for the actual claim handling. The underlying ORM domain model is depicted in Figure 8.2. The refined ORM domain model is depicted in Figure 8.4.

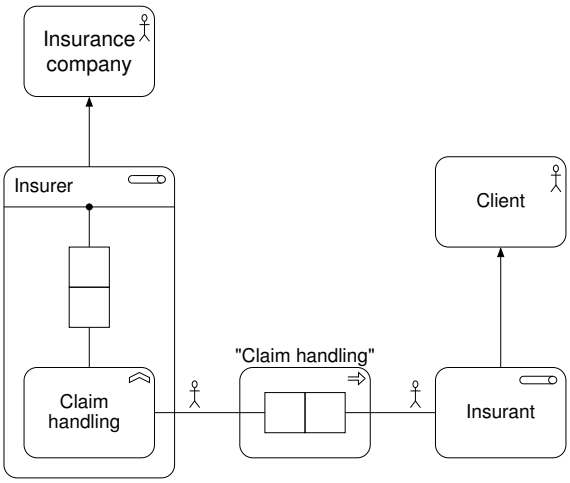


Figure 8.4: Claim handling process with business-functions mapped to ORM

8.2 Actand modeling

For the detailed modeling of actands, we essentially use ORM. This is illustrated in Figure 8.5.

Questions

1. 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?
2. 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.

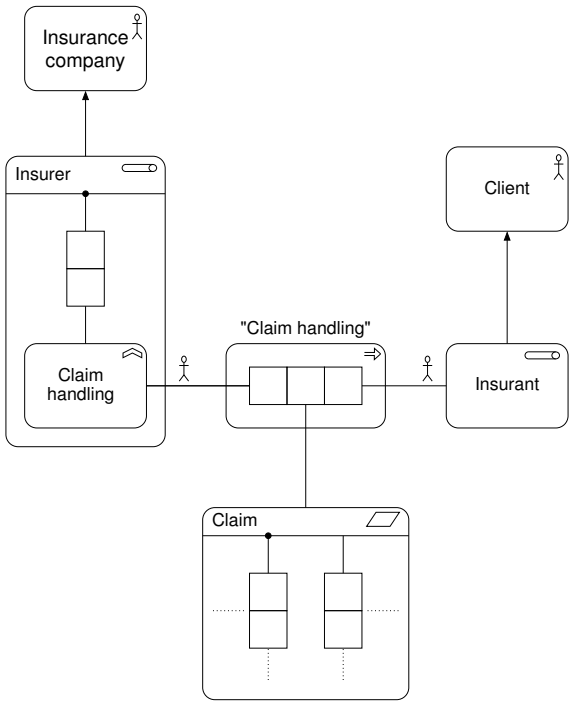


Figure 8.5: Refining actands for claim handling process

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 9

Service modeling

Version:
24-05-05

9.1 Service

- Service: The net effect (added value) of an actor's behaviour that is regarded to be useful (value!) by at least one actor.

  Examples:

  - Service your car
  - Provide you with a mortgage
  - Do yourself a favor, and ....

- Service offering: The potential of an actor to be engaged in an activity, where this engagement may be perceived as usefull by some other actor.

  Note: both of these actors might be complete (open active) systems!

  Is at the type level!

- Service delivery: The actual engagement.

  At the instance level!

- An information system can be regarded as a service providing actor:
  - Environment ⇒ Consumes services
  - System ⇒ Delivers services

- The system may use services offered by other systems in order to deliver the service. This "other system" then become "infrastructure" to the original system.

  The latter system is adding value to the basic services offered by the "infrastructural system".

- Regarding a system as a service-providing entity allows us to regard this system as a *black box*.

- If some part of a system domain is viewed as an information system by some viewer, then this information system *may* be regarded as a black-box which interacts with its environment.

- The activity and resource modeling as discussed in the previous chapters basically took a *white box* approach to a system. We were looking inside the system to see what activities take place, who the actors are that execute the activities, etc.

- When regarding a system as a black box, we obtain *service information hiding*, in other words, we do not see the way a service delivery is actually implemented.

- Service information hiding as defined by A.P. Barros:

  A technique (for business modeling) should allow the formulation of service requests to be independent of their actual processing.

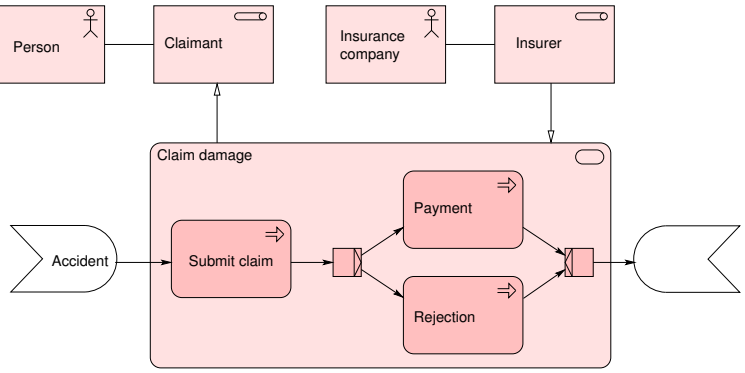
  In other words, we should be able to talk about service independent of the (business) processes that implement them
- Why information hiding?
  - One: description of a service
  - Many: ways to implement/deliver
  - Useful for:
    * Software
    * Organisations
    * ... any system that is purposely designed

9.2 Modeling services

- A service delivery can be regarded as moving through a number of states and transitions.

  We do not want to "see" the details of the transition, but do want to see the overall states of the service delivery as they are experienced by the service consumer.

  We will focus on the interaction between the consumer and offerer of the service.
- A service can be regarded as a collaborative activity between the provider of the service (e.g. an organisation), the consumer of the service, and potentially some additional parties.
- As such, a service can be modeled as any other activity, however, in this case the system providing the service is treated as a black-box.
- An example is shown below:



- Focus on added value of services
- Focus on interactions needed (between the supplying and consuming actors) in the service delivery
- Do not (yet) focus on the processes needed to deliver the actual service

9.3 Quality of service

The functional behavior of a service can be modeled by means of an activity model focusing on the interaction between the service provider and the consumer of the service. However, there is more to services than their "functional" behavior. Services may occur at different levels of *quality*. We now provide a brief discussion of the notion of quality, basing ourselves on [ISO01, ISO96c]. The essence of quality of a system is captured in the following definition:

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 [ISO01, ISO96c] 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 [ISO01, ISO96c] 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 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 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: 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.

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 work-days*. Such contracts make explicit the mutual committments/responsibilities of the actors involved in a service delivery.

9.4 Information systems as event-driven machines

- When viewing a system/organisation as a black box, its (model!) becomes a machine that "eats" input and "produces" output.

  In the case of an information system, the input/output consists of data representing information.

- Formally, we can model this as: $s : (\mathcal{TI} \rightarrowtail \mathcal{EL}) \rightarrow (\mathcal{TI} \rightarrowtail \mathcal{EL})$.

  When observing system s, a viewer has a conception of s where in the course of time "things" are fed into the system, while the system produces yet other "things". These "things" are *elements* from the viewer's conception.

- If $s(I) = O$, then system s converts input stream I into stream O.
- If $s(I)$ and $I{\downarrow}t$, then one can say that for system s an event occurs at t, being the reception of $I(t)$.
- In the case of an information system, we could have:

  ADD Person 'Erik' works for department 'IRIS'

  and

  LIST People who work for department 'IRIS'

  as inputs. The output might be:

  accept

  and

  Erik, Stijn, Theo

  respectively

Bibliography

[ISO96] ISO. *Kwaliteit van softwareprodukten*. ten Hagen & Stam, Den Haag, The Netherlands, 1996. In Dutch. ISBN 9026724306

[ISO01] *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001. `http://www.iso.org`

Part III

Model-driven System Engineering

Chapter 10

Models in System Engineering

Version: 25-05-2005

In this chapter we are concerned with the role of models in (information) system engineering. To this end, we will study the notion of system engineering more closely, as well as challenges on *information* system engineering in the digital age and consequences that follow from this for the use of models.

10.1 Systems engineering

In Chapter 1, we already defined information (system) engineering as:

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

Information system engineering – A system engineering process pertaining to the creation or change of information systems.

with as key sub-processes:

Definition process – A process aiming to identify all requirements that should be met by the system and the system description.

In literature this process may also be referred to as requirements engineering.

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.

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.

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

We claim that these four processes are key to system development in general.

When integrating these processes into a unified process, a first, and naive, way of representing the resulting overall development process, would lead to the situation as shown in Figure 10.1. In the situation depicted, it is presumed that some operational system exists, and that there is a need to change/improve/extend this system. By means of an definition process, the requirements of this desired system change can be ascertained. Using these requirements as a starting point, a new system can be designed, leading to a system design. Based on this design a new system may be constructed, which, after completion, may then be installed as part of the operational system.
.

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

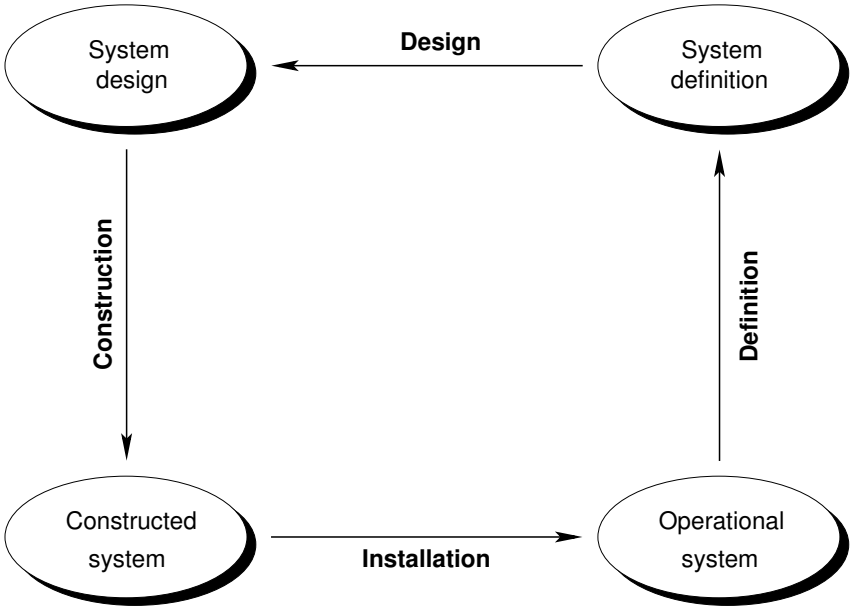


Figure 10.1: System development

- It suggests a linear flow of activities.
- It does not distinguish between system domains, conceptions of systems and system descriptions.

The arrows in Figure 10.1 should indeed not be interpreted as putting a requirement on the *start* of the processes, but rather of the *finalisation* of them. In other words, the processes may quite well run in parallel, however, the definition process should be finalised before the definition process can be finalised, etc. There are actually three major flavours of development approaches that may be used [FV99]:

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) Installation

In chapter **??**, where we will focus on a way of controlling for system engineering, we will discuss these options in more detail. In the next chapter, where we will discuss system requirements, we will also see how, in general, it is nearly impossible to first elicit all relevant requirements before starting design.

With regards to the distinction between system domain, conceptions of systems and system descriptions, a more refined view of figure 10.1 is given in figure **??**. Both the operational system (needing a change) and the constructed system (providing the actual change) are parts of the universe, i.e. that what we perceive to be 'the real world'. In other words, both the constructed system and the operational system from figure 10.1 actually refer to system domains in the universe. The system requirements, as well as the system design, are really system *descriptions*.

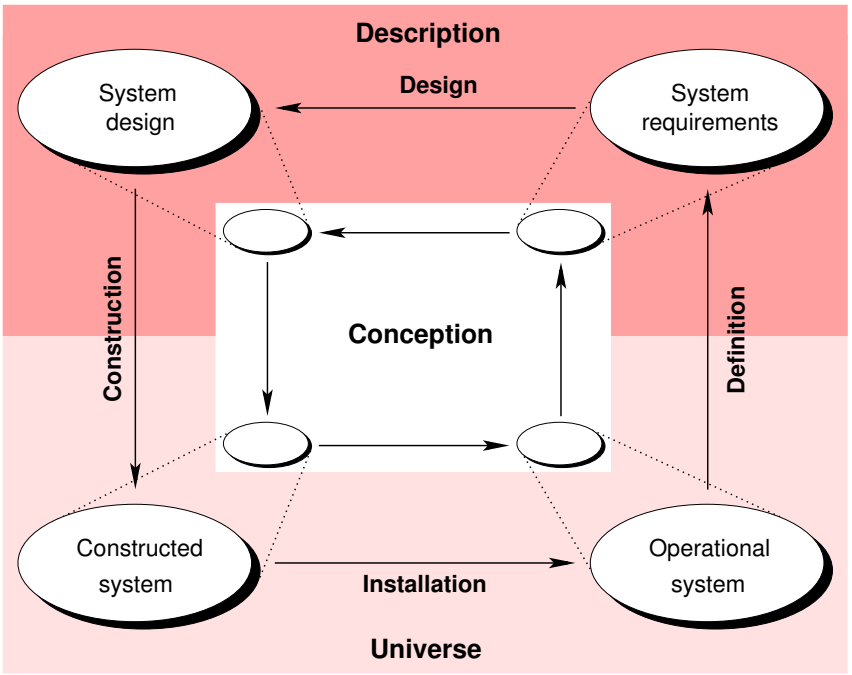


Figure 10.2: System engineering – refined view

Note: Requirements are models in the minds of viewers. They also have their descriptions.

10.2 System engineering community

In system engineering, models are primarily used as a means of communication between the actors involved in the engineering process. Given our focus on communication, it is important to identify the actors that can play a role in the communication that takes place during the system development process. 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.

(We will clarify below why we regard documents as being part of the community).

The *actors* in a system development community will (typically as a consequence of their *stakes*) have some specific interests with regards to the system being developed. This interest implies a sub-interest with regards to (the contents of) the the system descriptions that are communicated within the community. This interest, in line with [IEE00], is referred to as the *concern* of stakeholders:

Concern – An interest of a stakeholder, resulting from the stakeholder goals, and the role played by some system. 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.

Actors in a system development community who play an instrumental role will typically not have a concern with regards to the system being developed; they are neutral with regards to stakes and concerns. Some example of concerns are:

- The current situation with regard to the computerized support of a business process.
- The requirements of a specific stakeholder with regard to the desired situation.
- The improvements, which a new system may bring, to a pre-existing situation in relation to the costs of acquiring the system.
- The potential impact of a new system on the activities of a prospective user.
- The potential impact of a new system on the work of the system administrators that are to maintain the new system.

