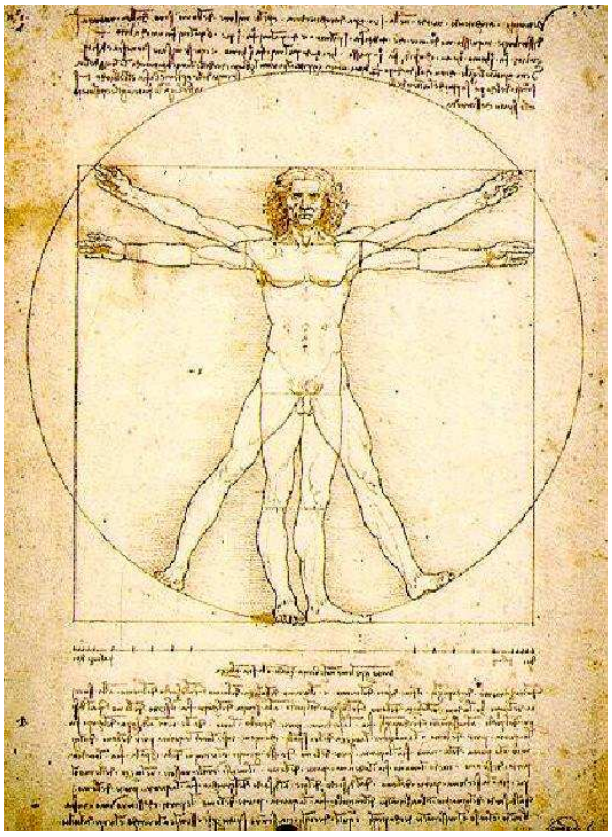
Modeling and Reasoning

The systematic construction of an ontology
Lecture Notes

Th.P. van der Weide

Version of: 01-12-2007



The Art & Science of Enterprise Engineering

Modeling and Reasoning

The systematic construction of an ontology
Lecture Notes

Th.P. van der Weide

Contents

The DAVINCI Series

Version: 30-08-05

The subtitle of the DAVINCI series of lecture notes is *The Art & Science of Enterprise 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 [**?**], we will (throughout the DAVINCI series) use the following definitions:

[data-us] –

[information-us] –

[knowledge-us] –

[communication-us] –

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-actor-lp]**), storage, processing, retrieval, and presentation (to **[human-actor-lp]**) 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 four key key processes in an information system's life-cycle:

[definition-process-us] –

[design-process-us] –

[realization-process-us] –

[architecting-us] –

[domain-modeling-us] –

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. Note that we regard maintenance of systems as being functionality that should be designed "into" the system. If a system needs to be maintained, and in most cases one indeed wants to, then the maintenance should be designed into the workings of this system and/or its context.

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

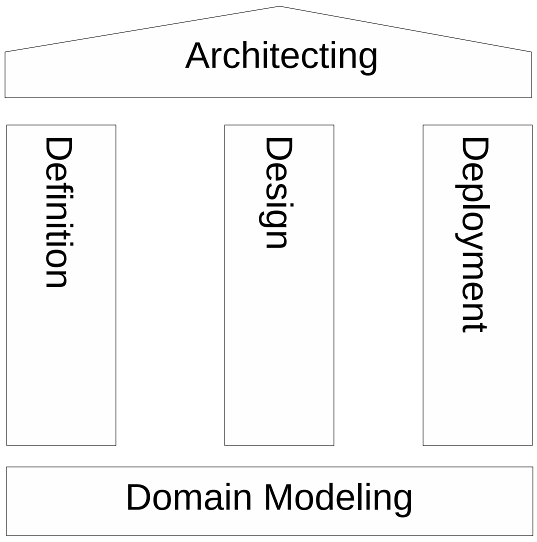


Figure 1: Aspects of Information Systems Engineering

and a theoretical perspective on information systems and their evolution [?]. 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 honor 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 [?] 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 [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?], information retrieval [?, ?, ?, ?, ?, ?, ?, ?, ?], (enterprise) information architecture [?] and information system engineering [?, ?] into the core of the DA VINCI series.

Course Description

Version: 01-02-07

Short description

This course covers an important prerequisite in the information system life-cycle where the intention is to obtain an (agreed) understanding of a given domain. In this course, students will be taught to demarcate a domain and identify its ontology, i.e. a specification of its conceptualization comprising the core concepts in the domain, their mutual relations and the laws (constraints) governing their behavior. Students will get a deep understanding of conceptual modeling, and will be able to transform a model into a computational environment in general, and a relational database structure and SQL in particular. This will be related to the Unified Modeling Language UML.

Objectives

After this course the student

1. understands what a model is.
2. understands the ORM normalform for natural language sentences, and can transform a sentence into this normalform.
3. can systematically derive a conceptual model from a structured domain description.
4. understands how the conceptual language ORC is derived, and can express conceptual operations in terms of this language.
5. has a base understanding of formal reasoning in ORC.
6. can transform a conceptual model and its conceptual operations into a relational database structure and SQL.
7. understands how the action-oriented approach generalizes the fact-oriented approach.
8. has a basic understanding of UML, and knows how to apply conceptual modeling during UML.

Topics

The course is organized in 2 blocks:

1. Modeling and Reasoning
   - Domain modeling is introduced as an activity that tries to describe how people communicate in some universe of discourse. The students learn how a controlled language approach helps to make an initial structuring.
   - Sentences from the controlled language are analyzed for concepts, leading to an overall description of concepts and their relations. This is called the conceptual schema, also referred to as a domain ontology.
   - The conceptual schema is seen as an information grammar. It is discussed how this grammar is the base for a conceptual domain language: Lisa-D.

- Students learn how to formulate domain properties and conceptual information system operations in terms of Lisa-D. It is also discussed how Lisa-D can be used to formally prove properties of the underlying application domain.
- As an information system creates a shadow world of the universe of discourse, special attention is paid to the required relation between the formal and informal world.

2. Transforming and Implementing
   - In this block the students will learn the principles of SQL as a general foundation to build information systems.
   - The students will learn syntax and get a clear impression of the semantics.
   - Students learn how the conceptual schema can be transformed into a relational model in SQL. They also learn how, in terms of this transformation, the conceptual information system operations (Lisa-D) can be transformed into SQL-statements.
   - They will be trained in constructing SQL queries in non-trivial cases. The technique of working with refinements is a basic tool, that also gives a clue to reason about the correctness of a query.
   - The students get concrete hands on experience from an concrete SQL system.
   - In this block we first discuss the Unified Modeling Language UML, and show how conceptual modeling explains submodels of UML formally and also gives a concrete assist for the construction of these submodels.
   - In conceptual modeling, we have used a fact-oriented approach. In this block we will discuss how this can be generalized into an action-oriented approach.
   - Especially we will discuss the nature and construction of the Object-Life model.

Outline

Version: 01-02-07

Modeling and Reasoning

A1 ORM Normal form and translation to conceptual schema

A2 Context, motivation and historical aspects

A3 ORM Calculus: the formal language
Support: Multisets

A4 ORM Calculus: the reasoning model

A5 Schema relations, frequently occuring constraints

A6 Advanced modeling concepts

A7 The (formal) modeling algorithm

Transforming and Implementing

B1 SQL (simple queries)

B2 SQL (more complex queries; DDL)

B3 Data transformatie

B4 SQL (Data integrity, DDL, DCL)

B5 UML (static aspects)

B6 UML (dynamic aspects)

B7 Varia
- Storing Hierarchical Data in a Database
- Relational Normalization

Introduction to Domain Modeling

Version: 17-07-05

"In theory is there is no difference between theory and practise. In practise there is"

(Jan LA van de Snepscheut)

Nowadays many methods exist for the development process of software ([**?**]). A number of examples are:

- *Iterative development* is a well known example on which the *the waterfall model* ([**?**]) is based. The waterfall model is a clear model for managing the development proces because it separates this process into several well defined phases. The Rational Unified Process (RUP) is an example of an iterative development method ([**?**]).
- *Evolutionary development* is based on the idea of developing an initial implementation (a prototype) and then refining and completing this prototype in interaction with the user. This way of working allows requirements and design decisions to be delayed.
- *Incremental development* as suggested in [**?**] combines the advantages of both iterative and evolutionary development leading to a clear management model with more user interaction. Another example of a hybrid, iterative model is the *spiral model* ([**?**]) which combines incremental development with explicit risk analysis thus improving the overall rate of success of the development process.
- *Reuse-oriented development* focusses on reusing existing software components. COTS (Commercial Off-The-Shelf systems, [**?**]) is an example of industrial practise.
- *Formal systems development* is based on formal mathematical transformation. The best known example is the Cleanroom process which was originally developed by IBM ([**?**, **?**]).

As different as all these development processes may be, there are fundamental activities common to all these processes. One of these activities is *requirements engineering* (RE), although this activity has its own rules in each development method. RE is the process of discovering the purpose for which the software system is meant, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, negotiation, decision-making (see [**?**]) and subsequent implementation. For an extensive overview of the field of RE we refer to [**?**].

Experts in the area of software engineering do agree that RE is the most import factor for the success of the ultimate solution for reasons that this phase closes the gap between the concrete and abstract way of viewing at phenomena in application domains (see [**?**], [**?**] and [**?**]). As a consequence, during the RE process, the involved information objects from the universe of discourse (UoD) have to be identified and described formally. We will refer to this process as *Information Modeling (IM).* The resulting model will serve as the common base for understanding and communication, while engineering the requirements.

Augustus 2005,

Th. P. van der Weide

Chapter 1

Conceptual Data Modeling

Version: 17-11-07

(Siep Weiland, Model Approximation of Dynamical Systems)*The general modeling problem invariably involves a trade-off between complexity and accuracy of models. Simple models are preferred above complex models and accurate models are evidently preferred above models which have poor descriptive or predictive power. To obtain high accuracy models, one usually needs to resort to complex models, low complexity models are generally inaccurate.*

1.1 What is a model?

According to the Merriam-Webster Online dictionary[1] the word *model* is a noun, (assumed) to originate from the Latin word *modellus*. The word can have the following meanings:

1. (obsolete) a set of plans for a building
2. (dialect British) COPY, IMAGE
3. structural design ⟨a home on the model of an old farmhouse⟩
4. a usually miniature representation of something; also: a pattern of something to be made
5. an example for imitation or emulation
6. a person or thing that serves as a pattern for an artist; especially : one who poses for an artist
7. archetype
8. an organism whose appearance a mimic imitates
9. one who is employed to display clothes or other merchandise (mannequin)
10. a type or design of clothing b : a type or design of product (as a car)
11. a description or analogy used to help visualize something (as an atom) that cannot be directly observed
12. a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs
13. version

model

In this course we will see a model as *system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs*, to be used as *an example for imitation or emulation*.

1.1.1 Model variety

universe of discourse

The central issue in this lecture is: A model tries to grasp (describe) the *essence* of some part of the world around us. This part is referred to as the *universe of discourse* (UoD). This leads us to the question: what is

[1] See http://www.m-w.com/dictionary.htm

the essence? From practise it is known that different points of view may lead to rather different descriptions of the same phenomenon. For example, different models of the universe have been developed:

- Aristoteles: earth and heaven
- Ptolemeus: heavenly bodies in orbit around the earth
- Copernicus: earth turning around the sun
- Kepler: ellipses

How can these different views all be correct models of the universe? For this question to answer, we consider the reasons for making a model. We see a model as a framework to communicate about the part of the world (to be) observed (the UoD). As a consequence, with a model is associated a criterion that determines what kind of statements are valid statements. The set of all valid statements is our first abstraction of the actual world being observed. A model then contains a description of this set of sentences.