10.2.1 Stakeholders and their concerns

During the development of a system, different people may have different interests with regards to the system. Collectively, we will refer to these people as stakeholders:

Stakeholder – A party (a system viewer) with a specific interest 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, ...

The interests of a stakeholder with respect to some system to be developed, originate from some deeper motivation: 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 system engineering costs, while the goal of users of the system may be to get their job done more efficiently.

The specific interests of a stakeholder are, in line with [IEE00], referred to as stakeholder concern:

Stakeholder concern – An interest of a stakeholder, resulting from the stakeholder's goal, and the role played by some system.

This usually pertains to the system's engineering, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

Usually we will abbreviate “stakeholder concern” to “concerns”. Each of these stakeholders are obviously system viewers, viewing the system with specific interests; their concerns.

A system (to be developed) is aimed to be beneficial for some group of stakeholders. This benefit is referred to as the system mission:

System mission – A role, to the benefit of stakeholder goals, for which the system is intended.

This leads to the situation as depicted in figure 10.3. A stakeholder will typically have some operational goal in which the system will/should/may play a role. Due to this potential role, the stakeholder is bound to have some concerns with regards to a pre-existing system or a system to be developed. Facing these concerns we find the system mission, in other words, that what the system aims to be for its stakeholders.

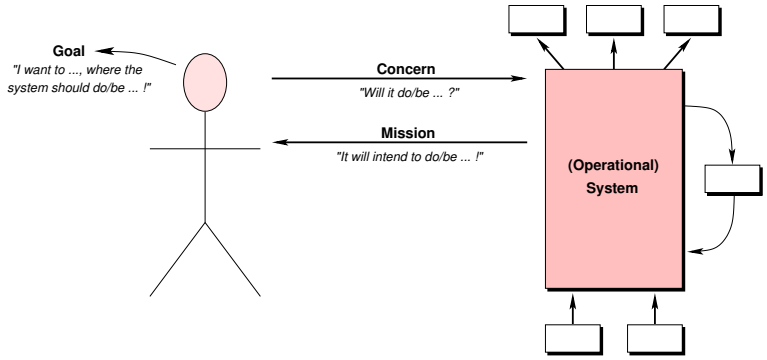


Figure 10.3: Meeting stakeholder concerns

10.3 Information system engineering as a wicked problem

Traditioneel wordt de ontwikkeling van informatiesystemen uitgevoerd middels projecten. Het ontwikkelen van informatiesystemen in het digitale tijdperk brengt, zoals ik al eerder heb beargumenteerd, diverse nieuwe uitdagingen met zich mee. We kunnen ons hierbij afvragen in hoeverre een traditionele manier van denken in termen van projecten hierbij nog van toepassing is.

Elke projectleider zal beweren dat een project goede uitgangspunten en einddoelen dient te hebben die helder afgebakend en stabiel zijn. Echter, als direct voortvloeisel van de geponeerde uitdagingen verwevenheid, pluriformiteit, ongrijpbaarheid en veranderlijkheid, heeft de ontwikkeling van informatiesystemen in het digitale tijdperk te maken met een grote mate van vaagheid, onstabiliteit en onzekerheid ten aanzien van juist deze uitgangspunten en einddoelen. Dit kan nogal wat gevolgen hebben voor ontwikkelprojecten. Laten we dit daarom eens nader analyseren.

Informatiesysteemontwikkeling kan gezien worden als het oplossen van een “probleem”. Als een bestaande situatie niet voldoet aan de systeemeisen, dan is het probleem dat opgelost moet worden: “Zorg dat er een systeem komt dat wél aan de eisen voldoet”.

10.3.1 Wicked problems

In het algemeen bestaan er problemen in soorten en maten. Een relevant onderscheid dat hierbij gemaakt kan worden is het onderscheid tussen zogenaamde “gemene” en “tamme” problemen. In het Engels worden deze typen van problemen respectievelijk “wicked problems” en “tame problems” genoemd. Een gemeen probleem laat zich typisch kenmerken door eigenschappen zoals de volgende:

- Je begrijpt een gemeen probleem pas goed als je er een oplossing voor hebt bedacht. Elke mogelijke oplossing brengt nieuwe aspecten van het probleem aan het licht, aspecten die verdere aanpassing van de oplossing vereisen.
- Gemene problemen hebben geen stopcriterium. Er is geen eenduidige en stabiele probleemdefinitie te geven. Als gevolg hiervan is het is niet duidelijk wanneer het probleem echt is opgelost.
- Oplossingen voor gemene problemen zijn niet simpelweg goed of fout. In plaats daarvan zijn ze "beter", "slechter" of "goed genoeg". Voor gemene problemen is het moeilijk om op een objectieve wijze de kwaliteit van een oplossing te beoordelen.
- Elke oplossing van een gemeen probleem krijgt slechts één kans. Elke realistische poging heeft direct consequenties. Je kunt niet eerst een realistisch prototype van de Betuwelijn bouwen om te zien of het allemaal wel zal werken in de praktijk. Hierbij komt meteen de in gemene problemen ingebouwde patstelling naar voren. Je kunt het probleem niet echt verkennen zonder oplossingen te proberen, maar elke oplossing die je probeert heeft onvoorziene bijeffecten die het probleem direct of indirect verder beïnvloeden.

Deze vier criteria zijn niet alle criteria voor gemene problemen, maar het zijn naar mijn mening wel de meest onderscheidende. Een tam probleem laat zich typisch kenmerken door eigenschappen zoals:

- Het probleem heeft een vooraf goed gedefinieerde en stabiele probleemdefinitie.
- Het heeft een duidelijk stopcriterium om te bepalen of een goede oplossing is gevonden.
- Er is een oplossing die op objectieve wijze beoordeeld kan worden op zijn correctheid.
- Het probleem heeft oplossingen die makkelijk, zonder ongewenste consequenties, uitgeprobeerd kunnen worden.

Deze typen van problemen zijn bedoeld als extremen. Een probleem kan best een "beetje gemeen" of "enigszins tam" zijn. Overigens zegt het gemeen of tam zijn van een probleem niets over de moeilijkheidsgraad van het probleem zelf. Het bewijzen van Fermat's laatste stelling was volgens deze definitie weliswaar een tam probleem, maar men heeft er jaren over gedaan om een oplossing te vinden.

Hoe zit het nu met het ontwikkelen van informatiesystemen? Komen we daar tamme of gemene problemen tegen? Mijn stelling is dat het ontwikkelen van informatiesystemen in het digitale tijdperk tendeert naar een gemeen probleem. Uitgaande van de uitdagingen voor informatiesysteemontwikkeling in het digitale tijdperk zullen veel ontwikkelprojecten al snel voldoen aan de criteria voor gemene problemen. Laat we de belangrijkste criteria wat nader bekijken.

Het eerste criterium betrof het feit dat een probleem pas echt begrepen kan worden als er een eerste oplossing voor is bedacht. Dit criterium sluit direct aan bij de uitdagingen verwevenheid en ongrijpbaarheid. Terwijl men een informatiesysteem ontwikkelt, zal men nieuwe belanghebbenden kunnen tegenkomen, en zullen belanghebbenden door het concreter worden van het nieuwe systeem steeds beter gaan inzien wat hun feitelijke belangen zijn.

Het tweede criterium betrof het ontbreken van een goed stopcriterium. Bij het ontwikkelen van informatiesystemen ontstaat er een soort natuurlijke drang om te gaan schuiven met de eisen ten aanzien van het nieuwe informatiesysteem. Ten dele zal dit het gevolg zijn van veranderingen die plaatsvinden in de socio-economische context gedurende de ontwikkeling van het systeem. De wereld staat niet stil. Er zijn echter ook redenen dichter bij huis te vinden. Het ontwikkelen van een informatiesysteem zal vrijwel altijd de directe context van het systeem beïnvloeden. Immers, om een ontwikkelproject überhaupt op te starten moet er in eerste instantie een concrete behoefte zijn om iets te veranderen aan een reeds bestaande situatie. Er moet een "knelpunt" zijn dat als ernstig genoeg wordt ervaren, om iemand er toe te brengen de moeite te nemen een ontwikkelproject op te zetten en te betalen. Het wegnemen van dat "knelpunt" zal ongetwijfeld het gedrag van de context beïnvloeden. Immers, de context van het systeem heeft moeten leren

leven met beperkingen. Zodra die beperkingen zijn weggenomen, zal dat ongetwijfeld leiden tot ander gedrag van de context. Een verandering die vrijwel zeker nieuwe veranderingswensen met zich meebrengt, omdat er ergens anders een nieuw knelpunt zal opduiken. Nu kunnen we vervolgens hard roepen "ja maar dat nieuwe knelpunt is niet het originele probleem", maar dat verandert niets aan het feit dat de definitie van het probleem is veranderd, en dat het dus maar de vraag is of we uiteindelijk het goede probleem hebben opgelost. Wat is het goede stopcriterium?

Een concreet voorbeeld van dit fenomeen is het oplossen van fileproblemen. Wanneer we bij een systeem van snelwegen op een bepaalde plek de wegen verbreden om op die plek de files op te lossen, dan zullen er vrijwel zeker op andere plekken nieuwe files ontstaan. Aan de ene kant zullen auto's op voormalige knelpunten sneller kunnen doorstromen, en daardoor ergens anders nieuwe files veroorzaken. Aan de andere kant, en dat is het gemene, zal de vermindering van de files waarschijnlijk veroorzaken dat ineens meer mensen tijdens de spitsuren van hun auto gebruik wensen te maken. Vooraf is dit gedrag moeilijk voorspelbaar. Een directe informatietechnologische tegenhanger van de filesituatie is het automatiseren van productieprocessen. Wanneer we knelpunten in productieprocessen oplossen door deze te automatiseren, kunnen er ergens anders weer nieuwe knelpunten ontstaan. Soms zijn deze knelpunten voorspelbaar, maar vaak ook niet.

Het derde criterium voor gemene problemen had betrekking op het feit dat oplossingen niet simpelweg goed of fout zijn. Wanneer is een informatiesysteem goed? Als het conform de specificaties werkt? Maar wat als het systeem, om dat niveau van correctheid te bereiken, te laat wordt opgeleverd? Is een op tijd opgeleverd systeem met een aantal fouten in niet al te cruciale delen van het systeem, niet "beter" dan een systeem dat een maand later wordt opgeleverd zonder fouten? Stel dat het gaat om een informatiesysteem dat noodzakelijk is voor de introductie van een nieuw product op de markt. Een maand eerder met het nieuwe product op de markt komen dan de concurrentie kan wel eens van doorslaggevend belang zijn. Hoe erg zijn die overgebleven fouten dan nog?

Daarnaast blijft het, zie ook de vorige twee criteria, moeilijk om te komen tot een kwantitatieve specificatie van wat eigenlijk het goede informatiesysteem is. Als mens zijn we uitermate adaptief ingesteld. Zelfs als een systeem niet precies doet wat we er van verwachten, kunnen we er nog steeds erg nuttig gebruik van maken. Is dat systeem dan "slecht"? Het kan beter. Maar dat kan het bijna altijd. Wanneer is goed goed genoeg?

Volgens het vierde criterium zou een oplossing van een gemeen probleem slechts één kans hebben om zich te bewijzen. Dit criterium zal niet altijd van toepassing zijn bij het ontwikkelen van informatiesystemen. Als het te ontwikkelen informatiesysteem klein genoeg van omvang is, zou men zich kunnen veroorloven om te experimenteren met alternatieve oplossingen. Echter, bij grootschalige informatiesystemen, zoals de OV chipkaart, elektronische belastingaangifte, het betalingsverkeer, etcetera, is er slechts beperkt ruimte om op realistische schaal te experimenteren. Na bijvoorbeeld een mislukte poging tot het introduceren van een OV chipkaart, moet er voor een volgende poging rekening worden gehouden met eventuele "overblijfselen" van deze eerdere poging. Het oorspronkelijke probleem bestaat daarom ook niet meer als zodanig. Je hebt maar één kans gehad om het originele probleem op te lossen.

Tenslotte moet nog opgemerkt worden dat gemeenheid een eigenschap van het probleem zelf is. Maar dat is nog niet het hele verhaal. Naast het probleem zelf, hebben we namelijk ook nog te maken met een context waarin het probleem opgelost moet worden. Twee van de genoemde uitdagingen voor de Informatiekunde hebben meer betrekking op de context van het probleem dan op het probleem zelf. Het gaat hierbij om de pluriformiteit van de belanghebbenden en de veranderlijkheid van de socio-economische en technologische context. Die twee factoren dragen in veel gevallen nog eens extra bij aan de gemeenheid. We mogen concluderen dat informatiesysteemontwikkeling in het digitale tijdperk tendeert een gemeen probleem te zijn, waarbij de context van het probleem de gemeenheid eerder zal verergeren dan verminderen.

10.3.2 Traditionele informatiesysteemontwikkeling

Hoe wordt bij het ontwikkelen van informatiesystemen traditioneel omgegaan met gemeenheid? Informatiesysteemontwikkeling kan opgedeeld worden in een aantal fasen. Nu zijn er verschillende manieren om deze indeling te maken, maar grofweg kunnen we stellen dat er vier belangrijke fasen zijn te onderkennen: definiëren, ontwerpen, construeren en implementeren. Bij het definiëren staan drie vragen centraal: wat moet het systeem doen, waarom moet het dit doen, en hoe goed moet het dit doen. Deze fase leidt tot de probleemdefinitie. Tijdens de ontwerpfase zijn de centrale vragen: hoe moet het systeem er uit zien om te voldoen aan de eisen zoals die opgesteld zijn in de definitiefase, en waarmee kan dit gerealiseerd worden. Na de ontwerpfase volgt de constructiefase. In deze fase worden alle benodigde onderdelen van het informatiesysteem bijeengebracht, en wordt het eindresultaat samengesteld. Het kan hierbij gaan om hardware, software, handleidingen, etcetera. Wat volgt, in de implementatiefase, is de daadwerkelijke in gebruik name van het eindresultaat door de opdrachtgevers. Na de implementatie volgt doorgaans het beheren van het systeem. Zoals ik echter eerder heb aangegeven, richt ik me in mijn onderzoek specifiek op systeemontwikkeling.