1.1.2 Model validity

A model is presumed to be valid as long as it has not been contradicted by observation (the principle of *falsification*). This principle and its limitations has been described by Popper ([**?**]).

falsification

For example, we could see a simple calculator as a model for human computing power. In fact several models have been developed for this purpose, the most famous one being the Turing Machine. Until now, no person has been able to perform a computation that could not be handled by a Turing Machine. So a Turing Machine is still seen as a valid model of human computing. The confidence in this statement is very large, there is probably no person that has any doubt about its validity. This confidence is known as the Church-Turing Thesis: *any real-world computation can be translated into an equivalent computation involving a Turing machine*.

Note that we may not have hope to do more than falsification. For a correctness proof, we would need a description of the universe of discourse formal enough to be able to make such a proof. After having presented this proof, we still would have the problem to show that this formal enough description of the universe of discourse in fact is a correct description, bringing us back to our starting point.

1.1.3 Using a model

A model is to be used as an example for imitation or emulation. Or, in more general terms, we model because modeling answers questions (see [**?**]). While this is too generic an answer to solve much, it does directly clarify our approach to conceptual modeling. By asking: *Who asks the questions that need to be answered?* and *Why these people ask those questions?*, we immediately arrive at a view on conceptual modeling that is deeply rooted in communication, involving language as a means to achieve communication.

The questions may reveal the following reasons for making a model:

1. To make certain behavioral predictions, for example strength computation for materials to be used in a construction process. Such predictions usually have a quantitative nature.
2. Derivation of properties. Properties concern qualitative aspect os the universe of discourse.
3. To make an emulation (also referred to as a simulation).

simulation

Simulation affords the opportunity to visit a world that would otherwise be impossible, difficult or impractical. For example, simulation may let us explore another planet, or inspect the implications of how we plan the arrangements in a new to build house. A typical example of simulation is a graphical computer game.

Simulations can be used to highlight relevant aspects of a model, and to let the user try virtual manipulations to see their effect. It is even possible to highlight *invisible* aspects. For example, an offside judgement in a football game may be evaluated by manipulating the recordings of different cameras to obtain (calculate) the required point of view.

1.1.4 Observation and representation

A model is a description of the universe of discourse. In order to obtain a model, observations of the universe of discourse are analyzed. Both observations and analysis will be made with a more or less specific *goal* in mind. These goals determine the required properties of the model: its form, the type of information it contains, precision, etc. Observation and analysis are separated activities of the modeling process. Even if these modeling roles are played by a single person, it will make sense to make an explicit formulation of the observations made. The observer of the universe of discourse is referred to as a viewer. A viewer can be seen as what described above as a criterion that determines what kind of statements are valid statements in the UoD. The viewer perceives the universe of discourse, and constructs (conceives) a mental model of that domain. Therefore, the information provided by a viewer is always subjective and reflects only the structure of the domain *as seen by the domain expert* [**?**]. Strictly speaking, a model is a representation of the *mental model* of that viewer, rather than a representation of the domain itself.

viewer

The viewer perceives the universe of discourse, leading to a mental model consisting of conceptions. In order to communicate these conceptions, some description mechanism has to be used. Symbols are used in descriptions to represent (elements of) conceptions. Those symbols have a meaning which is valid only in that context.

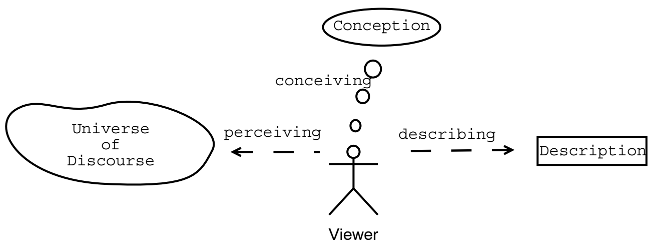


Figure 1.1: Observing the universe of discourse

One should be careful to avoid misconceptions. According to I. Kant (Critique of Pure Reason, 1789)[2]:

> *Perception without conception is blind;*
> *conception without perception is empty.*

For example, pictures and tables are an efficient representation mechanism. The danger is that, while revealing one point of view, (1) they hide underlying assumptions, (2) eliminate other possibilities and (3) prevent comparisons without silently and unobviously.

A viewer with the intention to describe a universe of discourse is called an informant. Different informants will produce different perspectives on the same domain. Even more, a single informant may still produce different perspectives on the same domain when modeling from a different interest. For example, a financial versus a organizational description will each emphasize their own point of view. Due to the particular role of the informant during modeling, the informant will later also be referred to as the domain expert. We will assume the modeling process as an activity between a domain experts and a system analyst.

informant

1.1.5 Model simplicity

Models may not give a complete description of a phenomenon. A model may be restricted to some specific characteristics of the phenomenon, and therefore can only be used in the context of those characteristics. The model may be convenient, while another model is used for more complex situations that the simpler model is not capable to handle. As a rule of practise, a simpler model is preferred above a complexer model if both are sufficiently expressive. This principle is known as *Ockham's Razor*: *The simplest explanation is usually the best.*

ockhams-razor

The essence of the universe of discourse is obtained by leaving out unessential matters. This process of omitting details is called *abstraction*. As a consequence, abstraction leads to a simplification of the observed

abstraction

[2]See http://www.bartleby.com/60/144.html

reality. This simplification depends of the chosen angle of incidence and the goal intended to achieve during modeling. The consequence is that a modeler (system analyst) is capable to handle abstraction and ordering conveniently.

Note that making models is a common human activity in daily life. Humans seem to be extremely abstraction-driven, and can learn even from a small set of examples.

Example 1.1.1 *The construction industry*
A construction drawing is a model of a building as used in the construction industry, for example:

1. construction of a dogs house
2. construction of a building
3. construction of an office building complex

Such a model is constructed by an architect. Making a model is recognized as a creative process. The model is used: *architect*

1. to make calculations, for example to estimate in what quantity a material will be required.
2. to derive properties, for example about stability reliability.
3. to make a simulation, for example a virtual journey within the future building.

1.1.6 Types of models

As argued before, a model is set up with the intention to answer some kind of questions. Some examples are:

1. scientific: a model describes phenomena from reality, the model should be capable to answer quantitative questions.
2. astronomy: the model should provide a description the population of heavenly bodies, their properties, their relations and their history.
3. linguistics: to describe the nature of communication, and its variation over time. A typical model is a grammar, which describes how sentences are formed using some lexicon, the meaning of sentences, and the pragmatics of their usage.
4. administration: to construct a shadow world. This shadow world should be powerful enough to answer all kinds of questions about the state of affairs in the shadowed world, both involving the phenomena it contains and the concepts that structure them. *shadow world*

We will be concerned with the administration type of models. Our goal is to enable the procedures that keep the real world and the shadow world in check with each other. The shadow world then is used as a simulation of the real world. The advantage is that the shadow world is more easily handled.

1.1.7 Model as ontology

In this course we will focus on the administration variant. In this variant modeling may also be seen as an activity to build an *ontology*. According to J.F. Sowa [**?**]: *ontology*

> *The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D.*

We will use the terminology from data modeling techniques and use the term information grammar rather than ontology. Ontologies are not only used in the context of information systems. Other application areas are: e-Science (such as Bioinformatics), Medicine (terminologies), Information Retrieval, User Interfaces, Linguistics and the Semantic Web ([**?**]).

In Computer Science and Information Science ontologies are used to capture a shared understanding of some domain of interest, in order to provide a formal and machine manipulable model of that domain.

1.2 Conceptual modeling

The intention of administrative modeling is to create a shadow world that can keep track of the wheeling and reeling within the universe of discourse. The model provides an understanding of the observed world, which makes it a tool of great value for management purposes. The model may also be used to build a simulated world. An information system typically realizes a simulated world, and is very helpful as a tool to obtain knowledge about the universe of discourse without the need to do a costly search in the real world.

The approach taken is to describe the conceptual model from the perspective of the language being used for the purpose of talking about the universe of discourse. The conceptual model thus may be seen as the grammar describing that language. A grammar consists of a lexicon and the set of rules by which sentences from that language can be constructed.

Another way to put it: a model will describe the essential concepts within the universe of discourse, and how they are related. The term *conceptual modeling* is used to stress this aspect of modeling.

conceptual modeling

1.3 Communication via sentences

For communication to be successful, the participants of the communication should agree on a number of things, such as:

1. the way *how* messages are exchanged. For example, the participants have agreed that messages are exchanged by speech, and agree on communication rules. For example, they agree that only one participant is speaking at each moment and the others are listening.

2. the way messages are constructed. In this course we assume messages to consist of sentences. Furthermore, these sentences have a structure that is known to the participants. For example, the participants have agreed to adopt the (universal) rules of the language Dutch to express their domain specific sentences.

3. the meaning of sentences. When a member of the UoD is trying to ask a favor from another member of the UoD, then both should agree to a sufficiently large extent on the actual meaning of the language they use.

During modeling, however, a rather limited understanding of the full semantics is not necessary. The structure of the sentences is the essential element to model. The underlying assumption is: *the sentence structure is a sufficiently rich base for meaning*.

Example 1.3.1 *data communication*
In a technical context, this agreement is referred to as a *protocol*. An example of a protocol is TCP/IP, based on the OSI layers[3]:

- *The Physical Layer defines the electrical and mechanical aspects of interfacing to a physical medium for transmitting data, as well as setup, maintenance, and disconnection of physical links. This layer includes the software driver for each communications device, plus the hardware itselfinterface devices, modems, and communications lines.*

[3]Cited from `http://www.blackbox.com/tech_docs/tech_overviews/osi.html`

- *The Data-Link Layer establishes an error-free communication path between network nodes over the physical channel, frames messages for transmission, checks the integrity of received messages, manages access to and use of the channel, and ensures the sequence of transmitted data.*
- *The Network-Control Layer addresses messages, sets up the path between communicating nodes, routes messages across intervening nodes to their destinations, and controls the flow of messages between nodes.*
- *The Transport Layer provides end-to-end control of a communication session once the path has been established, which enables the reliable and sequential exchange of data independent of both the systems that are communicating and their locations in the network.*
- *The Session Layer establishes and controls system-dependent aspects of communication sessions provided by the Transport Layer and the logical functions running under the OS in a participating node.*
- *The Presentation Layer translates and converts transmitted encoded data into formats that can be understood and manipulated by users.*
- *The Application Layer supports user and application tasks and overall system management, including resource sharing, file transfers, remote file servers, and database and network management.*

People communicate in a large variety of ways. The communication between people can be distinguished into:

informal and formal communication

1. *informal communication*: such as body language
2. *formal communication*: such as written text

It has been shown that in practise informal communication is far more important than formal communication. For example, a facial expression may have a decisive influence on the meaning of a sentence. Note that smileys or *emoticons* are an agreed formalism to add emotional value to written text `(-_-)`.