Er zijn verschillende strategieën om de vier genoemde fasen te doorlopen. Een lineaire strategie loopt, zoals de naam al suggereert, voor het gehele systeem stap voor stap alle fasen door. Dit wordt ook wel eens een waterval aanpak genoemd, omdat we als het ware van de ene fase naar de andere fase stromen. Een eerste variatie hierop is de incrementele strategie, waarbij de verschillende fasen per deelsysteem doorlopen worden. Deze strategie gaat er wel van uit dat we een zinnige opdeling van het systeem hebben kunnen maken in deelsystemen. Een derde variant is de iteratieve strategie. Hierbij wordt door veelvuldig op en neer springen tussen de verschillende fases een voldoende goed eindresultaat toegewerkt.

Wat al deze aanpakken gemeen hebben is dat ze uitgaan van wat ik zou willen noemen het "projectdenken". Men neemt als vertrekpunt de aanname dat men, idealiter, in staat is om heldere uitgangspunten en einddoelen te formuleren. Startende vanuit het projectdenken zou je het liefst willen zien dat je te maken hebt met situaties waarin je een nette, goed gestructureerde, lineaire aanpak kunt gebruiken. Omdat de praktijk weerbarstiger is dan dat, moest het projectdenken opgerekt worden. Dit heeft geleid tot de meer iteratieve strategieën. We kunnen echter stellen dat zelfs wanneer een iteratieve strategie gebruikt wordt, dit toch een ontkenning blijft van het gemene karakter van de problemen die men oplost. Men blijft immers het projectdenken, dat vraagt om heldere uitgangspunten en einddoelen, als kader gebruiken.

10.3.3 Evenwichtsdenken

We kunnen ons afvragen of we er niet veel beter aan doen om in plaats van het projectdenken een ander startpunt te kiezen, waarbij ik overigens niet wil zeggen dat projecten afgeschaft moeten worden. Projecten zijn zeker nodig, ze bieden immers een goede manier om stabiele en behapbare brokken werk te verzetten. Ze zouden echter ingebed moeten worden in een groter geheel dat niet persé uitgaat van stabiele uitgangspunten en einddoelen. Het is overigens interessant om te zien hoe grootschalige ontwikkelprojecten nogal eens de neiging hebben om instituten te worden. Wat begon als project wordt op een zeker moment bijna een staande organisatie op zichzelf. Het feitelijke werk hierbinnen wordt weer uitgevoerd middels kleinere projecten. Als tegenbeweging wordt ook wel gesproken over microprojecten, waarbij het gaat om het initiëren van vele kleine projecten, in plaats van één onbestuurbare kolos .

We kunnen informatiesysteemontwikkeling wellicht het beste zien als een continu evolutionair proces, met als doel het bewaren van evenwicht tussen drie polen: definitie, ontwerp en de operationele situatie. De definitiepool richt zich op het wat, waarom en hoe goed, terwijl de ontwerppool zich richt op het hoe en waarmee. De operationele pool betreft de actuele informatievoorziening, waarbij het overigens best mogelijk is dat er in een actuele situatie nog geen gebruik gemaakt wordt van een (gecomputeriseerd) informatiesysteem.

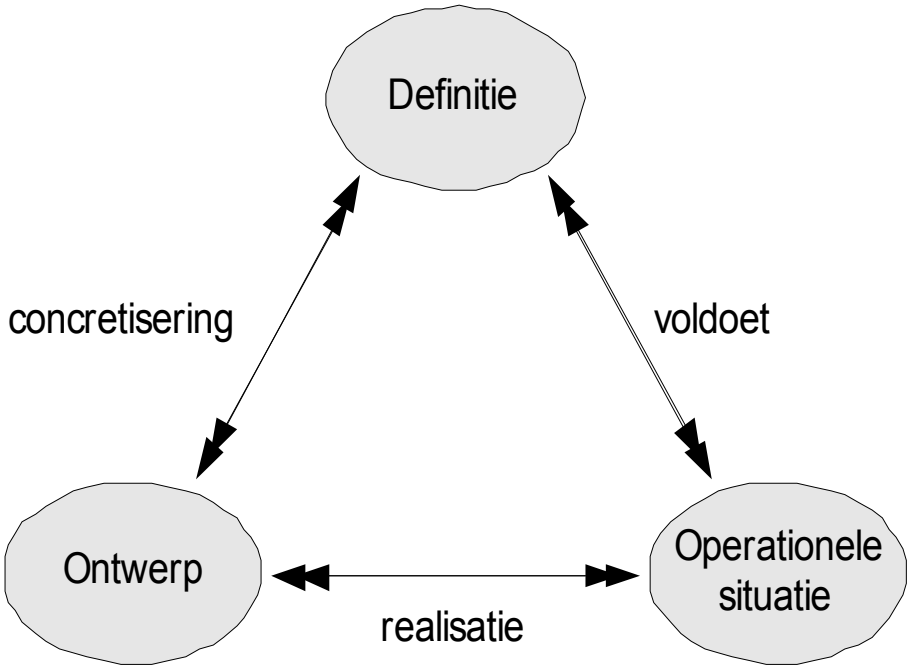


Figure 10.4: Evenwicht in informatiesysteemontwikkeling

Zoals in Figure 10.4 is geïllustreerd spannen de drie polen (metaforisch gezien) samen een spanningsveld op. De drie polen zijn onderling verbonden middels een drietal krachten; drie elastiekjes. Het elastiekje tussen de definitiepool en de operationele pool heeft betrekking op de vraag in hoeverre de operationele situatie voldoet aan de gestelde de definitie. Het elastiekje tussen de definitiepool en de ontwerppool richt zich op de vraag of het ontwerp een concretisering is van de definitie; een goede vertaling van het wat naar het hoe. Tenslotte richt het elastiekje tussen de ontwerppool en de operationele pool zich op de vraag of de operationele situatie een goede realisatie is van het ontwerp. Het evenwichtsmodel in Figure 10.4 is van toepassing op alle soorten informatiesysteemontwikkeling, dus zowel bij nieuwbouw, correctie, uitbouw als renovatie.

Het evenwicht bewaren

Het evenwichtsmodel zegt nog niets over de eventuele processen die het krachtenveld kunnen beïnvloeden. Idealiter is er een volledige balans tussen de drie polen. Bij een onbalans ontstaat er een behoefte aan verandering. Mits we een dergelijke verandering scherp genoeg kunnen formuleren in termen van uitgangspunten en einddoelen, en de verandering klein genoeg is qua omvang en te verwachten doorlooptijd, kan er een project opgestart worden om deze verandering daadwerkelijk te verwezenlijken.

In elk van de drie verbindingen kan er sprake zijn van een onbalans. Zo kan het zijn dat de behoeften van de belanghebbenden niet voldoende afgedekt worden door de operationele situatie. In zo'n geval kan de balans hersteld worden door de operationele situatie aan te passen, en/of door de behoeften bij te stellen. Dat laatste klinkt misschien raar, maar u zou zich een situatie kunnen voorstellen waarin de diverse belanghebbenden er onrealistische wensen op nahouden, die bijvoorbeeld met de huidige stand der technologie niet praktisch realiseerbaar zijn, terwijl de verschillende belanghebbenden elkaar misschien ook nog eens tegenspreken. Het kan in een dergelijke situatie heel effectief zijn om een project op te starten dat er voor moet zorgen dat de belanghebbenden in gaan zien wat realistische en haalbare wensen zijn, gelet op de stand der

technologie, de eventuele kosten van verbeteringen en de belangen van andere belanghebbenden.

Wanneer de definitie en het ontwerp met elkaar in onbalans zijn, moet óf het ontwerp geactualiseerd worden, óf moeten wederom de behoeften van de belanghebbenden bijgesteld worden. Een onbalans tussen realisatie en ontwerp kan er op duiden dat de operationele situatie nog verder geactualiseerd moet worden. Het kan er echter ook op duiden dat het ontwerp achterloopt op de operationele situatie. In de praktijk komt men dit laatste nogal eens tegen als er veranderingen aan een systeem zijn aangebracht zonder dat het ontwerp hierop is aangepast.

Elke actie die ondernomen moet worden om de balans weer te herstellen kan in potentie uitgevoerd worden middels een project. De uitgangspunten en einddoelen van een dergelijk project kunnen uitgedrukt worden in termen van de mate waarin de balans verbeterd moet worden. Dit klinkt nog vaag. Het zal in de toekomst exacter gemaakt moeten worden: exacte vaagheid! De uitdaging is om zo exact mogelijk uit te kunnen uitdrukken wat het is om in balans te zijn. Met andere woorden, we moeten er voor zorgen dat er meetinstrumenten komen om, voor elk van de elastiekjes uit het evenwichtsmodel, precies vast te stellen wat de mate van balans in een bepaalde situatie is.

Het evenwicht toetsen

Om in staat te zijn vast te stellen in welke mate een operationele situatie daadwerkelijk een realisatie is van het ontwerp, zijn instrumenten en methoden nodig om te testen of een operationele situatie zich gedraagt conform het ontwerp. De Informatica beschikt hier reeds over een uitgebreid instrumentarium voor het testen van software- en hardwareonderdelen van een informatiesysteem . Daarnaast bieden methoden zoals Total Quality Management aanknopingspunten om te toetsen in hoeverre het menselijke deel van een informatiesysteem zich conform het ontwerp gedraagt. Een belangrijke voorwaarde is wel dat de talen die gebruikt worden om het informatiesysteem te ontwerpen een goed gedefinieerde semantiek hebben; ze dienen een exacte basis te hebben. We moeten immers wel precies weten waar we op willen testen/toetsen.

Bepalen in welke mate een ontwerp een concretisering is van de definitie, en of een operationele situatie voldoet aan de gestelde definitie, is uitdagender dan het op het eerste gezicht lijkt. Allereerst zijn er specificatietalen nodig om de definitie eenduidig te kunnen vastleggen. Idealiter kunnen we alle soorten kwaliteiten, dus zowel de functionele als niet-functionele, waaraan het systeem zou moeten voldoen in deze talen uitdrukken. Vervolgens kunnen we dan instrumenten en methoden ontwikkelen om te controleren of de operationele situatie, of het ontwerp hiervan, voldoet aan de specificatie. Hier zal overigens niet altijd een simpel ja/nee antwoord mogelijk zijn.

De benodigde specificatietalen kunnen we zeker ontwikkelen. Echter, de aanname dat we de definitie eenduidig kunnen vastleggen is wat naef. Waar moet die eenduidige definitie vandaan komen? Er zijn twee factoren die het verkrijgen hiervan nogal bemoeilijken. Ten eerste zit de definitie van het systeem in eerste instantie in de hoofden van de belanghebbenden verstopt. En om de zaak nog erger te maken, zullen die belanghebbenden zich vaak niet eens precies bewust zijn van hun specifieke wensen en eisen. De ongrijpbaarheid van informatiesystemen speelt ons hierbij ook nog eens flink parten. We worden gedwongen om een onderscheid te maken tussen een deel van de definitie dat reeds expliciet is gemaakt, dus op papier staat, en een deel dat impliciet is gebleven. Zodra een belanghebbende zich bewust wordt van een aspect van zijn of haar belangen, en dit kan verwoorden, kunnen we het toevoegen aan het expliciete deel van de definitie. Pas dan zijn we in staat om middels ons instrumentarium te toetsen of de bestaande situatie, of een ontwerp daarvan, hieraan voldoet. Toetsen of een ontwerp of een bestaande situatie voldoet aan het impliciet gebleven deel van de definitie is uiteraard niet zomaar mogelijk. Wel kunnen methoden worden ontwikkeld om, op basis van een ontwerp of een operationele

situatie, belanghebbenden te helpen in het expliciet maken van hun behoeften. De tweede bemoeilijkende factor bij het vaststellen van een eenduidige definitie vloeit voort uit de pluriformiteit van de belanghebbenden en hun belangen. Als gevolg van deze pluriformiteit is de kans groot dat we te maken hebben met tegenstrijdige wensen ten aanzien van het informatiesysteem. Het verkrijgen van een eenduidige definitie vereist dus ook onderhandelingen tussen de verschillende belanghebbenden en hun belangen.

Het evenwicht sturen

Een voor de hand liggende vervolgvraag is de vraag hoe het evenwicht gestuurd kan worden, zowel wat betreft de inhoudelijke als de procesmatige sturing. Bij inhoudelijke sturing gaat het om de uiteindelijke inrichting van het ontwikkelde informatiesysteem, terwijl het bij procesmatige sturing gaat om de inrichting van de ontwikkelprocessen. In ons onderzoek gaan we er van uit dat het belangrijk is dat beide vormen van sturen zo bewust mogelijk dienen te gebeuren. Met bewust doelen we hier op het feit dat beslissingen ten aanzien van de inrichting van het informatiesysteem en het ontwikkelproces zoveel mogelijk op rationele grondslagen genomen dienen te worden. Een beslissing om op een bepaalde manier het informatiesysteem in te richten heeft invloed op de kwaliteitseigenschappen van het resulterende systeem. Gelijkelijk zal een beslissing om het ontwikkelproces op een bepaalde manier in te richten van invloed zijn op de kwaliteit van dat proces. In ons onderzoek beschouwen we "bewuste sturing" als een combinatie tussen de drie volgende ingrediënten:

- De semantiek van beslissingen dient gezien te worden als de impact die ze hebben op de kwaliteitseigenschappen van het resultaat. Op basis hiervan kan een goede kosten-baten analyse gemaakt worden ten behoeve van de besluitvorming.
- Inrichtingsbeslissingen dienen zoveel mogelijk expliciet genomen te worden. Bij informatiesysteemontwikkelprojecten worden nogal eens verborgen beslissingen genomen. Onbewust wordt er gekozen voor een bepaalde technologie, methode of techniek, of wordt er een motivatie van een project als gegeven beschouwd zonder nadere toetsing.
- Communiceren en onderhandelen tussen de verschillende belanghebbenden over inrichtingsbeslissingen is essentieel om tot een goede besluitvorming en concretisering van de gekozen oplossingsrichtingen te komen.

10.4 Viewpoints for system description

In this section we discuss the notion of *viewpoints* as basic building blocks in the communication in a system engineering community. In this context, 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. 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*.

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-designs. Similarly, sub-designs in a system design can be identified. By taking, a certain view on the design a sub-design follows.

10.4.1 Origin of viewpoints

The concept of viewpoint is not new. For example, in the mid eighties, Multiview [WAA85] already introduced the notion of views. In fact, Multiview already identified five viewpoints for the development of (computerised) information systems: Human Activity System, Information Modelling, 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"). As a precursor to the notion of *concern*, the CRIS Task Group identified several *human roles* 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*" [OHM+88] and in the proceedings of the CRIS conferences from 1982–1991 [OSV82, OST83, OSV86, OSV88, VO91].

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 phrased as "the multiple perspectives problem" [FKN+92, KS92, RMB95]. By this term, the authors referred to the problem of how to organise 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 [FKN+92, KS92]. In this framework, a viewpoint combines the notion of *actor*, *role*, or *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 [IEE00] on which we have based our definitions below.

10.4.2 Viewpoints on systems

In the context of system engineering, viewpoints provide a means to focus on particular aspects of an 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 uni-directional, informtive conversations, they can in general also be used in bi-directional classes of conversations (e.g. immediate respons situations; cooperative modeling). 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 computerised 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?

In line with IEEE 1471, and based on the detailed definition given in [Pro04], we define a viewpoint to be:

Viewpoint – a specification of the conventions for constructing and using views.

This involves:

Way of thinking – articulates the assumptions about the kinds of problem domains, solutions, and modellers involved. This notion is also referred to as the *Weltanschauung* [**?**, **?**], *philosophy* [**?**].

Way of modeling – identifies the core *meta-concepts* of the language that may be used to denote, analyse, visualise and/or animate system descriptions.

This notion is also compatible with the approach on viewpoints as it is taken in the Reference Model of Open Distributed Processing [ISO98b]:

> *"In order to represent an ODP system from a particular viewpoint it is necessary to define a structured set of concepts [the meta-concepts] in terms of which that representation (or specification) can be expressed. This set of concepts provides a language for writing specifications of systems from that viewpoint, and such a specification constitutes a model of a system in terms of the concepts."*

Way of communication – describes how the abstract concepts from the *way of modeling* are communicated to human beings, for example in terms of a textual or a graphical notation (syntax, style, medium).

Way of working – structures (parts of) the way in which a system is developed. 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 supporting – the support that is offered by (possibly automated) tools for the handling (creating, presenting, modifying, etc.) of views.

In general, a way of supporting is supplied in the form of some computerised tool (see for instance [**?**]).

Way of using – identifies heuristics that:

- define situations, classes of stakeholders, and concerns for which the viewpoint is most suitable,
- provide guidance in tuning the viewpoint to specific situations, classes of stakeholders, and their concerns.

10.4.3 Viewpoint frameworks

In the context of information system engineering, a score of viewpoint frameworks exists, leaving designers and systems with the burden of selecting the viewpoints to be used in a specific situation. Some of these frameworks of viewpoints are: The Zachman framework [Zac87], Kruchten's 4+1 framework [Kru95], RM-ODP [ISO98b], ArchiMate [JVB+03] and TOGAF [TOG04]. 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. We shall discuss two of these framework in more detail below.

The '4+1' view model

In [Kru95], Kruchten introduces a framework of viewpoints (a view model) comprising five viewpoints. 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 [Kru00, BRJ99].

The goals, stakeholders, concerns, and meta-model of the 4+1 framework can be presented, in brief, as follows:

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 ...

Note: in [Kru00, BRJ99], the viewpoints have been re-named; physical viewpoint → deployment viewpoint, development viewpoint → implementation viewpoint and scenario viewpoint → use-case viewpoint, to better match the terminology of the UML.

The framework proposes modeling concepts (the meta-model) for each of the specific viewpoints. It does so, however, without explicitly discussing how these modeling 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 modeling concepts. In [Kru00, BRJ99] this is also not documented. The viewpoints are merely presented 'as is'.

RM-ODP

The Reference Model of Open Distributed Processing (RM-ODP) [ISO98b, ISO96b, ISO96a, ISO98a] 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 modeling 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 follows:

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

The RM-ODP provides a modeling 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 modeling concepts selected in each of the views.

Questions

1. Why is it important to realise that system development is not necessarily a linear process?
2. Name three different axes along which a high-level design may be 'refined' to a more detailed design.
3. Describe, in your own words, the relationships between the concepts of: domain, viewer, conception, perception and architecture.
4. What is the difference between: an architecture, architectural description, architectural view, and architectural viewpoint?
5. Why is it important to carefully select relevant system viewpoints when developing systems?
6. Make a meta-model of the concepts introduced by the IEEE recommended practice for architecture [IEE00]. Make sure to include as many constraints as you can deduce from the text.

7. Why is it important to acknowledge the fact that different stakeholders' will have different views on a pre-existing or a future system?
8. Make a meta-model of the concepts introduced with respect to systems, domains, viewers, etc.
9. Identify, for the system description language you have see so-far as part of your studies:
   (a) The main concepts of these languages.
   (b) Typical interests and viewers for which these languages may be useful.
10. Why should the set of viewpoints that will be used to describe different aspects of a system that is being developed, be selected carefully.
11. What makes us consider a system description to be an architecture description?
12. What are possible dimensions for refinements of system descriptions?
13. Why is it important to consider responsibilities of system entities and collaborations among them?
14. Consider a candy vending machine and people purchasing candy from the machine. What would, in such a domain, be the relevant system entities? What would be there responsibilities and collaborations?
15. Suppose you would design a pocket calculator. What would be the essential system elements (at a functional level)? Maybe you could do a role-playing game with a group
16. What's the difference between an information system and a medium system?
17. Consider a bank. Not the one you sit on, but the one you entrust with your money. Provide a brief discussion of a possible business, work and information view (according to the Tapscott viewpoints) for such a domain.

Recommended reading

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000. ISBN 0738125180
`http://www.ieee.org`

[Zac87] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

Optional reading

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

Bibliography

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999. ISBN 0201571684

[FKN+92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system de-velopment. *International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments*, 2(1):31–58, 1992.

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000. ISBN 0738125180
`http://www.ieee.org`

[ISO96a] *Information technology – Open Distributed Processing – Reference model: Architecture*, 1996. ISO/IEC 10746-3:1996(E).
`http://www.iso.org`

[ISO96b] *Information technology – Open Distributed Processing – Reference model: Foundations*, 1996. ISO/IEC 10746-2:1996(E).
`http://www.iso.org`

[ISO98a] *Information technology – Open Distributed Processing – Reference model: Architectural semantics*, 1998. ISO/IEC 10746-4:1998(E).
`http://www.iso.org`

[ISO98b] *Information technology – Open Distributed Processing – Reference model: Overview*, 1998. ISO/IEC 10746-1:1998(E).
`http://www.iso.org`

[JVB+03] H. Jonkers, G.E. Veldhuijzen van Zanten, R. van Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Guillen Scholten, S.J.B.A. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M.M. Lankhorst, D. van Leeuwen, H.A. (Erik) Proper, A. Stam, and L. van der Torre. Towards a Language for Coherent Enterprise Architecture Descriptions. In M. Steen and B.R. Bryant, editors, *7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 28–39, Brisbane, Australia, September 2003. IEEE Computer Society Press, Los Alamitos, California, USA. ISBN 0769519946

[Kru95] P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, November 1995.

[Kru00] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 2000. ISBN 0201707101

[KS92] G. Kotonya and I. Sommerville. Viewpoints for requirements definition. *IEE/BCS Software Engineering Journal*, 7(6):375–387, 1992.

[OHM+88] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verrijn-Stuart. *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, Reading, Massachusetts, USA, 1988. ISBN 0201544431

[OST83] T.W. Olle, H.G. Sol, and C.J. Tully, editors. *Information Systems Design Methodologies: A feature analysis*, York, England, EU, 1983. North Holland/IFIP WG8.1. ISBN 0444867058

[OSV82] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: A Comparative Review*. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1982. ISBN 0444864075

[OSV86] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: Improving the practice*, Noordwijkerhout, Netherlands, EU, 1986. North Holland/IFIP WG8.1.

[OSV88] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: Computerized assistance during the information systems life cycle*. North Holland/IFIP WG8.1, Malham, England, EU, 1988. ISBN 0444705120

[Pro04] H.A. (Erik) Proper. *Da Vinci – Architecture-driven Information Systems Engineering*. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU, 2004.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[RMB95] J. Reeves, M. Marashi, and D. Budgen. A software design framework or how to support real designers. *IEE/BCS Software Engineering Journal*, 10(4):141–155, 1995.

[TOG04] The Open Group. *TOGAF – The Open Group Architectural Framework*, 2004. `http://www.togaf.org`

[VO91] A.A. Verrijn-Stuart and T.W. Olle, editors. *Methods and Associated Tools for the Information Systems Life Cycle*, Maastricht, Netherlands, EU, 1991. North Holland/IFIP WG8.1. ISBN 0444820744

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0632012168

[Zac87] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

Part IV

Apendixes

Appendix A

Mathematical Notations

The mathematical notations used in the DAVINCI series are explained briefly in this appendix.

A.1 Sets

In addition to the set operations: $\cup, \cap, \setminus, \subseteq$ with their usual meaning, we also define:

$$\begin{aligned} A \subset B &\triangleq A \subseteq B \wedge A \neq B \\ A \not\subset B &\triangleq \neg A \subset B \end{aligned}$$

The power set of a set A, i.e. the set of all subsets of A, is denoted as $\wp(A)$, where:

$$\wp(A) \triangleq \{ B \mid B \subseteq A \}$$

A.2 Functions

A partial function f from A to B is defined by $f : A \rightarrowtail B$. Formally, it is a relation $f \subseteq A \times B$ such that $\langle a, b\rangle \in f \wedge \langle a, c\rangle \in f \Rightarrow b = c$. This property makes it possible to write $f(a) = b$ instead of $\langle a, b\rangle \in f$.

A function f is a set of binary tuples. The first and second values of these binary tuples are identified as:

$$\begin{aligned} \pi_1(f) &\triangleq \{ a \mid \langle a, b\rangle \in f \} \\ \pi_2(f) &\triangleq \{ b \mid \langle a, b\rangle \in f \} \end{aligned}$$

The following abbreviations are used for (partial) functions:

$$\begin{aligned} \mathsf{dom}(f) &\triangleq \pi_1(f) \\ \mathsf{ran}(f) &\triangleq \pi_2(f) \\ f(a)\downarrow &\triangleq a \in \mathsf{dom}(f) \\ f(a)\uparrow &\triangleq a \notin \mathsf{dom}(f) \end{aligned}$$

For unary functions, we will write $f\downarrow a$, and $f\uparrow a$, instead of $f(a)\downarrow$, and $f(a)\uparrow$ respectively. Further, $f_1, \ldots, f_n\downarrow a_1, \ldots, a_m$ is employed as an abbreviation for: $\forall_{1\leq i\leq n} \forall_{1\leq j\leq m} [f_i\downarrow a_j]$.

A total function f from A to B is defined by $f : A \rightarrow B$. Formally, $f : A \rightarrowtail B$ for which $\mathsf{dom}(f) = A$.

A.3 Relations

If $R \subseteq X \times X$ is a relation, then we will use: $x\, R\, y\, R\, z$ as an abbreviation of: $x\, R\, y \wedge y\, R\, z$.

Appendix B

Answers to questions

B.1 Questions from Section 1.4

Version: 04-04-05

1. What is an information system? Give some examples of an information system.
2. What is information system development? What does 'architecture-driven' add to this?
3. What are the consequences of an evolving environment on information systems?
4. Sometimes organisations use the introduction of IT as a way to force changes in organisations.

   Why is this likely to fail?
5. Why is complexity a challenge for information system development?

   What will happen if you add evolution to the equation?
6. Why is it important to use an 'architecture-driven' approach for the development of systems that are used by some organisation in achieving its goals?

   Which role is played by the environment of the organisation?

   What is the role of the possible evolution of the organisation and its environment?
7. What is the essence of architecture in an information systems context?
8. Explain, in your own words, the essence of alignment.
9. Why is it important to optimise the alignment of an information system to its context?

B.2 Questions from Chapter 2

Version: 16-03-05

1. How are the terms 'organization', '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 the formal language given in this chapter.

   Answer:
   In natural language: The universe is the world under investigation by a viewer. A domain is a part or aspect of the universe in which the viewer is interested. A domain always has a direct environment. An organization is positioned in a universe, therefore being also part of a universe, combining at least one domain. So, here, a universe can be seen as the whole, containing organizations. Organizations contain domains.

In formal language: Let U be a universe observed by some viewer v. Let furthermore D be a domain with direct environment E_D (as part of a conception C) as it is observed by v in U. Then we formally have: $U \models^{\mathsf{d}}_v \langle C : E_D : D \rangle$.

The viewer v may see (in the domain D) an organizational system S with environment E_S. In other words: $U \models^{\mathsf{s}}_v \langle C : E_S : S \rangle$, where $E_S \subseteq E_D \cup D$ and $O \subseteq D$.

2. Proof Corollary 2.2.1 (page 53).

Answer:
Let $C \models^{\mathsf{d}}_v \langle C : E : D \rangle$, then the closedness of $E \cup D$ can be proven as follows:

$C \models^{\mathsf{d}}_v \langle C : E : D \rangle$	$\Rrightarrow$ {Axiom **S7** and **S8** (page 53)}
D and E are closed	$\Rrightarrow$ {Definition of closedness}
$\forall_{r \in \mathcal{LI}_E} [\mathsf{Involved}(r) \subseteq E] \wedge \forall_{r \in \mathcal{LI}_D} [\mathsf{Involved}(r) \subseteq D]$	$\Rrightarrow$ {Basic set theory}
$\forall_{r \in \mathcal{LI}_E \cup \mathcal{LI}_D} [\mathsf{Involved}(r) \subseteq E \vee \mathsf{Involved}(r) \subseteq D]$	$\Rrightarrow$ {Basic set theory}
$\forall_{r \in \mathcal{LI}_{E \cup D}} [\mathsf{Involved}(r) \subseteq E \cup D]$	$\Rrightarrow$ {Definition of closedness}
$E \cup D$ is closed	$\square$

3. Proof Lemma 2.2.1 (page 54).

Answer:
Let $U \models^{\mathsf{d}}_v \langle C : E : D \rangle$. Suppose P be a maximum connected subset of E, while:

$$\neg \exists_{e \in \mathcal{CO}_P} \exists_{r \in \mathcal{LI}_C, d \in \mathcal{CO}_D} [\{e, d\} = \mathsf{Involved}(r)]$$

In other words, suppose P is *not* connected to D.