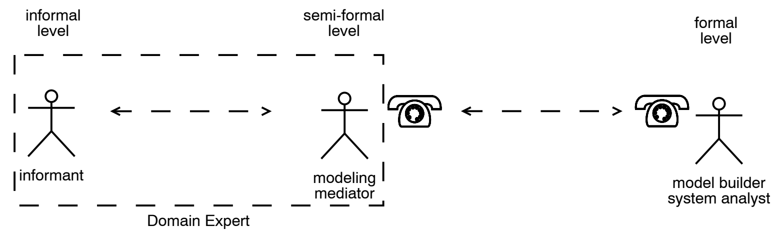


Figure 1.2: The telephone heuristic

For modeling purposes, as such circumstantial influences can only be useful when they adhere to rules, conventions and ceremony, we restrict ourselves to communication formats that are sufficiently stable. As a result, we will concentrate on formal communication. This principle has been formulated as the *telephone heuristic*: the only information carried over is the spoken text. This makes the information transfer independent of the actual interaction.

telephone heuristic

The problem that is encountered next is the huge richness that natural language offers to its users, both on structure as at the level of the lexicon. This richness can only be handled by the introduction of a normal form for sentences. Such a restricted language is an example of a *controlled language*. The transformation of sentences into this normal form requires a good understanding of the meaning of the sentences. As a consequence, this is typically done by a domain expert.

controlled language

Next we focus on the sentence itself. A sentence can do one of the following

fact oriented

1. describe a fact
   If we interpret sentences as describing facts, then the ordering of sentences is of minor importance.

2. describe an action *action oriented*
   In this case the order of sentences is meaningful.

In this lecture we will consider sentences as facts.

Later we will introduce a strict format (normalform) for sentences, which makes it easier to derive the essential parts of a sentence. We assume that in this normalform the structure of sentences is such that they can be (easily) decomposed into the following components: *normalform*

1. the kind of fact/action which determines the type of the sentence
2. the components and how they contribute to the fact/action.

ORNF

The resulting normalform will be called the ORM Normalform, abbreviated as ORNF. In the fact oriented approach, we can say that a sentence is categorized by its predicate and determined by the involved objects and their associated roles. This choice is the rationale for choosing the name *object-role modeling*. We will use the mechanism of *index expressions* to represent this structure. An index expression consists of a header term, followed by a number of modifiers. *index expression*

Example 1.3.2

The sentence

person Smith visits country Italy

will be decomposed into the following index expression:

visit **agens** (person **being** Smith) **patiens** (country **being** Italy)

agens patiens

The header of the index expression represents the predicate of the sentence category. The subtrees represent the various agents involved in this predicate. The labels mark their roles. Special roles are **agens** (most commonly associated with the grammatical subject) and **patiens** (most commonly associated with the grammatical object). Other objects will have a particle that clarifies their role.

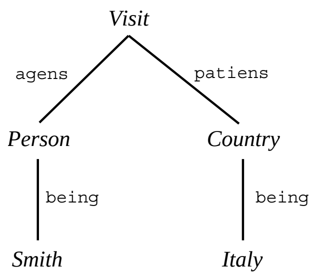
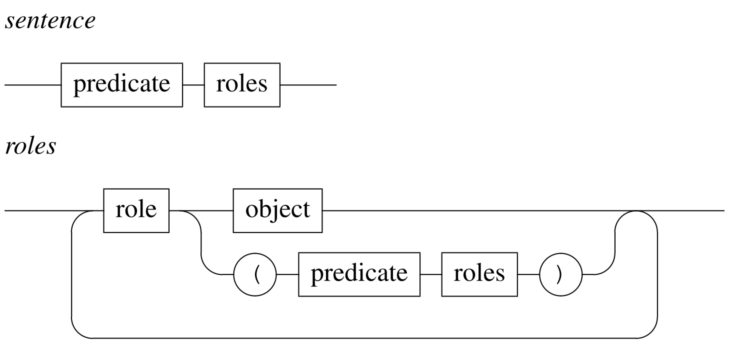


Figure 1.3: The index expression

This example demonstrates the structure of index expressions. We will not need further details of index expressions. For the sake of completeness, a grammatical description of index expressions is given:



1.4 Data, information and reality

The information system provides a shadow world for the universe of discourse. In the concrete world there are concrete objects, also referred to as a *thing*. In the abstract world of the information system for each thing there is an (abstract) object as counterplayer. A major point of concern will be to guarantee a 1-1 correspondence between things and abstract objects. Besides, there should be a mechanism for uniquely describing things and abstract objects in the communication between universe of discourse and information system. A straightforward method would be to have a unique name for each thing in the universe of discourse. A name then will uniquely refer to a single thing in the UoD, while this same name is also capable to identify the unique corresponding object in the formal world. However, this unique name requirement is not realistic. For example, the name Fritz may address both a person and a cat in some universe of discourse.

In practise things are grouped around some common properties. For example, cats are a group of animals whose members are obviously dissimilar to those of the group of things denoted as dogs. Groups of related things in the universe of discourse will lead to a type, i.e. a group of related objects. For example, in the UoD of figure 1.4 a group of similar things is demonstrated, which leads to the type *Person* in the model.

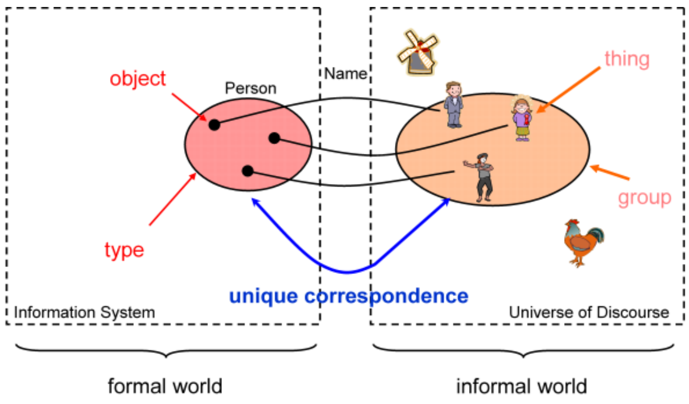


Figure 1.4: Relating the system and the universe of discourse

Remark 1.4.1

In our approach, a concrete thing from the universe of discourse is represented in the information system as an abstract object. This is done by storing the (relevant) facts of things. A fact in which an object participates is a property of that object. The representation of a thing thus is the set of properties of the associated object.

weak identification

A sufficient condition to be able to uniquely identify objects and things, it that no two objects or things have the same properties. This is called the *Principle of Weak Identification*.

standard name

A usual way to resolve ambiguous names is to provide the associated group as a context. In our example we then get the *cat Fritz* and a *person Fritz*. The resulting correspondence is displayed in figure 1.5. In the context of modeling, this unique reference is called the *standard name* of both the thing and the object. For the moment we restrict ourselves to this simple format for standard names. In a later section we will see that compound standard names are also possible when complex identifications are required.

Remark 1.4.2

Note that the introduction of a standard name for an object type guarantees that all objects of that type have at least one different property: their standard name. This property is referred to as *Strong Identification*.

strong identification

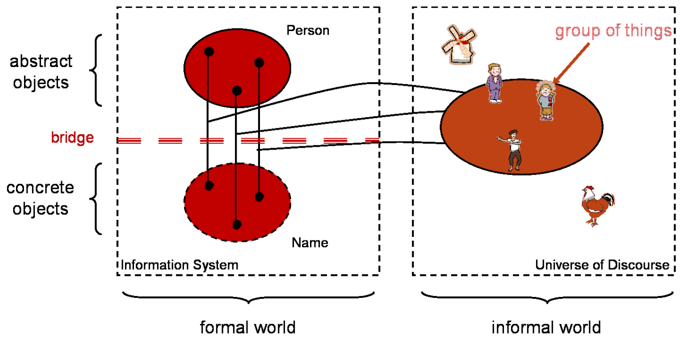


Figure 1.5: A more detailed view

1.5 Removal of syntactic variation

The transformation of natural language sentences into ORNF is done in several steps. First we describe the convention to eliminate the syntactic variation which is so common for natural language. Consider the following sentences in the context of our running example:

> In this company there are cooperators and departments. Cooperators have names, departments have department names. A cooperator can work for only one department. Janssen and Pietersen work for department Sales, and Klaassen works for the Personnel department. A department may have at most 10 cooperators.

Note that this description contains sentences at different levels of abstraction.

1.5.1 Splitting sentences

We may distinguish two kind of sentences:

1. non-elementary sentences. The information conveyed by such a sentences can be split in several sentences without changing the information content.
2. elementary sentences. Elementary sentences can not be split into smaller sentences without losing information. *elementary sentence*

For example, consider the sentence

> Janssen and Pietersen work for department Sales, and Klaassen works for the Personnel department.

The conjunction *and* is not functional in this sentence. Therefore, this sentence can be split without loss of information into the following ones:

1. Janssen works for department Sales.
2. Pietersen works for department Sales.
3. Klaassen works for the Personnel department.

These sentences are elementary as is easily verified.

Another example:

The married couple Els and Jan possess a car with license plate DC-12-10.

This sentence contains a conjunction, yet it can not be split without loss of information. The reason is that the married couple plays the role of car owner. The married couple is identified by the names of both spouses. Note that this sentence is equivalent with

The married couple Jan and Els possess a car with license plate DC-12-10.

We will come back to this in a later section.

1.5.2 Grouping by deep sentence structure

There are several ways to formulate the same message. For example:

- Janssen works for department Sales.
- department Sales provides work to Janssen.

deep sentence structure

The first sentence is an active voice formulation, the second sentence is in passive voice. The sentences have the same deep sentence structure. Sentences with the same deep structure are grouped together into the same sentence type. For example, the sentences above are grouped into the sentence type that is named *WorksFor*.

The associated (concrete) facts will have a corresponding (abstract) fact object in the information system. The corresponding object type will also be referred to as a fact type. The deep sentence structure describes the linguistic format.

1.5.3 Full qualification

The sentence Janssen works for department Sales contains the following references to concrete objects from the universe of discourse:

1. *Sales* is the unique name for some instance of ...
2. ... a meaningful kind of objects form the UoD. The name of this type of objects is *department*.
   We say that the name *Sales* is qualified by the type name *department*.
3. 'Janssen' is a unique reference to an instance of an unspecified type of objects.
   We say that this instance is *not* qualified.

fully qualified

A requirement for sentences to be well formed is that all instances are fully qualified. The rationale is that qualification is a step towards the forming of standard names. In order to meet this requirement, the sentence is reformulated as: Cooperator Janssen works for department Sales.

Still the format of the sentence is not sufficiently well formed. From the sentence Cooperator Janssen works for department Sales we can not really derive that *Janssen* is part of the sentence structure, or that it is used to label a particular instance of the UoD. In order to make this distinction, concrete names have a special notation: "Janssen". Concrete names are also referred to as *lexical objects*. They correspond to non-terminal symbols in the information grammar. Abstract object are also referred to as *nonlexical objects*, they depend on lexical objects for their representation.

lexical object

nonlexical object

Using this naming convention, the example sentence is reformulated as: Cooperator "Janssen" works for department "Sales".

1.5.4 Standard names

We are not yet satisfied with the reformulated sentence Cooperator ”Janssen” works for department ”Sales”. The problem is as follows. Cooperator ”Janssen” will probably correspond with a cooperator having *name* ”Janssen”. So the name is an attribute of a cooperator. There may be other attributes for cooperators, for example the mobile phone number of this person.

For a well formed sentence, it is required that abstract objects and concrete objects are clearly separated by adding attribute types. After reformulation, our sample sentence is transformed into: Cooperator with name ”Janssen” works for department with departmentname ”Sales”.

identifying attribute

A special property of the attribute *name* is that it is an identifying attribute. The expression Cooperator with name ”Janssen” is therefore called the standard name for objects of type cooperator. The standard name provides the unique correspondence between the formal and informal world.

Standard names are minimal. For example, the expression Cooperator with name ”Janssen” born in city ”Leeuwarden” implicitly expresses the combination name-birthplace is required for unique identification of cooperators in the underlying UoD. Would this be not the case, then a sentence containing this expression would be splittable.

1.5.5 The initial capital convention

A final modification of the sample sentence is the following naming convention:

- Type names are formed by a single word that starts with a capital letter
- all other letters are lower case

The result is the following sentence: Cooperator with Name ”Janssen” works for Department with Departmentname ”Sales”.

1.5.6 Object Role Normalform

After all these modifications, the sentence is well formed, and ready to be processed to be actually modelled. This normal form for sentences is called *Object Role Normalform*, abbreviated as ORNF. The dialogue between domain expert and system analyst should result in a significant set of sample sentences in Object Role Normalform.

Summarizing, a sentence is in Object Role Normalform if:

1. the sentence is not splittable without loss of information
2. each instance is fully qualified
3. only standard names are used
4. the proper naming convention is applied

In figure 1.6 we see how an ORNF sentence is represented as an index expression.

1.6 Processing ORNF sentences

Processing an ORNF sentence start with the parsing of that sentence. The most important element of a sentence is its predicate. The predicate has associated a number roles (subject, object, direct object, etc.) The predicate corresponds to the sentence type.

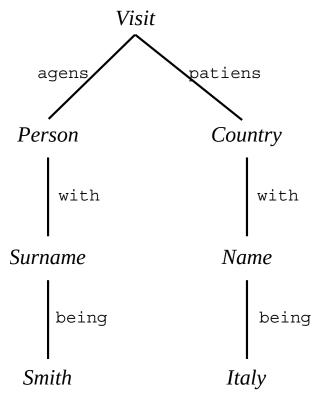


Figure 1.6: ORNF sentence as index expression

1.6.1 Binary fact types

We consider the sentence

Cooperator with Name "Janssen" works for Department with Departmentname "Sales"

A more global view on the index expression representation of this sentence (see figure 1.7) is the nested sentence structure as displayed in figure 1.8.

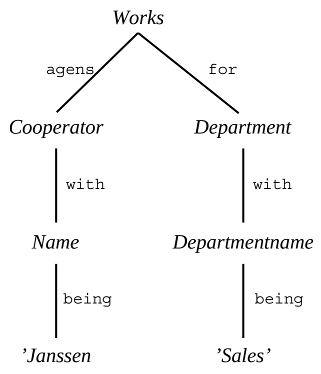


Figure 1.7: ORNF sentence as index expression

In the nested sentence structure, the node at the top level represents the whole sentence. The decomposition is as in figure 1.7. Directly underneath the root we see the representations of the agents involved. Next nested sentence structure shows from what instances they are composed, etc.

From the nested sentence structure (also referred to as parse tree) we generate figure 1.8. This figure has drawn circles around the object instances to denote the corresponding object type. The role of fact types are drawn by rectangles connected with the object type playing that role. The roles get associated their so-called role-name. This figure is called a *populated schema fragment*. Note the dotted line in this figure, that is used to demonstrate the division between abstract and concrete world. Fact types that bridge this gap between the abstract world (Person) with the concrete world (Name) are called *bridge types*. Bridge types form the only possibility to bridge this gap. Bridge types can only be binary fact types, i.e., fact types with in which 2 object types play a role. *bridge type*

The instances that are drawn in the populated schema fragment are called a population. They populate the schema fragment that is obtained by omitting the instances. The resulting schema fragment (see figure 1.10)

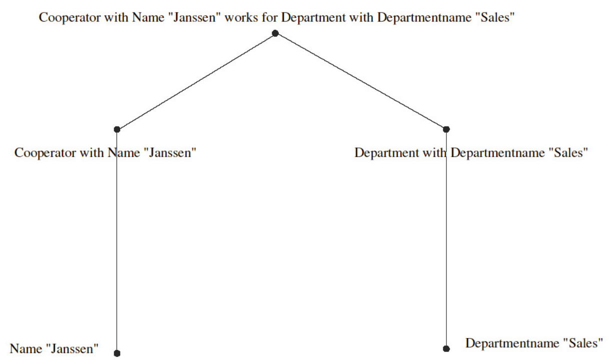

Cooperator with Name "Janssen" works for Department with Departmentname "Sales"
Cooperator with Name "Janssen"
Department with Departmentname "Sales"
Name "Janssen"
Departmentname "Sales"

Figure 1.8: Nested sentence structure of sample sentence

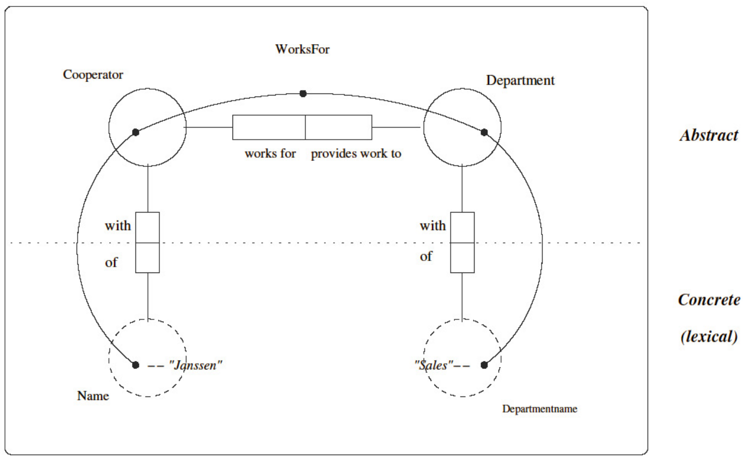


Figure 1.9: Sample sentence as populated schema fragment

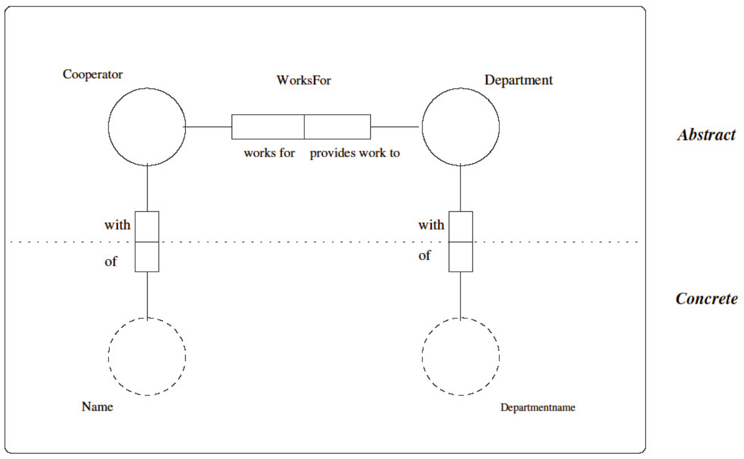


Figure 1.10: The resulting schema fragment

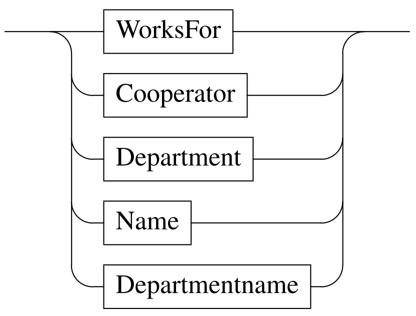
is a compact representation of the part of the information grammar that corresponds to the sentence structure from the sample sentence.

This schema fragment provides the syntactic categories and how they are related, abstracting form concrete verbalizations. The corresponding part of the information grammar:

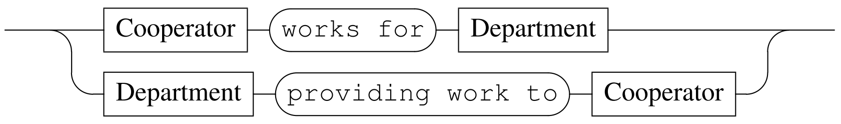
- lexical object types (label types): Name, Departmentname
- nonlexical object types: Cooperator, WorksFor, Department

The corresponding grammar rules describe the standard names for objects of these categories. This grammar is displayed in the following syntax diagrams:

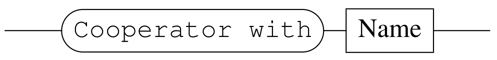
sentence



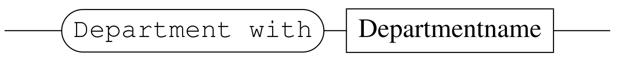
WorksFor



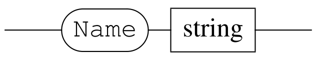
Cooperator



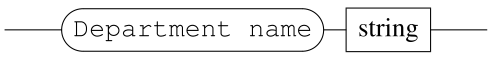
Department



Name



departmentname



Note that we have allowed both the active voice and the passive voice formulation of facts of type *WorksFor*.

1.6.2 Unary fact type

As a next example, we consider the fact

Person with Name ”Pete” smokes

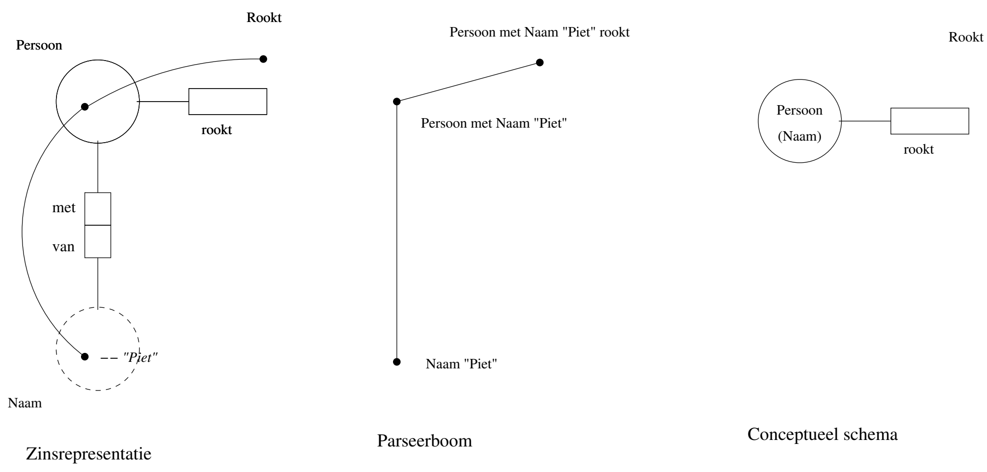


Figure 1.11: Abstracting from sample sentence

Analyzing this sentence, we see an object type *Person*, a label type *Name*, a sentence type with predicate *Smoke* and *Person* as its single participant. This is reflected in figure 1.11.

In figure 1.11 we see the parse tree of the sample sentence. The figure to the left of the parse tree has drawn circles around the object instances to denote the corresponding object type. The role of fact types are drawn by rectangles connected with the object type playing that role. The roles get associated their so-called role-name. This figure is called a *populated schema fragment*.

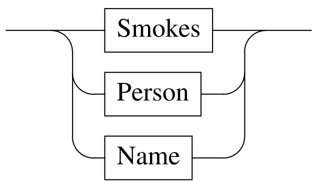
The figure to right of the parse tree is obtained form the populated schema fragment by omitting the population, and by using a shorthand notation for identifying bridge type. In this shorthand notation, we write the identifying label type directly below the name of the corresponding object type. This schema fragment is a compact representation of a fragment of the information grammar.