As P is a maximum connected subset of E (and also non-empty), there is no other element in E to which an element from P is connected to. If P is *not* connected to D it would mean that some subset of E is not connected to D, which would contradict Axiom **S10**. We can therefore not assume:

$$\neg \exists_{e \in \mathcal{CO}_E} \exists_{r \in \mathcal{LI}_C, d \in \mathcal{CO}_D} [\{e, d\} = \mathsf{Involved}(r)]$$

We should therefore conclude:

$$\exists_{e \in \mathcal{CO}_E} \exists_{r \in \mathcal{LI}_C, d \in \mathcal{CO}_D} [\{e, d\} = \mathsf{Involved}(r)]$$

4. Not all conceptions of a domain produce models. Why not?

Answer:
Conceptions need to be described in terms of a description. Conceptions are made by viewers, but are not made on purpose. If a conception is made on purpose, so if a domain is purposely abstracted, it is called a model. Therefore, not all conceptions produce models.

5. 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).

Answer:

- Active System: an example of an active system is the traffic jam information supply system. These are the LED-signs along highways supplying information about which of the highways are jammed, where they are jammed and the length of the jam. Drivers can, based on the supplied information, choose to take a different route than their initial route. This way, the system actively changes the universe.

- Dynamic System: an example of a dynamic system is the Dutch political system. The system keeps on existing, but it changes due to new laws, the abolishing of old laws, etcetera.
- Open System: an example of an open system is a 'smart' chess computer. Consider a chess computer with a set of standard possible sets of movements. Each time you make a move which is unknown to the system, the system remembers the move. After a few games, the chess computer has 'learned' from your input and, as an effect, becomes harder to beat.

6. From a modeling point of view, organizations can be considered as systems containing a.o. entities and relations.
   - Why is it important to be aware of the aspect of subjectivity when creating models?
   - What view does an information system developer have when modeling organizations?
   - Why would an information system developer want to start by creating a model of an organization, instead of directly focusing on modeling an information system?

   Answer:

   - Models are based on conceptions, which are viewer-bound. Therefore, models are described from the modelers' point of view. These models also include the worldview of the modeler, and are thus a subjective representation of a domain.
   - An information system developer describes a domain from a certain point of view, namely his own point of view, and with a certain purpose, namely to create a model of the domain which on which an information system shall/could be based. So, he only look from one point of view within his own point of view to the domain.
   - By creating a model of an organization before creating a model of a possible information system which should be deployed in the organization, the developer is forced to gain knowledge about the organization in which he operates. This should lead to better insight in several aspects of the organization, like structure, evolution, culture, etcetera. The use of this knowledge when creating an information system should lead to a better 'fitting' system.

7. Proof Corollary 2.2.2 (page 55).

   Answer:
   Because of Axiom **S12**, we already know:

   $$\exists_{u \in \mathcal{CC}_C} \forall_{x \in \mathcal{CC}_C} [u \rightrightarrows_C x]$$

   But is there only one such u?

   Suppose we have $u_1, u_2 \in \mathcal{CC}_C$ such that $u_1 \neq u_2$ while:

   $$\forall_{x \in \mathcal{CC}_C} [u_1 \rightrightarrows_C x \wedge u_2 \rightrightarrows_C x]$$

   In other words, both u_1 and u_2 are tops of the decomposition hierarchy.

   This allows us to derive:

   $$\begin{array}{ll} u_1 \neq u_2 \wedge \forall_{x \in \mathcal{CC}_C} [u_1 \rightrightarrows_C x \wedge u_2 \rightrightarrows_C x] & \Rrightarrow \{\text{Since } u_1, u_2 \in \mathcal{CC}_C\} \\ u_1 \rightrightarrows_C u_2 \wedge u_2 \rightrightarrows_C u_1 & \Rrightarrow \{u_1 \neq u_2\} \\ u_1 \rightarrow_C u_2 \wedge u_2 \rightarrow_C u_1 & \Rrightarrow \{\text{Transitivity of } \_ \rightarrow_C \_\} \\ u_1 \rightarrow_C u_1 & \square \end{array}$$

As stipulated by Axiom **S11** (page 55), decompositions are acyclic. Therefore, we are noy allowed to have $u_1 \rightarrow_C u_1$. We therefore cannot have $u_1, u_2 \in \mathcal{CO}_C$ such that $u_1 \neq u_2$ while:

$$\forall_{x \in \mathcal{CO}_C} [u_1 \rightrightarrows_C x \wedge u_2 \rightrightarrows_C x]$$

Therefore there can only be one such u.

8. Suppose you are requested by a large organization (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:
   (a) Where would you start modeling?
   (b) What would you model?
   (c) Why model that?

   Answer:

   (a) When a model of a very large organization is required, one should start by focusing on the problem, instead of focusing on a to-be-created model. The first logical step is gaining some knowledge of the whole of the organization. This can indeed be done by creating a (holistic) sketch of the complete organization.
   (b) When that sketch is made, and if the board agrees on that sketch, the modeler can start focusing on several aspects concerning information flows. Perhaps an iterative approach might be useful: expanding a model step by step, keeping the whole of the organization in mind.
   (c) Before an analysis can be made of a problem, or before a problem can be identified or localized, some research has to be done. A board of directors can have some ideas of where the problem may lie, but they may be wrong as well. Simply following orders of some board may then result in the wrong conception of the organization, and thus to a correct solution for some problem, but not to a solution for their problem. In fact, the solution then does not 'fit', or it changes, making it perhaps even more difficult to find. Therefore, a model of the whole should be made: to gain insight in the organization and their problem.

9. Proof Corollary 2.3.1 (page 60).

   Answer:
   Suppose $U \models^{\mathsf{s}}_v S' \subset S$, then:

$$\begin{array}{ll}
U \models^{\mathsf{s}}_v S' \subset S & \Rrightarrow \{\text{definition of sub-system}\} \\
U \models^{\mathsf{s}}_v S' & \Rrightarrow \{\text{definition of abbreviation}\} \\
\exists_{C,E} [U \models^{\mathsf{s}}_v \langle C : E : S' \rangle] & \Rrightarrow \{\text{Axiom } \mathbf{S19}\} \\
\exists_{C,E} [U \models^{\mathsf{m}}_v \langle C : E : S' \rangle] & \Rrightarrow \{\text{Axiom } \mathbf{S18}\} \\
\exists_{C,E} \left[U \models^{\mathsf{d}}_v \langle C : E : S' \rangle\right] & \Rrightarrow \{\text{Axiom } \mathbf{S13}\} \\
\exists_{d \in S'} \forall_{x \in \mathcal{CO}_{S'}} [d \rightrightarrows_{S'} x] & \square
\end{array}$$

10. Consider a home cinema set.
    (a) Describe the systems elements.
    (b) Distinguish proper sub-systems.

(c) Can you derive typical aspect systems and component systems?

Explain your answers.

Answer:

(a) A home cinema set usually contains of some elements of input (a receiver, a DVD-player, a VCR, a game-console, a remote control, etcetera), and some elements of output (a television screen, and some speakers). The relations between these elements are the cables and the radio- or infra-red signals broadcasted by the RC.

(b) possible sub-systems are: the receiver, the dvd-player, the VCR, a game-sonsole, a remote-control, a television, a set of speakers, the video-stream, the audio-stream, the input-streams, the output-streams, and the power-supply.

(c) A list of proper component sub-systems is already given in the first item. Each of these elements can properly function in other settings, and can thus be considered as component sub-system. When looking at aspect systems, input-systems, output-systems, video-stream system, audio-stream system and the power-supply. Please recognize the difference between a speaker and a set of speakers. A speaker can be identified as a sub-system itself, but since a set of speakers (for example in a Dolby 5.1 setting) has certain properties which individual speakers do not have (surround sound), a set of speakers can be considered as a sub-system just as well as a single speaker.

11. Consider a travel agency.

(a) Describe the most important system characteristics and exposition characteristics.

(b) Describe its behavior in terms of internal and external functions.

Answer:

(a) Consider a travel agency as an open active system: it anticipates to a lot of factors in its environment, like customers, travels offered by airline companies, etcetera. On the other hand, large travel agencies can promote certain destinations by promotion, or enforce a quantity rebate at some airline company or hotel. This way they have influence on the world in which they function.

- system characteristics: The agency is thus considered as an open active system.
- exposition characteristics: The travel agency mainly plays three roles: the customer role (buying tickets at hotels and airline agencies), the vendor-role (selling travels to customers) and the mediator-role (searching the best fitting travel for each customer)

(b)
- Internal functions: examples of internal functions are: the management of the agency, the administration of the agency, etcetera.
- External functions: examples of external functions are: the number of customers interested in traveling, the supply of possible travels by hotels and airline companies, etcetera.

12. Describe why *information* systems contain *data*bases. Use the descriptions of the terms data, information and knowledge as described in Chapter 2 in your description.

Answer:
Databases contain loads of data. Data, which is only useful when placed in some context and when the data is related to each other in some way. This is usually done by queries built in an information system. The system gains some data from a database and displays it in context to the user of the system. By placing it in context, the data becomes information. Due to the information supplied by the system, the view on the world of the systems' user may be changed (or in other words: it may change the knowledge of the user).

13. Consider our definition of information intensive organizations.
    (a) Give some examples of:
        i. Information intensive organizations
        ii. Non-information-intensive organizations
    (b) How would you describe the distinction between information intensive organizations and non-information organizations?

    Answer:

    (a) Some examples:
        - An example of an information intensive organization is a bank. Banks work large amounts of money, mostly computerized money (Internet banking, cash machines, electronic payment methods, etcetera.), so it is quite necessary that the bank has well-functioning information systems. Failure may lead to the banks' bankrupt.
        - An example of a non-information intensive organization is an Amish farm. Since the Amish refuse to use modern equipment, like information systems, it is very likely that an Amish farm is a non-information-intensive organization.
    (b) The mayor difference between information intensive organizations and non-information intensive organizations is the importance of the role the information system as a subsystem of the organization plays. If, in some organization, the information system becomes more important, then the organization becomes more information intensive.
14. Describe, in your own words, the differences between knowledge, information and data.

    Answer:
    Data are loose building blocks, for example one term or one fact. They are quite meaningless without any context. They are the smallest units of which information can be constructed. Information is data placed in some context, thereby giving it some meaning. This meaning may have influence on someone's knowledge. Knowledge represents someones view on the world.
15. Proof Corollary 2.5.1 (page 66).

    Answer:
    Suppose $t_1 \rhd t_2 \wedge t_1 \rhd t_3 \wedge t_2 \neq t_3$, then this would lead to the following contradiction:

$$t_1 \rhd t_2 \wedge t_1 \rhd t_3 \wedge t_2 \neq t_3$$

$$\Rrightarrow \{\text{since } < \text{ is a complete total order}\}$$

$$t_1 \rhd t_2 \wedge t_1 \rhd t_3 \wedge (t_2 < t_3 \vee t_3 < t_2)$$

$$\Rrightarrow \{\text{definition of } \rhd\}$$

$$t_1 < t_3 \wedge \neg\, \exists_s\, [t_1 < s < t_2] \wedge t_1 < t_3 \wedge \neg\, \exists_s\, [t_1 < s < t_3] \wedge (t_2 < t_3 \vee t_3 < t_2)$$

$$\Rrightarrow \{\text{rewrite}\}$$

$$\neg\, \exists_s\, [t_1 < s < t_2] \wedge \neg\, \exists_s\, [t_1 < s < t_3] \wedge (t_1 < t_2 < t_3 \vee t_1 < t_3 < t_2)$$

$$\Rrightarrow \{\text{rewrite}\}$$

$$\neg\, \exists_s\, [t_1 < s < t_2 \vee t_1 < s < t_3] \wedge (t_1 < t_2 < t_3 \vee t_1 < t_3 < t_2)$$

$$\Rrightarrow \{\text{contradiction}\}$$

$$\text{FALSUM}$$

$$\square$$

Therefore, if $t_1 \rhd t_2 \wedge t_1 \rhd t_3$ it cannot be that $t_2 \neq t_3$. In other words, we must have $t_2 = t_3$.

16. Proof Corollary 2.5.2 (page 67).

Answer:
We provide the prove of $H_E(t)(t) \subseteq H_C(t)$. The prove for $H_E(t)(t) \subseteq H_C(t)$ goes analogously.

Suppose $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then:

$$
\begin{array}{ll}
U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle & \Rrightarrow \{H_E \text{ is a function to } \wp\mathcal{O}(H_C)\} \\
H_E(t) \in \wp\mathcal{O}(H_C) & \Rrightarrow \{\text{definition of powerset}\} \\
H_E(t) \subseteq H_C & \Rrightarrow \{\text{rewrite}\} \\
h \in H_E(t) \Rightarrow h \in H_C(t) & \Rrightarrow \{h \text{ is a function } h : \mathcal{TI} \rightarrowtail \mathcal{EL}\} \\
\left\{ h(t) \;\middle|\; h \in H_E(t) \right\} \subseteq \left\{ h(t) \;\middle|\; h \in H_C \right\} & \Rrightarrow \{\text{definition of shorthand } H(t)\} \\
H_E(t)(t) \subseteq H_C(t) & \square
\end{array}
$$

17. Proof Corollary 2.5.3 (page 67).

Answer:
Let $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, then:

$$
\begin{array}{ll}
U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle & \Rrightarrow \{\text{Axiom } \textbf{S21} \text{ (page 67)}\} \\
\forall_{t \in \mathcal{TI}} \left[U \models^{\mathsf{d}}_v \langle H_C(t) : H_E(t)(t) : H_D(t)(t) \rangle \right] & \Rrightarrow \{\text{Axiom } \textbf{S6} \text{ (page 53)}\} \\
\forall_{t \in \mathcal{TI}} \left[H_E(t)(t) \cap H_D(t)(t) = \emptyset \right] & \square
\end{array}
$$

18. Proof Lemma 2.5.1 (page 67).

Answer:
Let $U \models^{\mathsf{d}}_v \langle H_C : H_E : H_D \rangle$, while $\neg \forall_{t \in \mathcal{TI}} \left[H_E(t) \cap H_D(t) = \emptyset \right]$, then:

$$
\begin{array}{ll}
\neg \forall_{t \in \mathcal{TI}} \left[H_E(t) \cap H_D(t) = \emptyset \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \left[H_E(t) \cap H_D(t) \neq \emptyset \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \exists_h \left[h \in H_E(t) \cap H_D(t) \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \exists_h \left[h \in H_E(t) \wedge h \in H_D(t) \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \exists_h \left[h(t) \in H_E(t)(t) \wedge h(t) \in H_D(t)(t) \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \exists_e \left[e \in H_E(t)(t) \wedge e \in H_D(t)(t) \right] & \Rrightarrow \{\text{rewrite}\} \\
\exists_{t \in \mathcal{TI}} \left[H_E(t)(t) \cap H_D(t) \neq \emptyset \right] & \square
\end{array}
$$

This result would contradict Corollary 2.5.3. In other words, we must have $\forall_{t \in \mathcal{TI}} \left[H_E(t) \cap H_D(t) = \emptyset \right]$.