This schema fragment provides the syntactic categories and how they are related, abstracting form concrete verbalizations. The corresponding part of the information grammar:

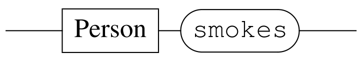
- lexical object types (label types): Name
- nonlexical object types: Person, Smokes

The corresponding grammar rules:

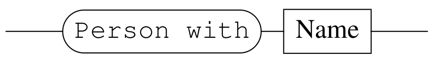
sentence



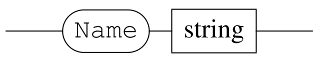
Smokes



Person



Name



1.6.3 Example

The following table is an example of how information is exchanged is some universe of discourse:

Room reservation organization:

Room	Time	Activity
20	Mon 9.00h	VMC
20	Tue 14.00h	VMC
33	Mon 9.00h	AQD
33	Fri 17.00h	SP

A straightforward way to formulate the facts that are stored in this table is:

- *Room with Number 20 at Time with Dh "Mon 9.00h" is used for Activity with Code "VMC".*
- *Room with Number 20 at Time with Dh "Tue 14.00h" is used for Activity with Code "VMC".*
- *Room with Number 33 at Time with Dh "Mon 9.00h" is used for Activity with Code "AQD".*
- *Room with Number 33 at Time with Dh "Fri 17.00h" is used for Activity with Code "SP".*

where Dh is an abbreviation for day-hour. Note that the sentence format is decided upon by the domain expert, and to be followed by the system analyst. In a common sense interpretation, derived from the example table, we see that actors play these roles independent of each other. For example, there does not seem to be a dependence between the activity code and the room number (though this could very well be the case!). Would there be any dependency between a proper subset of the roles, then this dependency requires a special sentence type to express this dependency. Typical examples of such dependencies are:

1. functional dependency: if the value of one role implies the value of another rol uniquely *functional dependency*
2. multi-valued dependency: if the value combinations of some combinations of roles follow some relational pattern. *multi-valued dependency*

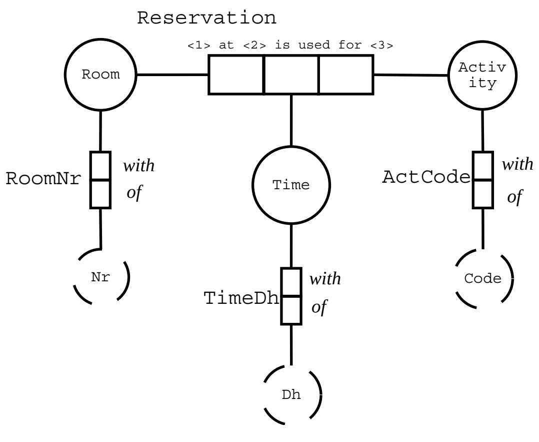
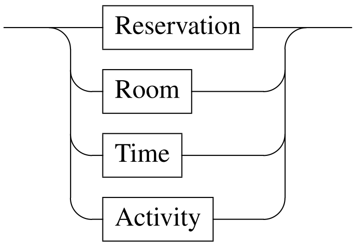


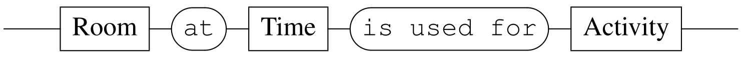
Figure 1.12: Conceptual schema for room reservation

As a consequence we have a fact type with 3 (independent) roles, a ternary fact type. The resulting schema is displayed in figure 1.12. The resulting grammar:

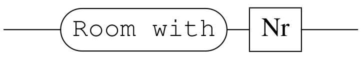
sentence



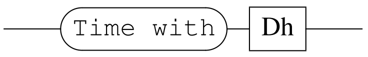
Reservation



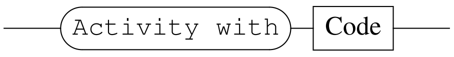
Room



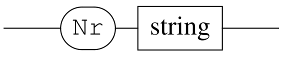
Time



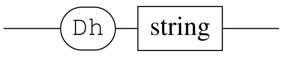
Activity



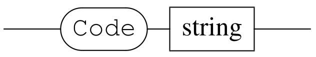
Nr



Dh



Code



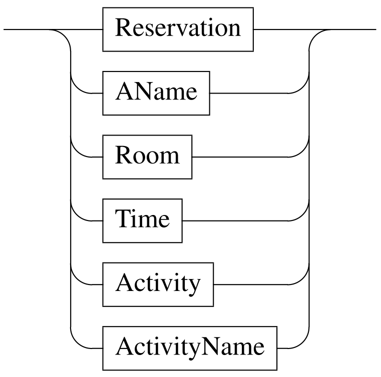
Next we extend this example with the following table:

Activity naming:

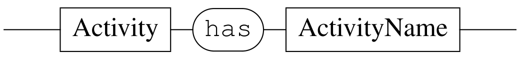
Activity	ActivityName
AQD	ActiveQuery demo
SP	Staff party
VMC	VisioModeler class
Y2K	Year 2000 seminar

The corresponding sentence format is: *Activity has ActivityName*. The resulting schema is shown in figure 2.19. The resulting grammar is:

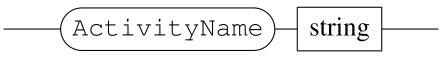
sentence



AName



ActivityName



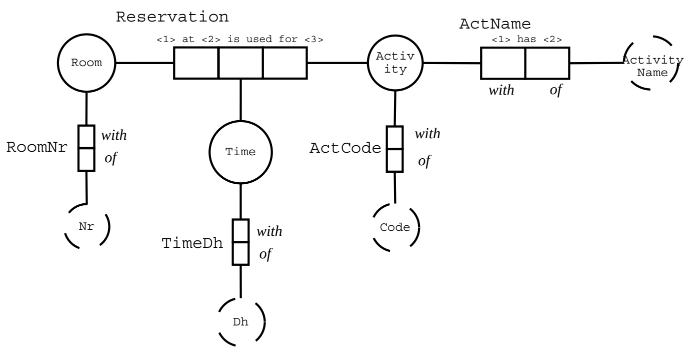


Figure 1.13: Conceptual schema for room reservation

.

1.7 Definitions overview

Model – A system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs, to be used as an example for imitation or emulation.

In the context of these lecture notes this is refined to:
A system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs, to be used as an example for imitation or emulation.

[uod-us] –

[application-domain-us] –

[falsification-us] –

[simulation-us] –

[viewer-us] –

[informant-us] –

[ockhams-razor-us] –

[ontology-us] –

[domain-expert-us] –

[architect-us] –

[system-analyst-us] –

[architect-us] –

[model-builder-us] –

[modeling-mediator-us] –

[elicitation-us] –

[abstraction-us] –

[controlled-language-us] –

[100-percent-principle-us] –

[conceptualization-principle-us] –

[turing-test-us] –

[shadow-world-us] –

[closed-world-assumption-us] –

[information-system-us] –

[information-grammar-us] –

[information-processor-us] –

[information-base-us] –

[niam-us] –

[psm-us] –

[ridl-us] –

[lisa-d-us] –

[sql-us] –

[uml-us] –

Questions

Version: 01-12-07

1. We willen nader kijken naar het verschil tussen model en werkelijkheid, en richten ons op het aspect: hoe goed kun je in het model over de werkelijkheid redeneren. Wat voor soort dingen kun je eruit afleiden, en voor wat voor soort dingen is het ongeschikt. Doe dit aan de hand van de volgende voorbeelden:

   (a) De bouwtekening van de bolderkar.

   (b) Een klassiek schilderij werkt met symbolen bijvoorbeeld:

   Aap : symbool van duivel, ketterij, ontucht en veinzerij. Ook het bewijs van welstand en rijkdom.

   Anker : symbool van de hoop (ook bijenkorf, spade en sikkel).

   Appel : symbool van verlokking en zonde, maar ook het attribuut van Venus en het symbool van liefde en vruchtbaarheid en, in afgeleide zin, van de huwelijksgemeenschap. Ook symbool van de heerschappij (rijksappel).

   Arend : symboliseert kracht en macht. Hij staat op de lezenaars in functie van boodschapper tussen de aarde en de hemel.

(c) Geef zelf een voorbeeld van een model, waarvan de bruikbaarheid hoog is, maar dat grote hiaten met de realiteit vertoont.

2. Splits de volgende zinnen zover mogelijk in kleinere zinnen, zonder dat de inhoud verloren gaat.

   (a) Student Jan volgt vak DM en Studente Ria volgt vak P1.
   (b) Studente Els heeft voor opgave 4 vak DM een 10 gehaald.
   (c) Student Jan had voor opgave 3 van vak DM een 9 en voor opgave 4 een 10.
   (d) Student Jan en studente Els doen het practicum van DM.
   (e) Student Jan en studente Els doen samen de casus van DM.
   (f) De student met studentnummer 12345 krijgt voor het vak DM het cijfer 8.
   (g) De student met studentnummer 12345 volgt het vak DM en behaalt daarvoor het cijfer 8.

3. In some UoD designers of programming languages form an important issue. The following table is an example of the way in which information is being exchanged in this UoD. Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

Designers:

Designer	Language
Wirth	Pascal
Kay	Smalltalk
Wirth	Modula-2

4. The following table describes presidential visits. For each visit, the name of the president, and the country and year of visit is recorded. Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

Presidential visits:

Politician	Country	Year
Clinton	Australia	1994
Clinton	Italy	1994
Clinton	Australia	1995
Keating	Italy	1994

5. A student administration records the results of students for subjects being offered. The following table is an example of the way in which information is being exchanged in this UoD. Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

Student results:

student	course	rating
Bright S	CS112	7
Bright S	CS100	6
Collins T	CS112	4
Jones E	CS100	7
Jones E	CS112	4
Jones E	MP104	4

6. This exercise is a modification of previous exercise. The difference is that in this case course enrollment is also being registered. If a student has enrolled into a course, and does not get a mark yet, then the rating field is left open.

The following table is an example of the way in which information is being exchanged in this UoD. Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

Student results:

student	course	rating
Bright S	CS112	7
Bright S	CS100	
Collins T	CS112	4
Jones E	CS100	7
Jones E	CS112	
Jones E	MP104	4

7. The following table describes for students their class. Each class has assigned a mentor, the mentor of each student also is recorded in the table.

Students:

Student#	Name	Class	Mentor
001	Adams J	11A	Hiltin D
002	Brown C	12B	Baker P
003	Brown C	11A	Hilton D

Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

8. The tables below are used to administrate programming languages data.

Designers:

Designer	Language
Wirth	Pascal
Kay	Smalltalk
Wirth	Modula-2

Origins:

Designer	Language	Year
Wirth	Pascal	1971
Kay	Smalltalk	1972
Wirth	Modula-2	1979

Give the elementary sentences that can be derived from this table. Then derive from these sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

Chapter 2

The basic modeling technique

Version: 20-11-07

2.0.1 Complications

domain expert, system analyst

The complexity of information modeling is the nature of the human world to be modelled. Whereas a physician has to model physical concepts and their relations, information modeling is directed towards crossing the border between informal and formal where humans beings are required to cross this border. We assume a domain expert as a person that can assign meaning to statements about the universe of discourse. We assume a system analyst, also referred to as the architect, as the person making the model of this UoD. Some specific problems:

1. The universe of discourse is defined as the part of reality that is being observed. But how do we specify what part?
2. Domain experts may not report the actual UoD but *their* vision on this UoD.

2.1 Communication during modeling

A practical description of a universe of discourse typically is:

- detailed, in terms of relevant concepts of the UoD
- informal (concepts are not formally described)

As a consequence, such a description is hard to understand, and easily misunderstood, even by people that are familiar with the universe of discourse. For a model we need a description that is:

- formal (and therefore may even be used as a juridical contract)
- concrete (a program should specify all details that are required by the computer)

2.1.1 Schematically

In figure 2.1 we see that the following parameters seem to be dominant: (1) the level of detailedness, and (2) the level of formality. The ultimate goal, when building an information system, is to have a concrete formal description. The description should be concrete, as the system should not be in doubt about the intentions of the description. And the description should be formal, as computer systems can only be programmed in a formal language.

A concrete formal description is very hard for human beings to generate. Usually, domain experts have an understanding of their universe of discourse and can only provide an informal description, i.e., a description

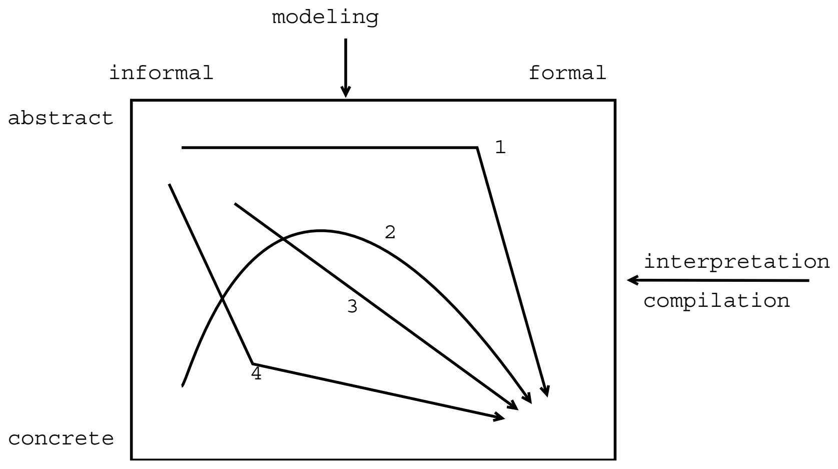


Figure 2.1: Levels of abstraction and formality

using an informal language (such as natural language). Their understanding is usually at a very concrete level, in terms of details of the universe of discourse.

In figure 2.1 these two parameters are correlated, enabling us to plot a modeling process seen from these two parameters. In practice, route 4 is a typical path. The domain expert will almost immediately start discussing about details. Transforming informal details into formal details is known to be an error prone and costly route. The ideal route is route 1. In this case, the transformation from informal to formal is done at the highest level of abstraction. A formal model may be seen as a best result from the modeling process. Mapping this formal model onto the characteristics of the underlying hardware may be done automatically, either by interpretation or by compilation.

2.1.2 A conceptual detour

In this lecture focus is on information modeling as an exchange process between a domain expert and a system analyst. Our intention is to describe this exchange process, and the underlying assumptions on its participants. This exchange process would be straightforward when domain expert and system analyst would speak a common language. In general, this is not the case, as the language used by the domain expert, the domain language, may be rather dissimilar from the language in which a system analyst has been trained, syntactically, semantically and also pragmatically. As a consequence, a direct communication process may not be possible, and a detour is required. This detour involves a common description in a language close to both participants.

In general, when different areas of expertise meet, natural language may be seen as the base mechanism for (human) communication. It is for this reason that each general modeling technique should support this basis for communication to some extent. Using natural language in a formalized way can, for example, be seen as a supplement to the concept of *use cases* (see [**?**]).

As a consequence, the quality of the modeling process is bounded by the quality of concretizing into a semi-formal description augmented with the quality of abstracting from this description. This triangular inequality is depicted in figure 2.2.

Roughly speaking, a domain expert can be characterized as someone with (1) superior detail-knowledge of the UoD but often (2) minor powers of abstraction of that same UoD. The characterization of a system analyst is the direct opposite. We will describe the required skills of both system analysts and domain experts from this strict dichotomy and pay attention to the areas where they (should) meet. Of course, in practice this separation is less strict. Note that as a result of the interaction during the modeling process the participants will learn from each other. The system analyst will become more or less a domain expert, while

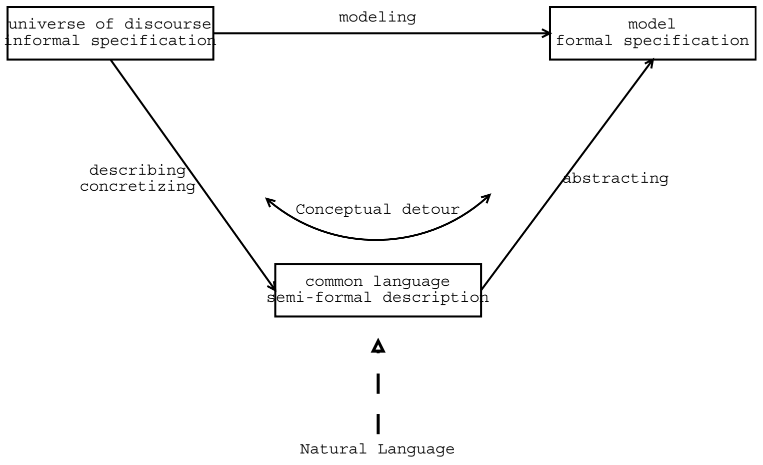


Figure 2.2: A conceptual detour

the domain expert will develop a more abstract view on the UoD in terms of the concepts of the modeling technique. This learning process has a positive influence on effectiveness and efficiency of the modeling process, both qualitative (in terms of the result) and quantitative (in terms of completion time).

2.1.3 Using natural language: proc and cons

As stated in [?], natural language is the vehicle of our thoughts and their communication. Since good communication between system analyst and domain expert is essential for obtaining the intended information system, the communication between these two partners should be in a common language. Consequently natural language can be seen as a basis for communication between these two partners. Preferably natural language is used in both the modeling process as well as the validation process. In practice, system analyst and domain expert will have gained some knowledge of each other's expertise, making the role of natural language less emphasized, moving informal specification towards formal specification.

For the modeling process, natural language has the potential to be a precise specification language *provided* it is used well. Whereas a formal specification can never capture the pragmatics of a system, an initial specification in natural language provides clear hints on the way the users wants to communicate in the future with the information system.

Since modeling can be seen as mapping natural language concepts onto modeling technique concepts, paraphrasing can be seen as the inverse mapping intended as a feedback mechanism. This feedback mechanism increases the possibilities for domain experts to *validate* the formal specification, see e.g. [?] and [?]. Besides the purpose of validation, paraphrasing is also useful to (1) lower the conceptual barrier of the domain expert, (2) to ease the understanding of the conceptual modeling formalism for the domain expert and (3) to ease the understanding of the UoD for the system analyst.

Up to now we focussed on the positive aspects of the usage of natural language, but in practice there are not many people who can use natural language in a (1) complete, (2) non-verbose, (3) unambiguous, (4) consistent way, (5) expressed on a uniform level of abstraction. In the sequel of this section we will make it plausible how the base skills for domain experts and systems analysts can reduce the impact of the these disadvantages, as these are the main critical succes factors of information modeling, of which the efficiency and effectiveness is a direct consequence.

Specifications in natural language tend to be verbose, hiding essentials in linguistic variety. Complex (verbose) sentences will be feed back to the domain expert for splitting and judging significance for the problem domain. A natural language specification may also get verbose by exemplification, providing examples (instantiations) of the same sentence structure.

An often raised problem of natural language usage is ambiguity, i.e. sentences with the same sentence structure yet having a different meaning. The system analyst should have a nose for detecting ambiguities. A typical clue comes from establishing peculiarities in instantiated sentences. In order to decide about a suspected ambiguity, the system analyst will offer the domain expert these sentences for validation.

On the other hand, the system analyst may also wish to elicit further explanation from the domain expert by requesting alternative formulations or more sample sentences with respect to the suspected ambiguity.

Sentences of a natural language specification are often on a mixed level of abstraction. As a system analyst has limited detail knowledge, and thus also limited knowledge at the instance level, a prerequisite for abstraction is typing of instances and map these types on the concepts of a modeling technique. The analysis of instances within a sentence is in fact a form of *typing*, attributing types to each of its components.

2.1.4 Controlled Language

controlled langauge

A controlled language is a restricted form of natural language, with only a limited number of constructions, and a restricted dictionary. Typically, a controlled language is a compromise between flexibility and expressivity of natural language and structure of formal language. Controlled languages are used when there is a requirement to reduce the possibility of misinterpretation. An example is the communication between airplane pilots and control units of airports.

An example of an early controlled language is the programming language COBOL (COmmon Business Oriented Language) (1959). COBOL uses an English-like syntax for common programming statements. A program division consists of sections. Sections consist of paragraphs, paragraphs consist of sentences. A sentence is an elementary grouping of statements in COBOL. COBOL has a rich arsenal of data types tuned for business applications (such as fixed-point arithmetics). COBOL arithmetic is simple but all fine tunings that are required in business applications (such as the specification of rounding and the handling of exceptions). Sections and paragraphs can be performed, as a counterpart of the procedure mechanism in other programming languages. The overwhelming popularity of COBOL is still impressive. As an example, consider the following COBOL Sentence:

```
Add Sales-Tax to Balance giving Amount-Due.
```

In a C-like language, this would be expressed as:

```
amountDue = salesTax + balance;
```

Another example is AECMA Simplified English (Association Europenne des Constructeurs de Matriel Arospatial). This controlled language limits the length of instructional sentences to no more than 20 words, forbids the omission of articles in noun phrases, and requires that sequential steps be expressed in separate sentences. This language is used in aerospace industry.

Another example is ACE (Attempto Controlled English, Rolf Schwitter). The following fragment (specifying the operations for updating a library database named LibDB) will illustrate the readability and expressive power of ACE:

```
If a borrower asks for a copy of a book
   and the copy is available
   and LibDB calculates the book amount of the borrower
   and the book amount is smaller than the book limit
   and a staff member checks out the copy to the borrower
then the copy is checked out to the borrower.

If a copy of a book is checked out to a borrower
   and a staff member returns the copy
then the copy is available.
```

In this course we will introduce the ORM controlled language.

2.2 The process of modeling

In this section we focus on the modeling process itself. What participants are assumed, and how do they cooperate.

2.2.1 The participants

The participants in the modeling process are:

1. the domain expert. This participant has expert knowledge of the universe of discourse, and is capable to generate significant well formed sample sentences to describe this domain. The domain expert is also capable to validate a description of the universe of discours, for example generated by the system analyst.

   domain expert

   The main concern of this participant is the completeness provided description of the universe of discourse.

   A more refined view to the domain expert is shown in figure 2.3.

2. the system analyst, also referred to as model builder. This participant is capable to derive the underlying grammar structure from well formed sample sentences. The system analyst is also capable to construct a description of the model under construction, providing the domain expert the opportunity to validate this model.

   System analyst

   The main concern of this participant is correctness of the resulting model. As has been argued, a model is deemed to be correct unless it has been falsified, the correctness concern is also referred to als the falsification concern.

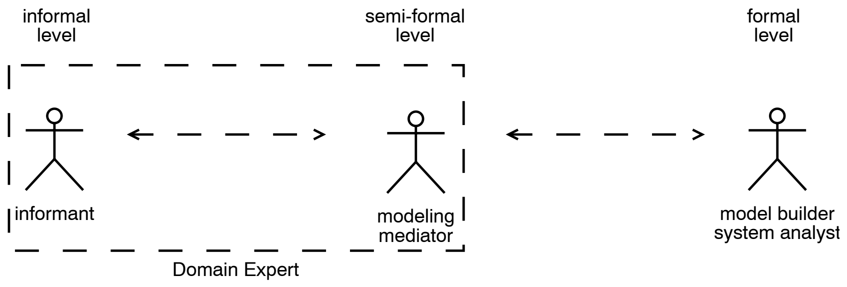


Figure 2.3: The modeling participants

modeling mediator, system builder

We restrict modeling as an activity between domain expert and system analyst. It might be argued that most of the actual modeling in fact takes place in the dialog between the informant and the modeling mediator. This is also referred to as the informal dialog. The informal dialog will be studied in a later course of the DaVinci Series.