B.3 Questions from Chapter 3

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. Stel je maakt een ontwerp voor een geldautomaat. Wat zijn voor dat domein de belangrijkste concepten en hun onderlinge links? Hoe werken ze samen?
3. Proof Corollary 3.4.1 (page 75).

   Answer:
   We will prove $\mathcal{FC}_t = \mathsf{Fact}(\mathcal{RO}_t)$. The prove of $\mathcal{PC}_t = \mathsf{Player}(\mathcal{RO}_t)$ goes analogously.

$$
\begin{array}{ll}
f \in \mathsf{Fact}(\mathcal{RO}_t) & \equiv \{\text{Definition of } \mathsf{Fact}\} \\
\exists_{r \in \mathcal{RO}_t}\,[f = \mathsf{Fact}(r)] & \equiv \{\text{Axiom } \mathbf{S29}\} \\
\exists_{r \in \mathcal{RO}_t}\,[f = \mathsf{Fact}(r) \wedge \mathsf{Fact}(r) \in \mathcal{FC}_t] & \equiv \{\text{Rewrite}\} \\
\exists_{r \in \mathcal{RO}_t}\,[f = \mathsf{Fact}(r) \wedge f \in \mathcal{FC}_t] & \equiv \{\text{Rewrite}\} \\
\exists_{r \in \mathcal{RO}_t}\,[f = \mathsf{Fact}(r)] \wedge f \in \mathcal{FC}_t & \equiv \{\text{Definition of } \mathcal{FC}\} \\
f \in \mathcal{FC} \wedge f \in \mathcal{FC}_t & \equiv \{\text{Definition of } \mathcal{FC}_t\} \\
f \in \mathcal{FC}_t & \Box
\end{array}
$$

4. Proof Corollary 3.5.1 (page 77).

   Answer:

$$
\begin{array}{ll}
x \in \mathcal{IN} & \Rrightarrow \{\text{Definition of } \mathcal{IN}\} \\
x \in \left\{x \;\middle|\; \exists_y\,[x\ \mathsf{HasType}\ y]\right\} & \Rrightarrow \{\text{Rewrite}\} \\
\exists_y\,[x\ \mathsf{HasType}\ y] & \Rrightarrow \{\text{Definition of } \mathcal{TP}\} \\
\exists_{y \in \mathcal{TP}}\,[x\ \mathsf{HasType}\ y] & \Box
\end{array}
$$

   Therefore: $\forall_{x \in \mathcal{IN}} \exists_{y \in \mathcal{TP}}\,[x\ \mathsf{HasType}\ y]$
5. Proof Corollary 3.5.2 (page 77).

   Answer:
   We will prove: $\hat{\mathcal{PC}} \triangleq \mathsf{Player}(\hat{\mathcal{RO}})$, the other proofs are analogously.

$$
\begin{array}{ll}
p \in \hat{\mathcal{PC}} & \equiv \left\{\text{Definition of } \hat{\mathcal{PC}}\right\} \\
p \in \mathcal{PC} \wedge p \in \mathcal{TP} & \equiv \{\text{Definition of } \mathcal{PC}\} \\
p \in \mathsf{Player}(\mathcal{RO}) \wedge p \in \mathcal{TP} & \equiv \{\text{Definition of } \mathsf{Player}\} \\
\exists_{r \in \mathcal{RO}}\,[p = \mathsf{Player}(r) \wedge p \in \mathcal{TP}] & \equiv \{\text{Rewrite}\} \\
\exists_{r \in \mathcal{RO}}\,[p = \mathsf{Player}(r) \wedge \mathsf{Player}(r) \in \mathcal{TP}] & \equiv \{\text{Axiom } \mathbf{S34}\} \\
\exists_{r \in \mathcal{RO}}\,[p = \mathsf{Player}(r) \wedge r \in \mathcal{TP}] & \equiv \{\text{Rewrite}\} \\
\exists_{r \in \mathcal{RO} \cap \mathcal{TP}}\,[p = \mathsf{Player}(r)] & \equiv \left\{\text{Definition of } \hat{\mathcal{RO}}\right\} \\
\exists_{r \in \hat{\mathcal{RO}}}\,[p = \mathsf{Player}(r)] & \equiv \{\text{Definition of } \mathsf{Player}\} \\
\mathsf{Player}(\hat{\mathcal{RO}}) & \Box
\end{array}
$$

Note: for the Fact versions Axiom **S33** should be used rather than Axiom **S34**.

6. Proof Corollary 3.5.5 (page 79).

   Answer:
   Let $f \in \hat{\mathcal{FC}}$, then:

$$
\begin{array}{ll}
i \in \mathsf{Pop}(f) \wedge f \in \hat{\mathcal{FC}} & \equiv \{\text{Definition of } \mathcal{FC}\} \\
\exists_r \left[\mathsf{Fact}(r) = f \wedge i \in \mathsf{Pop}(f)\right] & \equiv \{\text{Definition of RolesOf}\} \\
\exists_{r \in \mathsf{RolesOf}(f)} \left[i \in \mathsf{Pop}(\mathsf{Fact}(r))\right] & \equiv \{\text{Axioms } \mathbf{S37} \text{ and } \mathbf{S38}\} \\
\exists_{r \in \mathsf{RolesOf}(f)} \left[i \in \mathsf{Fact}(\mathsf{Pop}(r))\right] & \equiv \{\text{Definition of Pop}\} \\
i \in \mathsf{Fact}(\mathsf{Pop}(\mathsf{RolesOf}(f))) & \Box
\end{array}
$$

7. Proof Corollary 3.5.4 (page 79).

   Answer:
   Let $f \in \hat{\mathcal{FC}}$, then:

$$
\begin{array}{ll}
x \in \mathsf{Types}(\mathsf{RolesOf}(f)) & \equiv \{\text{Definition of Types}\} \\
x \in \hat{\mathcal{TP}} \wedge \exists_{r \in \mathsf{RolesOf}(f)} \left[x \in \mathsf{Types}(r)\right] & \equiv \{\text{Rewrite}\} \\
x \in \hat{\mathcal{TP}} \wedge \exists_{r \check{\mathcal{RO}}} \left[x \in \mathsf{Types}(r) \wedge r \in \mathsf{RolesOf}(f)\right] & \equiv \{\text{Definition of RolesOf}\} \\
x \in \hat{\mathcal{TP}} \wedge \exists_r \left[x \in \mathsf{Types}(r) \wedge \mathsf{Fact}(r) = f\right] & \equiv \{\text{Since } r \in \check{\mathcal{RO}} \text{ and Axiom } \mathbf{S33}\} \\
x \in \hat{\mathcal{RO}} \wedge \exists_r \left[x \in \mathsf{Types}(r) \wedge \mathsf{Fact}(r) = f\right] & \equiv \{\text{Definitions of Types and Pop}\} \\
x \in \hat{\mathcal{RO}} \wedge \exists_r \left[r \in \mathsf{Pop}(x) \wedge \mathsf{Fact}(r) = f\right] & \equiv \{\text{Definitions of Fact}\} \\
x \in \hat{\mathcal{RO}} \wedge f \in \mathsf{Fact}(\mathsf{Pop}(x)) & \equiv \{\text{Axioms } \mathbf{S37} \text{ and } \mathbf{S38}\} \\
x \in \hat{\mathcal{RO}} \wedge f \in \mathsf{Pop}(\mathsf{Fact}(x)) & \equiv \{\text{Definition of Types and Pop}\} \\
x \in \hat{\mathcal{RO}} \wedge \mathsf{Fact}(x) \in \mathsf{Types}(f) & \equiv \{\text{Definition of RolesOf}\} \\
x \in \mathsf{RolesOf}(\mathsf{Types}(f)) & \Box
\end{array}
$$

8. Consider the following case:

   Een onderneming produceert en verkoopt een tiental soorten gevulde chocoladeartikelen. 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 €0,10 per stuk. Op de artikelen met een prijs tot €5,- komt één, op de overige artikelen (tussen €5,- en €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.

B.4 Questions from Chapter 4

1. 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?
2. 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.

B.5 Questions from Chapter 6

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 entities and relationships depicting this domain.

2. Consider the following domain:

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

   Wat zijn hier de entiteiten en de relaties? Wat zijn de acties, actoren, actanden en predications.?

3. Stel je maakt een ontwerp voor een geldautomaat. Wat zijn voor dat domein de belangrijkste systeem entiteiten en hun onderlinge relaties? Hoe werken ze samen? Wat zijn hier de entiteiten en de relaties? Wat zijn de acties, actoren, actanden en predications.?

4. Proof Corollary 3.4.1 (page 75).
5. Proof Corollary 3.5.1 (page 77).
6. Consider the case from Question 3.8.

   Answer the following questions:

   (a) (Re)produce elementary facts for this domain.
   (b) What are the actions, actors and actands?

B.6 Questions from Chapter 7

1. 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 activity model voor.
2. Consider the domain as discussed in Question 3.8. Produce an activity model for this domain.
3. 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 gentroduceerde clubcard waarmee klanten kortingen bij alle aangesloten bedrijven kunnen krijgen. Inmiddels is er al een groot aantal klanten met een clubcard.

   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 credit card) 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 clubcard heeft en adviseert op basis van de klantbehoefte een auto uit een bepaalde tariefgroep. Hij controleert daarbij ook of er zoń 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 accoord 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 bijvoorbeeld een afkoop eigen risicio of een inzittenden verzekering. Dit wordt ook in de offerte opgenomen. Wanneer de klant ingaat op de offerte, dan maakt de verkoper een huurcontract nadat hij de credietwaardigheid 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 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 copie 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 accoord 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 zo'n 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.

Analyseer nu het hierboven beschreven domein en produceer hier een activity model voor.

4. Consider the claim handling process with quantative information as depicted in Figure 7.11. 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.

B.7 Questions from Chapter 8

1. 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?

2. 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.

B.8 Questions from Chapter 10

1. Why is it important to realise that system development is not necessarily a linear process?
2. Name three different axes along which a high-level design may be 'refined' to a more detailed design.
3. Describe, in your own words, the relationships between the concepts of: domain, viewer, conception, perception and architecture.

4. What is the difference between: an architecture, architectural description, architectural view, and architectural viewpoint?
5. Why is it important to carefully select relevant system viewpoints when developing systems?
6. Make a meta-model of the concepts introduced by the IEEE recommended practice for architecture [IEE00]. Make sure to include as many constraints as you can deduce from the text.
7. Why is it important to acknowledge the fact that different stakeholders' will have different views on a pre-existing or a future system?
8. Make a meta-model of the concepts introduced with respect to systems, domains, viewers, etc.
9. Identify, for the system description language you have see so-far as part of your studies:
   (a) The main concepts of these languages.
   (b) Typical interests and viewers for which these languages may be useful.
10. Why should the set of viewpoints that will be used to describe different aspects of a system that is being developed, be selected carefully.
11. What makes us consider a system description to be an architecture description?
12. What are possible dimensions for refinements of system descriptions?
13. Why is it important to consider responsibilities of system entities and collaborations among them?
14. Consider a candy vending machine and people purchasing candy from the machine. What would, in such a domain, be the relevant system entities? What would be there responsibilities and collaborations?
15. Suppose you would design a pocket calculator. What would be the essential system elements (at a functional level)? Maybe you could do a role-playing game with a group
16. What's the difference between an information system and a medium system?
17. Consider a bank. Not the one you sit on, but the one you entrust with your money. Provide a brief discussion of a possible business, work and information view (according to the Tapscott viewpoints) for such a domain.

Bibliography

[Ack71] R.L. Ackoff. Towards a system of system concepts. *Management Science*, 17, July 1971.

[AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[AH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[Alt99] S. Alter. A general, yet useful theory of information systems. *Communications of the Association for Information Systems*, 1(13), 1999.
`http://cais.isworld.org/articles/1-13/default.asp`

[Alt02] S. Alter. The work system method for understanding information systems and information system research. *Communications of the Association for Information Systems*, 9(9):90–104, 2002.
`http://cais.isworld.org/articles/default.asp?vol=9&art=6`

[Avi95] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, USA, 2nd edition, 1995. ISBN 0077092333

[BB97] F.C. Berger and P. van Bommel. Augmenting a characterization network with semantical information. *Information Processing & Management*, 33(4):453–479, 1997.

[BBMP95] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, and H.A. (Erik) Proper. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3):213–235, 1995.

[BCK98] L. Bass, P.C. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, Massachusetts, USA, 1998. ISBN 0201199300

[BCN92] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin Cummings, Redwood City, California, 1992.

[Bem98] T.M.A. Bemelmans. *Bestuurlijke Informatiesystemen en Automatisering*. Kluwer, Deventer, The Netherlands, EU, 7th edition, 1998. In Dutch. ISBN 9026727984

[Ber01] L. von Bertalanffy. *General Systems Theory – Foundations, Development, Applications*. George Braziller, New York, New York, USA, revised edition, 2001. ISBN 0807604534

[BFW96] P. van Bommel, P.J.M. Frederiks, and Th.P. van der Weide. Object-Oriented Modeling based on Logbooks. *The Computer Journal*, 39(9):793–799, 1996.

[BHW91] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[Boa99] B.H. Boar. *Practical steps for aligning information technology with business strategies*. Wiley, New York, New York, 1999. ISBN 0471076376

[BP88] B.W. Boehm and P.N. Papaccio. Understanding and controlling software costs. *IEEE Transactions of Software Engineering*, 14(10):1462–1477, October 1988.

[BPH04] A.I. Bleeker, H.A. (Erik) Proper, and S.J.B.A. Hoppenbrouwers. The role of concept management in system development – a practical and a theoretical perspective. In J. Grabis, A. Persson, and J. Stirna, editors, *Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004)*, pages 73–82, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU. ISBN 998497670X

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999. ISBN 0201571684

[Bro95] F. Brooks. *The Mythical Man-Month; anniversary edition*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201835959

[Bub86] J.A. Bubenko. Information System Methodologies - A Research View. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*, pages 289–318. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1986.

[BW90] P.D. Bruza and Th.P. van der Weide. Assessing the Quality of Hypertext Views. *ACM SIGIR FORUM (Refereed Section)*, 24(3):6–25, 1990.

[BW91] P.D. Bruza and Th.P. van der Weide. The modelling and retrieval of documents using index expressions. *ACM SIGIR FORUM (Refereed Section)*, 25(2), 1991.

[BW92a] P. van Bommel and Th.P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8:269–292, 1992.

[BW92b] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.

[Cas00] M. Castells. *The Information Age: Economy, Society and Culture*. Volume 1 – The Rise of the Network Society. Blackwell, Oxford, United Kingdom, EU, 2nd edition, 2000. ISBN 0631221409

[Che76] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Che81] P. Checkland. *Systems thinking, systems practice*. John Wiley & Sons, New York, New York, USA, 1981. ISBN 0471279110

[CHP96] L.J. Campbell, T.A. Halpin, and H.A. (Erik) Proper. Conceptual Schemas with Abstractions – Making flat conceptual schemas more comprehensible. *Data & Knowledge Engineering*, 20(1):39–85, 1996.

[Coc01] S. Cochran. The rising cost of software complexity. *Dr. Dobb's Journal*, April 2001.

[Coh89] B. Cohen. Justification of formal methods for system specification. *Software Engineering Journal*, 4(1):26–35, January 1989.

[CP96] P.N. Creasy and H.A. (Erik) Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering*, 20(2):119–162, 1996.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-Oriented Systems Analysis – A model-driven approach*. Yourdon Press, Englewood Cliffs, New Jersey, USA, 1992. ASIN 0136299733

[EN94] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, Redwood City, California, 1994. Second Edition.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1:75–116, 1985.

[FKN+92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system de-velopment. *International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments*, 2(1):31–58, 1992.

[Fre97] P.J.M. Frederiks. *Object-Oriented Modeling based on Information Grammars*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1997. ISBN 9090103384

[FV99] M. Franckson and T.F. Verhoef, editors. *Introduction to ISPL*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, 1999. ISBN 9076304858

[FVSV+98] E.D. Falkenberg, A.A. Verrijn-Stuart, K. Voss, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, and R.K. and Stamper, editors. *A Framework of Information Systems Concepts*. IFIP WG 8.1 Task Group FRISCO, IFIP, Laxenburg, Austria, EU, 1998. ISBN 3901882014

[FW02] P.J.M. Frederiks and Th.P. van der Weide. Deriving and paraphrasing information grammars using object-oriented analysis models. *Acta Informatica*, 38(7):437–88, June 2002.

[FW04a] P.J.M. Frederiks and Th.P. van der Weide. Information modeling: the process and the required competencies of its participants. *Data & Knowledge Engineering*, 2004. To appear in a special issue on the NLDB 2004 conference.

[FW04b] P.J.M. Frederiks and Th.P. van der Weide. Information modeling: the process and the required competencies of its participants. In F. Meziane and E. Métais, editors, *9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004)*, volume 3136 of *Lecture Notes in Computer Science*, pages 123–134, Manchester, United Kingdom, EU, 2004. Springer-Verlag, Berlin, Germany, EU.

[Hal95] T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition, 1995.

[Hal01] T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001. ISBN 1558606726

[Ham90] M. Hammer. Re-engineering work: don't automate, obliterate. *Harvard Business Review*, 68(4):104–112, April 1990.

[HBP05] S.J.B.A. Hoppenbrouwers, A.I. Bleeker, and H.A. (Erik) Proper. Facing the conceptual complexities in business domain modeling. *Computing Letters*, 1(2):59–68, 2005.

[HL89] I. van Horenbeek and J. Lewi. *Algebraic specifications in software engineering: an introduction*. Springer-Verlag, Berlin, Germany, 1989.

[HM95] J.H. Holland and H. Mimnaugh, editors. *Hidden Order : How Adaptation Builds Complexity*. Perseus Press, Cambridge, Massachusetts, 1995. ISBN 0201442302

[Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[Hop03] S.J.B.A. Hoppenbrouwers. *Freezing Language; Conceptualisation processes in ICT supported organisations*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 2003. ISBN 9090173188

[Hor00] T.A. Horan. *Digital Places – Building our city of bits*. The Urban Land Institute (ULI), Washington DC, United States of America, 2000. ISBN 0874208459

[HP95] T.A. Halpin and H.A. (Erik) Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[HP98] A.H.M. ter Hofstede and H.A. (Erik) Proper. How to Formalize It? Formalization Principles for Information Systems Development Methods. *Information and Software Technology*, 40(10):519–540, October 1998.

[HPW93] A.H.M. ter Hofstede, H.A. (Erik) Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[HPW97] A.H.M. ter Hofstede, H.A. (Erik) Proper, and Th.P. van der Weide. Exploiting Fact Verbalisation in Conceptual Information Modelling. *Information Systems*, 22(6/7):349–385, September 1997.

[HV93] J.C. Henderson and N. Venkatraman. Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):4–16, 1993.

[HVH97] J.J.A.C. Hoppenbrouwers, B. van der Vos, and S.J.B.A. Hoppenbrouwers. Nl structures and conceptual modelling: Grammalizing for KISS. *Data & Knowledge Engineering*, 23(1):79–92, 1997.

[HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[HW94] A.H.M. ter Hofstede and Th.P. van der Weide. Fact Orientation in Complex Object Role Modelling Techniques. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 45–59, Townsville, Australia, July 1994.

[HW97] A.H.M. ter Hofstede and Th.P. van der Weide. Deriving Identity from Extensionality. *International Journal of Software Engineering and Knowledge Engineering*, 8(2):189–221, June 1997.

[IEE00] Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA, September 2000. ISBN 0738125180
`http://www.ieee.org`

[Iiv83] J. Iivari. Contributions to the theoretical foundations of systemeering research and the pioco model. Technical Report 150, University of Oulu, Oulu, Finland, EU, 1983. ISBN 9514215435

[ISO96a] *Information technology – Open Distributed Processing – Reference model: Architecture*, 1996. ISO/IEC 10746-3:1996(E).
`http://www.iso.org`

[ISO96b] *Information technology – Open Distributed Processing – Reference model: Foundations*, 1996. ISO/IEC 10746-2:1996(E).
`http://www.iso.org`

[ISO96c] ISO. *Kwaliteit van softwareprodukten*. ten Hagen & Stam, Den Haag, The Netherlands, 1996. In Dutch. ISBN 9026724306

[ISO98a] *Information technology – Open Distributed Processing – Reference model: Architectural semantics*, 1998. ISO/IEC 10746-4:1998(E).
`http://www.iso.org`

[ISO98b] *Information technology – Open Distributed Processing – Reference model: Overview*, 1998. ISO/IEC 10746-1:1998(E).
`http://www.iso.org`

[ISO01] *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001.
`http://www.iso.org`

[JLB+04] H. Jonkers, M.M Lankhorst, R. van Buuren, S.J.B.A. Hoppenbrouwers, M. Bonsangue, and L. van der Torre. Concepts for Modeling Enterprise Architectures. *International Journal of Cooperative Information Systems*, 13(3):257–288, 2004.

[Jon86] C.B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[JVB+03] H. Jonkers, G.E. Veldhuijzen van Zanten, R. van Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Guillen Scholten, S.J.B.A. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M.M. Lankhorst, D. van Leeuwen, H.A. (Erik) Proper, A. Stam, and L. van der Torre. Towards a Language for Coherent Enterprise Architecture Descriptions. In M. Steen and B.R. Bryant, editors, *7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 28–39, Brisbane, Australia, September 2003. IEEE Computer Society Press, Los Alamitos, California, USA. ISBN 0769519946

[Kee91] P.W.G. Keen. *Shaping the Future - Business Design Through Information Technology*. Harvard Business School Press, Boston, Massachusetts, USA, 1991. ISBN 0875842372

[Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In T.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.

[Koe03] B.I. Koerner. What is smart dust, anyway? *Wired*, 11(6), June 2003.
`http://www.wired.com/wired/archive/11.06/start.html?pg=10`

[Kri94] G. Kristen. *Object Orientation – The KISS Method, From Information Architecture to Information System*. Addison-Wesley, Reading, Massachusetts, USA, 1994. ISBN 0201422999

[Kru95] P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, November 1995.

[Kru00] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 2000. ISBN 0201707101

[KS92] G. Kotonya and I. Sommerville. Viewpoints for requirements definition. *IEE/BCS Software Engineering Journal*, 7(6):375–387, 1992.

[Lan71] B. Langefors. *Editorial notes to: Computer Aided Information Systems Analysis and Design*. Studentlitteratur, Lund, Sweden, EU, 1971.

[Lev79] A.Y. Levy. *Basic Set Theory*. Springer-Verlag, Berlin, Germany, 1979.

[Lin92] P. Lindgreen. A General Framework for Understanding Semantic Structures. In E.D. Falkenberg, C. Rolland, and E.N. El Sayed, editors, *Information System Concepts: Improving the understanding – Proceedings of the second IFIP WG8.1 working conference (ISCO-2)*, Alexandria, Egypt, April 1992. North Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU. ISBN 0444895078

[Lo05] M.M. Lankhorst and others. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, Berlin, Germany, EU, 2005. ISBN 3540243712

[LS80] B. Lientz and E. Swanson. *Software Maintenance Management – a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, Reading, Massachusetts, 1980. ISBN 0201042053

[Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. In Danish.

[McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309

[Mer03] Meriam-Webster Online, Collegiate Dictionary, 2003. `http://www.webster.com`

[MR02] M.W. Maier and R. Rechtin. *The Art of System Architecting*. CRC Press, Boca Raton, Florida, 2nd edition, 2002. ISBN 0849304407

[Neg96] N. Negroponte. *Being Digital*. Vintage Books, New York, New York, 1996. ISBN 0679762906

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630

[NP90] J. Nosek and P. Palvia. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 3(2):157–174, 1990.

[Ode00a] J. Odell. Agents (part 1): Technology and usage. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[Ode00b] J. Odell. Agents (part 2): Complex systems. Technical report, Cutter Consortium, Arlington, Massachusetts, USA, 2000.

[OHM+88] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verrijn-Stuart. *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley, Reading, Massachusetts, USA, 1988. ISBN 0201544431

[OMG03] OMG. UML 2.0 Superstructure Specification – OMG Draft Adopted Specification. Technical Report ptc/03-08-02, August 2003. `http://www.omg.org`

[OST83] T.W. Olle, H.G. Sol, and C.J. Tully, editors. *Information Systems Design Methodologies: A feature analysis*, York, England, EU, 1983. North Holland/IFIP WG8.1. ISBN 0444867058

[OSV82] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: A Comparative Review*. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1982. ISBN 0444864075

[OSV86] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: Improving the practice*, Noordwijkerhout, Netherlands, EU, 1986. North Holland/IFIP WG8.1.

[OSV88] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: Computerized assistance during the information systems life cycle*. North Holland/IFIP WG8.1, Malham, England, EU, 1988. ISBN 0444705120

[PB89] M.M. Parker and R.J. Benson. Enterprisewide information management: State-of-the-art strategic planning. *Journal of Information Systems Management*, (Summer):14–23, 1989.

[PB99] H.A. (Erik) Proper and P.D. Bruza. What is Information Discovery About? *Journal of the American Society for Information Science*, 50(9):737–750, July 1999.

[PBH04] H.A. (Erik) Proper, A.I. Bleeker, and S.J.B.A. Hoppenbrouwers. Object-role modelling as a domain modelling approach. In J. Grundspenkis and M. Kirikova, editors, *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)*, volume 3, pages 317–328, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU. ISBN 9984976718

[PBHJ00] H.A. (Erik) Proper, H. Bosma, S.J.B.A. Hoppenbrouwers, and R.D.T. Janssen. An Alignment Perspective on Architecture-driven Information Systems Engineering. In D.B.B. Rijsenbrij, editor, *Proceedings of the Second National Architecture Congres*, Amsterdam, The Netherlands, EU, November 2000.

[Pei69a] C.S. Peirce. *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 0674138007

[Pei69b] C.S. Peirce. *Volumes III and IV – Exact Logic and The Simplest Mathematics*. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA, 1969. ISBN 0674138005

[Pei69c] C.S. Peirce. *Volumes V and VI – Pragmatism and Pragmaticism and Scientific Metaphysics*. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA, 1969. ISBN 0674138023

[Pei69d] C.S. Peirce. *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 0674138031

[PH04] H.A. (Erik) Proper and S.J.B.A. Hoppenbrouwers. Concept evolution in information system evolution. In J. Gravis, A. Persson, and J. Stirna, editors, *Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU*, pages 63–72, Riga, Latvia, EU, June 2004. Faculty of Computer Science and Information Technology, Riga Technical University. ISBN 998497670X

[PPY01] M.P. Papazoglou, H.A. (Erik) Proper, and J. Yang. Landscaping the information space of large multi-database networks. *Data & Knowledge Engineering*, 36(3):251–281, 2001.

[Pro94] H.A. (Erik) Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[Pro97] H.A. (Erik) Proper. Data Schema Design as a Schema Evolution Process. *Data & Knowledge Engineering*, 22(2):159–189, 1997.

[Pro98] H.A. (Erik) Proper. Da Vinci – Architecture-Driven Business Solutions. Technical report, Origin, Amsterdam, The Netherlands, EU, Summer 1998.

[Pro01] H.A. (Erik) Proper, editor. *ISP for Large-scale Migrations*. Information Services Procurement Library. ten Hagen & Stam, Den Haag, The Netherlands, EU, 2001. ISBN 9076304882

[Pro03] H.A. (Erik) Proper. *Informatiekunde; Exacte vaagheid*. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU, November 2003. In Dutch. ISBN 9090172866

[Pro04] H.A. (Erik) Proper. *Da Vinci – Architecture-driven Information Systems Engineering*. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen, Nijmegen, The Netherlands, EU, 2004.

[PW94] H.A. (Erik) Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

[PW95a] H.A. (Erik) Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[PW95b] H.A. (Erik) Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Rec91] E. Rechtin. *Systems architecting: creating and building complex systems*. Prentice-Hall PTR, Upper Saddle River, New Jersey, 1991. ISBN 0138803455

[RMB95] J. Reeves, M. Marashi, and D. Budgen. A software design framework or how to support real designers. *IEE/BCS Software Engineering Journal*, 10(4):141–155, 1995.

[RMD99] V.E. van Reijswoud, J.B.F Mulder, and J.L.G. Dietz. Commucation Action Based Business Process and Information Modelling with DEMO. *The Information Systems Journal*, 9(2):117–138, 1999.

[Rop99] G. Ropohl. Philosophy of socio-technical systems. *In Society for Philosophy and Technology*, 4(3), 1999.

[SFG+00] J.J. Sarbo, J.I. Farkas, F.A. Grootjen, P. van Bommel, and Th.P. van der Weide. Meaning Extraction from a Peircean Perspective. *International Journal of Computing Anticipatory Systems*, 6:209–227, 2000.