The skills of domain expert and system analyst are complementary, and together are sufficient for effective modeling. A classification of skills is given in table **??**. The expertise of the domain expert is detailed

		domain knowledge	
		yes	no
abstraction skill	yes	experienced analyst/manager	system analyst
	no	domain expert	-

Figure 2.4: Skill overview

knowledge, not assuming skills to make abstractions. On the other hand, the system analyst is trained in making abstractions, but is not assumed to be familiar with the concepts of the universe of discourse. Communication is used to overcome the differences between these complementary skills.

2.2.2 The interaction

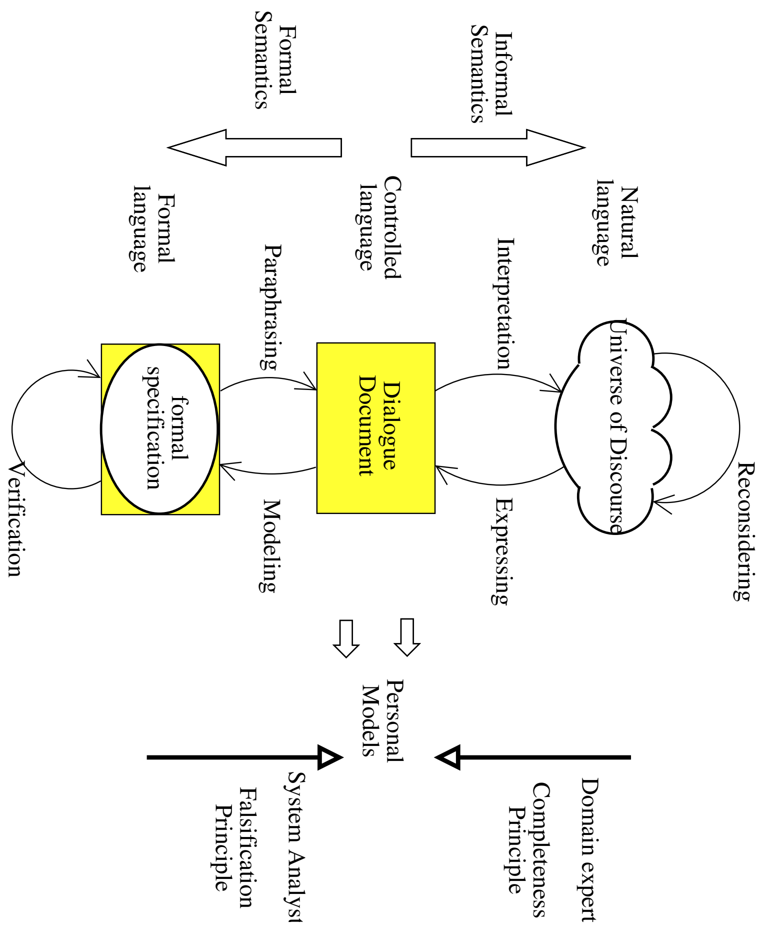


Figure 2.5: Domain expert and system analyst interaction

elicitation

The interaction process between domain expert and system analyst is displayed in figure **??**. Elicitation is the activity in which the system analyst tries to obtain sample sentences form the domain expert. We will see in a later section that sentences should be in a well agreed format, enabling the system analyst to understand the meaning of the sentence.

After the sample sentences have been elicitated, the system analyst derives from these sentences their underlying model as a contribution to the domain model. The system analyst will also provide a number of tests for consistency of the model (sofar). This quality process is called verification.

Finally, the system analyst generates sentences that are to be verified by the domain expert. The reason to do this might be twofold. Either the system analyst provides the description of (part of) the model to test for completeness, or the system analyst provides an assumption in order to test the validity of the underlying model property.

2.3 Modeling the communication

As stated in the previous sections, we focus on the (formal) communication within the UoD. Therefore we consider the UoD as a set of interacting agents. These agents are the members of the UoD. The term *agent* is used to emphasize that we do not restrict ourselves to human participants. The automated parts of the UoD also have a contribution to the communication within the UoD.

Human communication may be very complex. For example, psychological studies show that body language seems to convey most of the information that is exchanged by human beings (and other living creatures!). The problem is that such languages are hard to formalize. People may or may not recognize a signal uttered by body language, are may disagree about its meaning.

information-grammar

For modeling purposes we need a more stable basis. The term *formal* communication is used for such types of communication where the format and meaning of the messages is agreed on by the members of the UoD. The goal of the modeling activity is to make a model of the underlying format of the messages and to grasp the essentials of their meaning. This model is referred to as the *information grammar*.

Based on this model, a system may be constructed to support the members of the UoD to obtain information about their UoD. Such a system is referred to as an *information system*. Due to its nature, such a system is used to register the history of the UoD, and to support the other members of the UoD with relevant information.

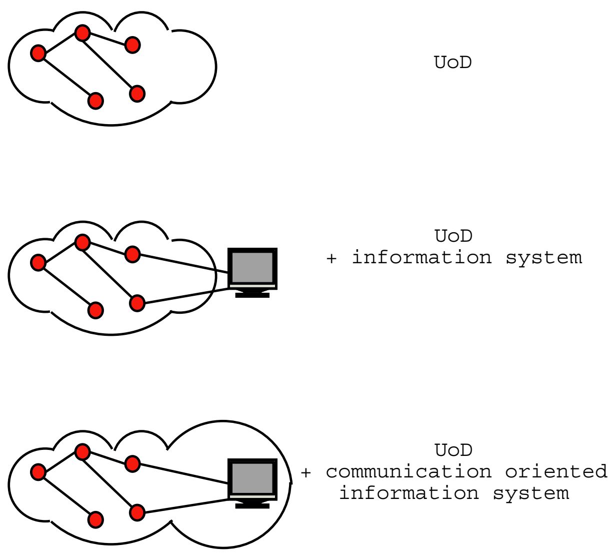


Figure 2.6: Communicating members

At this point we get a problem of a philosophical nature. After building an information system, the UoD will have a new member, the information system itself. As a consequence, the introduction of the information system may have led to a modification on the formal communication within the UoD. This can be represented by the following pseudo-formula:

$$\text{IS} = \text{Model (UoD + IS)}$$

where the problem is to solve IS from this recursive equation.

The goal of the modeling activity is to derive the information grammar describing the formal language within the UoD. The information grammar should be a *formal* and *abstract* description of the UoD.

100% principle, conceptualization principle

As a consequence, the information grammar:

1. *The 100% principle*: The conceptual schema should describe *precisely all* (relevant) conceptual aspects
2. *The conceptualization principle*: but does *not* provide any realization (implementation) aspect (see figure 2.1).

It is the actual focus on these principles that draws the distinction between the modeling technique described in this book and other modeling techniques (such as ER). The problem is that such techniques rely on modeling concepts that stem from an efficiency principle. By basing the modeling process on sentences, the resulting model will be close to the concepts as used in the universe of discourse.

2.4 Architecture of an information system

The information system has been introduced as an agent capable to register the history of the UoD, and to support the other members of the UoD with relevant information. The intelligence of this agent is based on the knowledge of how members of that UoD communicate, as described in the information grammar.

2.4.1 The user interface

The user interface describes the common part of interaction between a computer (program) and a user. The user interface allows the user to interact with the system. This interaction may be purely text based, or may be based on a metaphor of direct manipulation of graphical images and widgets in addition to text.

The main tasks for the information system are:

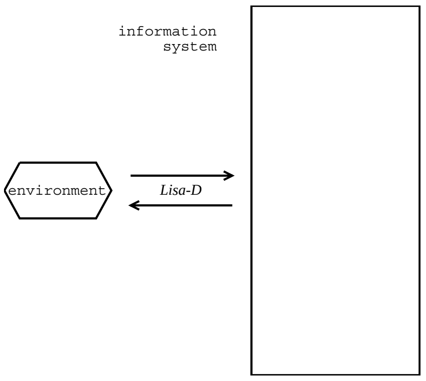


Figure 2.7: The user interface

- Accept information. This is the information that is required to keep the agent's knowledge of the vicissitudes of the UoD.
- Store information. The information system should be a reliable partner. It is supposed to remember what has been told.
- Process information. Processing can be very simple, answering some concrete question about the state of the UoD. However, more advanced questions are also possible, questions that appeal to the reasoning capabilities of the information system to derive information. An information system may also be seen as a knowledge system.
- Presentation of information. The information system will also try to present the requested information in a neat way, depending on the cognitive nature of the requesting agent. For example, a human agent will prefer visual information, while a system agent needs a very strict communication format (such as XML).

The communication with the information system is based on the information grammar. The information grammar describes the peculiarities of the UoD. Lisa-D is a general purpose extension of the information grammar, resulting in a complete formalized communication language.

2.4.2 The information base

The information system is continuously informed about the relevant happenings in the UoD. The sentences used are instances according to the information grammar. For example, the following mentions cold have been made to the information system:

1. student 12345 enrolls in course A
2. student 23456 enrolls in course B

Using the information grammar, the information system is capable to detect the underlying structure of these sentences. This enables the information system to correctly understand these sentences. Besides they have the same structure, these sentence mention different facts. In order to remember these facts, the system maintains for each sentence type from the information grammar a pool of instances that have been mentioned.

information base

In this case, the information system will store in the pool for fact type *enrolls in* the following actual enrollments: (12345, A) and (23456, B). The fact pools together form the *information base* of the information system.

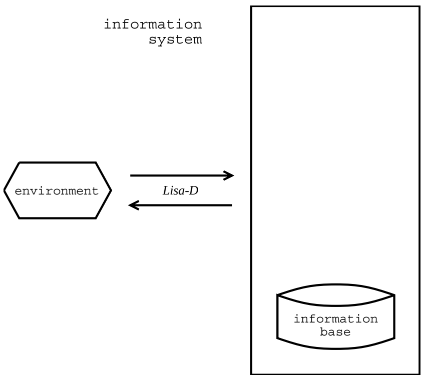


Figure 2.8: The information base

Closed World Assumption

The rationale behind this storing strategy is the *closed world assumption*, which states:

a fact is valid
$\Leftrightarrow$ the fact is stored in the information base

The information base thus is supposed to provide a complete description of the current state of the UoD.

In our model theoretic view, the information base is assumed to record its knowledge about (the current state) of the universe of discourse as elementary sentences. Sentences are elementary when they can not be split into a combination of smaller sentences with the same meaning. The consequence of this rule of being elementary is that the information base does not suffer from redundancy. Or, each fact stores a separate information fragment that is not available in another information fragment.

The information system will use an internal language to process the information base. Typically, the information base is organized according to the relational data model, and processed by the operational data processing language SQL (Structured Query Language).

2.4.3 The information grammar

The information base forms the minimal required storage to represent the current state of the universe of discourse. The current state is being described by recording all statements that are valid in the universe of discourse. The facts stored are the elementary statements to be made about the universe of discourse.

The description of these sentences, the rules to construct more complex sentences, and the constraints that govern the contents of the information base, are described in the information grammar. The information grammar provides a model of the underlying format of the messages that can be exchanged by members of the UoD and to grasp the essentials of their meaning.