[Sim62] H.A. Simon. The architecture of complexity. In *Proceedings of the American Philosophical Society*, volume 106, pages 467–482, 1962.

[Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0444867058

[Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.

[Spi88] J.M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge University Press, Cambridge, United Kingdom, EU, 1988.

[SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.

[Tap96] D. Tapscott. *Digital Economy - Promise and peril in the age of networked intelligence*. McGraw-Hill, New York, New York, USA, 1996. ISBN 0070633428

[TC93] D. Tapscott and A. Caston. *Paradigm Shift – The New Promise of Information Technology*. McGraw-Hill, New York, New York, USA, 1993. ASIN 0070628572

[TOG04] The Open Group. *TOGAF – The Open Group Architectural Framework*, 2004. `http://www.togaf.org`

[TP91] T.H. Tse and L. Pong. An Examination of Requirements Specification Languages. *The Computer Journal*, 34(2):143–152, April 1991.

[Vel92] J. in 't Veld. *Analyse van organisatieproblemen – Een toepassing van denken in systemen en processen*. Stenfert Kroese, Leiden, The Netherlands, EU, 1992. In Dutch. ISBN 9020722816

[Ver93] T.F. Verhoef. *Effective Information Modelling Support*. PhD thesis, Delft University of Technology, Delft, The Netherlands, EU, 1993. ISBN 9090061762

[VHP03] G.E. Veldhuijzen van Zanten, S.J.B.A. Hoppenbrouwers, and H.A. (Erik) Proper. System Development as a Rational Communicative Process. In N. Callaos, D. Farsi, M. Eshagian-Wilner, T. Hanratty, and N. Rish, editors, *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, volume XVI, pages 126–130, Orlando, Florida, USA, July 2003. ISBN 9806560019

[VHP04] G.E. Veldhuijzen van Zanten, S.J.B.A. Hoppenbrouwers, and H.A. (Erik) Proper. System Development as a Rational Communicative Process. *Journal of Systemics, Cybernetics and Informatics*, 2(4), 2004. `http://www.iiisci.org/Journal/sci/pdfs/P492036.pdf`

[VO91] A.A. Verrijn-Stuart and T.W. Olle, editors. *Methods and Associated Tools for the Information Systems Life Cycle*, Maastricht, Netherlands, EU, 1991. North Holland/IFIP WG8.1. ISBN 0444820744

[WAA85] A.T. Wood-Harper, L. Antill, and D.E. Avison. *Information Systems Definition: The Multiview Approach*. Blackwell Scientific Publications, Oxford, United Kingdom, EU, 1985. ISBN 0632012168

[WBW00] B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide. Matching Index Expressions for Information Retrieval. *Information Retrieval Journal*, 2(4), 2000. To appear.

[WBW01] B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide. Combining Boolean Logic and Linguistic Structure. *Information & Software Technology*, (43):53–59, 2001.

[WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag, Berlin, Germany, EU. ISBN 3540526250

[Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.

[WJK00] M. Wooldridge, N.R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Zac87] J.A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.

List of Symbols

$U \models^{\mathsf{s}}_v S' \subset_a S \triangleq U \models^{\mathsf{s}}_v S' \subset S$ and $(S' \cap \mathcal{LI}) \subset S$ – For **viewer** v viewing **universe** U, **system** S' is a **aspect system** of S

$U \models^{\mathsf{s}}_v S' \subset_c S \triangleq U \models^{\mathsf{s}}_v S' \subset S$ and $(S' \cap \mathcal{CO}) \subset S$ – For **viewer** v viewing **universe** U, **system** S' is a **component system** of S

$\mathcal{CE} \triangleq \wp(\mathcal{EE})$ – The set of **conception evolutions**.

$\_ \models^{\mathsf{c}}_\_ \_ \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL})$ – A relationship expressing which **conception** is held by which **viewer**. The fact that a **viewer** v harbours a **conception** C for **universe** U is expressed as $U \models^{\mathsf{c}}_v C$.

$\mathcal{CO}_X \triangleq X \cap \mathcal{CO}$ – A subset of the set of **concepts**.

$\mathcal{CO} \subseteq \mathcal{EL}$ – The set of **elements** of a **conception** that are **concepts**.

$x \rightrightarrows_C y \triangleq x = y \vee x \rightarrow_C y$ – A derived relationship providing the decomposition of a composed **concept** in some **viewer**'s **conception**. If $x \rightrightarrows_C y$, the **concept** x in **conception** C is decomposed into (possibly amongst others) **concept** y, or x and y are equal.

$\_ \rightarrow_\_ \_ \subseteq \mathcal{CO} \times \wp(\mathcal{EL}) \times \mathcal{CO}$ – A derived relationship providing the decomposition of a composed **concept** in some **viewer**'s **conception**. If $x \rightarrow_C y$, the **concept** x in **conception** C is decomposed into (possibly amongst others) **concept** y.

$\mathcal{DC} \subseteq \mathcal{LI}$ – Decomposer **links**.

$\_ \models^{\mathsf{d}}_\_ \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$ – A relationship expressing which **conception** of a **domain** and **environment** is held by which **viewer** with a particular **interest**. The fact that a **viewer** v harbours a **conception** C of domain D with environment E an for **universe** U is expressed as $U \models^{\mathsf{d}}_v \langle C : D : E \rangle$.

$\mathcal{EE} \triangleq \mathcal{TI} \rightarrowtail \mathcal{EL}$ – The set of **element evolutions**.

$\mathcal{EL}$ – The set of **elements** that may be part of a **conception**.

$\mathsf{From} : \mathcal{LI} \rightarrow \mathcal{CO}$ – The source **concept** of a **link**.

$\mathsf{Involved}(r) \triangleq \{\mathsf{From}(r), \mathsf{To}(r)\}$ – The set comprising the source and destination of a relationship from a conception.

$< \subseteq \mathcal{TI} \times \mathcal{TI}$ – A complete and total order over points in time.

$\mathcal{LI}_X \triangleq X \cap \mathcal{LI}$ – A subset of the set of **links**.

$\mathcal{LI} \subseteq \mathcal{EL}$ – The set of **elements** of a **conception** that are **links** between **concepts**.

$U \models^{\mathsf{m}}_v M \triangleq \exists_{C,E} [U \models^{\mathsf{m}}_v \langle C : E : M \rangle]$ – A relationship expressing which **model** is held by which **viewer**. The fact that a **viewer** v harbours a **model** M of part of **universe** U is expressed as $U \models^{\mathsf{m}}_v M$.

$\_ \models^{\mathsf{m}}_{\_} \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$ – A relationship expressing which **model** and **environment** or some part of the **universe** are held by which **viewer**. The fact that a **viewer** v with a **conception** C harbours a **model** M with **environment** E for a part of **universe** U is expressed as $U \models^{\mathsf{m}}_{v} \langle C : M : E \rangle$.

$t_1 \rhd t_2 \triangleq t_1 < t_2 \wedge \neg \exists_s [t_1 < s < t_2]$ – The next point in time. As $<$ is a complete and total order, there is always a unique next point in time. This allows us to write $\rhd t$.

$U \models^{\mathsf{s}}_{v} S' \subset S \triangleq U \models^{\mathsf{s}}_{v} S,\ U \models^{\mathsf{s}}_{v} S'$ and $S' \subset S$ – For **viewer** v viewing **universe** U, **system** S' is a **sub-system** of S

$U \models^{\mathsf{s}}_{v} S \triangleq \exists_{C,E} [U \models^{\mathsf{s}}_{v} \langle C : E : S \rangle]$ – A relationship expressing which **system** is viewed by which **viewer**. The fact that a **viewer** v views system S in **universe** U is expressed as $U \models^{\mathsf{s}}_{v} S$.

$\_ \models^{\mathsf{s}}_{\_} \langle \_ : \_ : \_ \rangle \subseteq \mathcal{UN} \times \mathcal{VW} \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL}) \times \wp(\mathcal{EL})$ – A relationship expressing which **system** and **environment** are viewed in the **universe** by a **viewer**. The fact that a **viewer** v with **conception** C views **system** M with **environment** E for a part of **universe** U is expressed as $U \models^{\mathsf{s}}_{v} \langle C : S : E \rangle$.

$\mathcal{TI}$ – Points of time.

$\mathsf{To} : \mathcal{LI} \to \mathcal{CO}$ – The destination **concept** of a **link**.

$\mathcal{UN}$ – The set of **universes**.

$\mathcal{VW}$ – The set of **viewers**.

Dictionary

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

Activity participation – A **system link** between a **system activity** and one of its **actor**.

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**.

Alignment – A **system** S_1 is aligned to a system S_2 if, and only if, system S_1 meets *all* **requirements** posed by S_2 on S_1.

This alignment may be strived for at different levels of refinement of a system design.

Architectural description – A **system description** documenting/describing an **architecture**.

Architecture – A **model** of which the **system description**, the so-called **architectural description**, is used during **system engineering** to:

- express the fundamental organization 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) **organizations** can be regarded as **autonomous systems**.

Communication – An exchange of **messages**, i.e. a sequence of mutual and alternating message transfers between at least two **human actors**, called communication partners, whereby these **messages** represent some **knowledge** and are expressed in languages understood by all communication partners, and whereby some amount of **knowledge** about the domain of communication and about the action context and the goal of the communication is made present in all communication partners.

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.

Component – An abbreviation of: **component system**

Computerized information system – A **sub-system** of an **information system**, whereby all activities within that sub-system are performed by one or several computer(s).

Conception evolution – The evolution of a **conception**.

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 **links**.

Concern – An **interest** of a **stakeholder**, resulting from the **stakeholder goals**, and the role played by some **system**. 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.

Data – Any representation in some language. Data is therefore simply a collection of symbols that may, or may not, have some meaning to some **actor**.

Decomposer – The **link** between a composed **concept** and one of its underlying **concepts**.

Definition process – A process aiming to identify all requirements that should be met by the **system** and the **system description**.

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 – 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 process – The combination of the **definition process** and **design process**.

Description – The result of a **viewer** denoting a **conception**, 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 an information system. These descriptions should identify *how* an information 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 evolution – The evolution of a **domain** over time.

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 evolution – The evolution over time of an **element** in the **conception** of a **viewer**.

Element version – The version of an **element evolution** as it holds at some point in time. This version is an **element** from a **viewer**'s **conception** of a **universe**.

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

Environment evolution – The evolution of an **environment** over time.

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**.

Goal – An abbreviation of: **stakeholder goal**

Human actors – An **actor** which is a single human being, or essentially a set of human-beings, such as a team.

Information intensive organization – An **organization** conceived of as an **organizational system**, where the **information system sub-system** forms an essential part of the **organizational system**.

Information system engineering – A system engineering process pertaining to the creation or change of **information systems**.

Information system – A **sub-system** of an **organizational system**, comprising the conception of how the communication and information-oriented aspects of an **organization** 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 **organizational system**.

Information technology – To be defined.

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**).

Knowledge – A relatively stable, and usually mostly consistent, set of **conceptions** posessed by a single (possibly composed) **actor**.

In more popular terms: "an actor's picture of the world".

Link – Any **element** from a **conception** that relates two **concepts**.

Maintenance – An information 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 information system operational as is.

Message – **Data** that is transmitted from one **actor** (the sender) to another **actor** (the receiver).

A message may actually be 'routed' via several **actors** before reaching its actual receiver. For example, when **human actor** exchange messages, they usually need to make use of some other **actor** playing the role of a medium (for example, vibrations in the air, or an e-mail system).

Model concept – A **concept** from a **conception** which is a **model**.

Model link – A **link** from a **conception** which is a **model**.

Modeling technique – The combination of a **way of modeling** and a **way of communicating**.

Modeling – 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 semantical 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**.

Organizational system – A special kind of **system**, being normally active and open, and comprising the conception of how an **organization** 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.

Organization – 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 **organization** in itself is an **actor** as well, and may as such participate in yet another **organizations**.

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.

Quality attribute – A specific class of **quality properties**.

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

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

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

Requirement – an essential **quality property** that a **system** or its **system description** has to satisfy.

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 concern – An **interest** of a **stakeholder**, resulting from the stakeholder's **goal**, and the role played by some **system**.

This usually pertains to the system's engineering, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

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 **system engineering** costs, while the goal of users of the system may be to get their job done more efficiently.

Stakeholder requirement – A **requirement** posed on a future/changed **system** by a specific **stakeholder**.

These requirements should essentially be refinements of the **stakeholder concerns**.

Stakeholder – A party (a **system viewer**) with a specific **interest** 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 activity – A **system concept** that is conceived of as changing parts of the **universe**.

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

System description – The **description** of a **system**.

System design – To do

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 aimed at producing a changed system, involving the execution of four sub-processes: definition, design, construction and installation. 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 **interest** 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 mission – A role, to the benefit of **stakeholder goals**, for which the **system** is intended.

System requirement – A **requirement** on a future/changed **system**. These requirements are usually an negotiated integration of **stakeholder requirements** and should essentially be refinements of the **system mission**.

System type – A type that determines the potential kinds of **systemic properties**, **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 properties**) 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**.

Way of communicating – describes how the abstract concepts from the **way of modeling** 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 modeling and the **way of working**, it matches the abstract concepts of the way of modeling to the pragmatic needs of the way of working.

Note that it may very well be the case that different **modeling techniques** are based on the same **way of modeling**, yet use different notations.

Way of controlling – The managerial aspects of system development. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management and governance (see [Ken84, Sol88]).

Way of modeling – Identifies the *core concepts* of the language that may be used to denote, analyse, visualise and/or animate **system descriptions**.

Way of thinking – Articulates the assumptions on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* [Sol83, WAA85], *underlying perspective* [Mat81] or *philosophy* [Avi95].

Way of working – Structures (parts of) the way in which a system is developed. 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.

Author Index

Subject Index

The following conventions are used in this index:

- A page where a concept is defined: *203*.
- A page where a concept is discussed or mentioned: 203.
- The page in the dictionary where a concept is defined: **203**.

The DAVINCI Lecture Notes Series:

The DAVINCI series of lecture notes is concerned with *The Art & Craft of Information Systems Engineering*. On the one hand, this series of lecture notes takes a fundamental view (*craft*) on the field information systems engineering. At the same time, it does so with an open eye to practical experiences (the *art*) gained from information system engineering in industry.

Main contributors:

P. (Patrick) van Bommel

S.J.B.A. (Stijn) Hoppenbrouwers

H.A. (Erik) Proper

Th.P. (Theo) van der Weide