In some more detail, the information grammar will describe:

- A description of the concepts that are communicated about in the UoD, and their relations. This description is also referred to as the *conceptual model*. These concepts are grammatical units (so called *non terminal symbol*) in the information grammar. The relations between the concepts are modelled as *context free rules* in the information grammar. As a consequence, the context free rules are used to describe the structure of the UoD.
- The constraints that govern the way this model may be instantiated (populated). Especially, there may be constraints on state changes. In technical terms, restriction rules are non-context free rules in the information grammar that restrict the context free structural derivations.

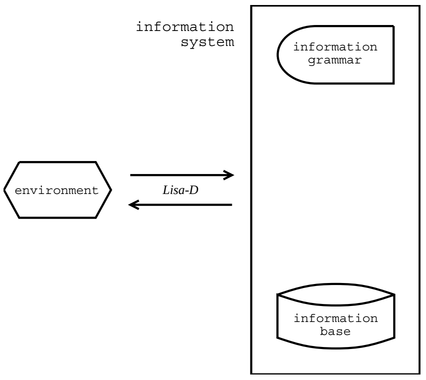


Figure 2.9: The information grammar

- The actual grammar rules. These rules describe how concepts are communicated. In this part the communication units used in the user interface (forms, windows, etc) are also described.
- The lexicon describes the elementary names and their meaning. This is the same approach as taken for natural language descriptions, they also discriminate between grammar rules and a lexicon.

In this course, it is assumed that the information grammar does not change over time. In practise this is not realistic, system evolution is more a rule than an exception. In such a situation, it will also be required to answer historical questions. This will require to have an overview of the history of the conceptual schema, and an historical overview of facts in terms of that conceptual history.

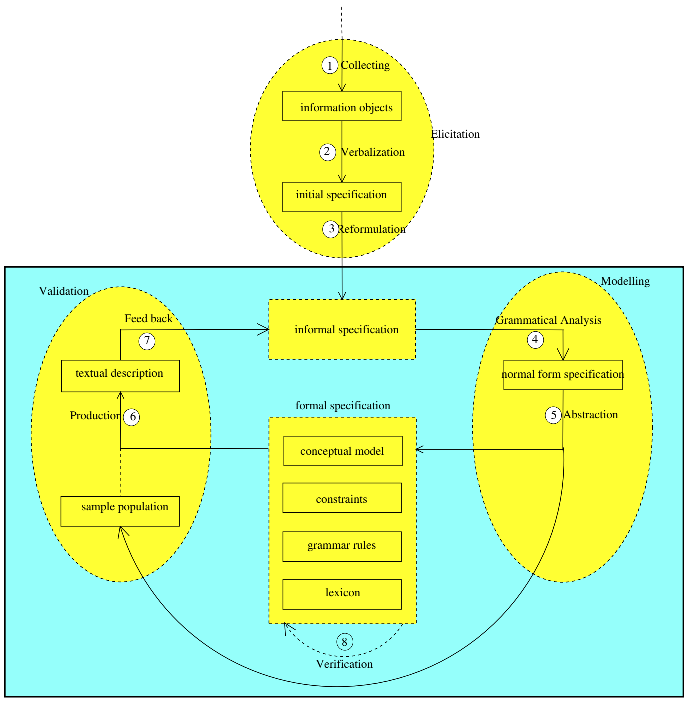


Figure 2.10: The development cycle

In figure 2.10 the schema of the modeling process is further elaborated, in order to reflect these elements of the information grammar.

For the description of the conceptual model, graphical techniques are used. The reason is that conceptual models tend to be very large, and therefore not effectively accessible by a textual description. This is a serious drawback for the validation of the conceptual model (see 2.10) We will use the graphical notation originating from NIAM (aN Information Analysis Method), with extensions from PSM (Predicator Set Model), ORM (Object Role Modeling) and FCO NIAM (Fully Communication Oriented NIAM).

We will also show how the information grammar may be depicted using the contemporary modeling technique UML (Unified Modeling Language).

2.4.4 The information processor

The information processor accepts requests that are entered via the user interface (section 2.4.1), takes care for their execution, and presents their results. *information processor*

The main steps taken by the information processor are:

1. parse request according to the information grammar in order to understand the nature of the request in terms of conceptual database operations
2. translate the conceptual operations in terms of operation according to the actual database management system used (usually SQL-based)
3. transform the results according to the rule to transform the resulting data into information (as described in the actual grammar rules)

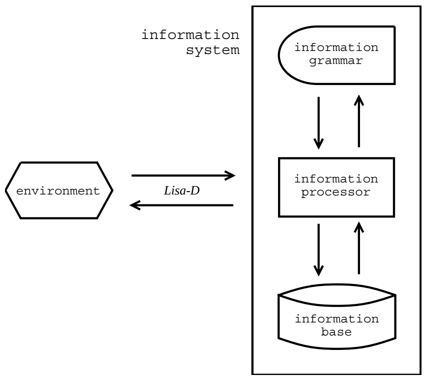


Figure 2.11: The information processor

The information processor is the active component of the information system. We take a straightforward approach, and assume the information system to be a single-processor system. The reason is that data in the first part of this lecture is considered as a passive element.

In the second part of this lecture, we will have an more active view on data as active objects.

2.4.5 Turing's Test Of Intelligence

Programming languages (see figure 2.13) may be characterized from its usability aspect, in terms of the level of communication with the computer, as follows:

Figure 2.12: Turing test for intelligence

1. 1st generation: Machine Code
   Such languages consist of hexadecimal codes and correspond to the operations a digital processor can perform. For example, the Z80 processor will interpret the following code:

   ```
   C3 05 00
   ```

   as follows: proceed execution with the instruction at address 0100 (hexadecimal!).
2. 2nd generation: Assembly Language
   In order to make life more easy for programmers, mnemonics are introduced for alle processor instructions. Furthermore, a mechanism to handle addresses by labels. The above instruction

   ```
   jmp     BDOS
   ```

   The label `BDOS` is defined elsewhere in the assembly program, for example as follows:

   ```
   BDOS     equ      0005h
   ```

   A program called *assembler* translates an assembly program into machine code.
3. 3th generation: Procedural Languages
   Such languages are also called high-level programming languages. Examples are Algol, Pascal, C, C++, and Java. These languages allow the programmer to focus on solving the problem without

worrying about the actual computer on which the program has to run. Special programs, called *compilers* translate the program into the machine code for a specific computer.

4. 4th generation: Non-procedural (declarative) Languages
In 4th generation languages, the programmer only specifies what has to be done rather than giving a procedure of how this is actually obtained. A typical 4th generation language is SQL (Structured Query Language).

SQL

5. 5th generation: (Controlled) natural languages
5th Generation systems recognize natural language like sentences.

   An example is the language Lisa-D (Language for Information Structure and Access Descriptions). An interesting test for the quality of such a system it the Turing Test (see figure 2.12). The idea is as follows. Human A 'converses' via 'chat' software with person B on the other side of an opaque wall (via a computer terminal) in natural language about a desired topic. Human B eventually hands over the conversation to Machine C. If A can tell the difference between B and C, then the machine is *not* intelligent, otherwise it is.

Lisa-D

Turing test

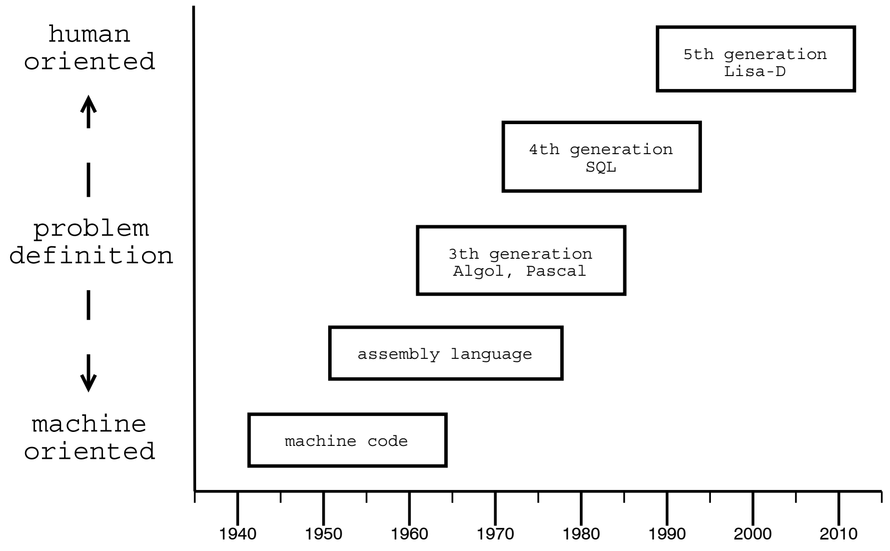


Figure 2.13: Programming language history

A practical variant of the Turing test is called *captcha*, which is an acronym for *completely automated public Turingtest to tell computers and humans apart*. A captcha (see for for example figure 2.14) is a test that can be generated by a computer, such that humans can pass the test, but not readily solvable for computer programs.

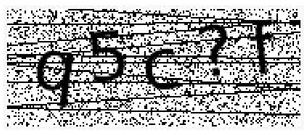


Figure 2.14: Captcha test

2.5 Grammars

Languages are widely used for communication purposes. They are restricted to formal human communication. Being formal, there are certain

rules for the proper use of language. This is the basis for communica-
tion to be successful, as structure is a prerequisite for understanding. Generally accepted is the view that a languages defines sentences as the base units for communication. The rules that sentences should adhere to are called the grammar of that language. These rules are used in the context of this course in a rigid way, in contrast to daily practise. Human beings have a great flexibility in recognizing sentences that do not obey the grammar rules. Sentence fragments are used to convey a complete sentence, illformed sentences are easily understood. The reasons why such forms of sloppy use of language still work are: sentences contain information in a redundant way, sentences are presented in a context that clarifies missing information and even provides opportunities for error correction.

The reason for the rigid approach in this course are twofold. The first reason is a quality argument. Sentences uttered by an information system should be free of misinterpretation. The second reason is that an information system should not be tempted with the meaning of incoming sentences.

The intention of a grammar is to describe what sentences are acceptable in a particular language. Grammars are used by human beings both to construct wellformed sentences and to understand them. The base rule is that a sentence is a sequence of words, intermingled with punctuation marks to enforce some structure in the sentence, or to emphasize some part of it. Lexica define the words than are acceptable for a language.

Syntactical categories are used to describe the structure of sentences. Examples of syntactic categories are: predicate, subject and object. For example, a valid sentence may consist of a subject, followed by a verb, followed by an object. We use the AGFL formalism to describe this grammar rule for the syntactic category `sentence`:

```
sentence:
  subject, verb, object.
```

Another view at this rule is: during generation of a valid sentence, the syntactic category `sentence` may be replaced by the sequence of `subject`, `verb` and `object`. The comma-symbol is used in the rule to separate the syntactic category to be rewritten from the result of this rewriting. The comma symbol is used to separate syntactical categories, the point symbol marks the end of the rule. Another examples of a grammar rule is:

```
subject:
  "I";
  "you".
```

This rule expresses that the syntactic category `subject` may be rewritten either by the word ”I” or by the word ”you”. The quotes thus mark the actual words. The semicolon is a separator for alternative rewrite patterns. Using semicolons leads to a more compact notation for rules. The above example is a compact notation for the following two rules:

```
subject: "I".

subject  "you".
```

We add some more rules:

```
verb:
  "amuse";
  "call".

object:
  "you";
  "the man".
```

From this set of rules, we can derive sentences. We start with the syntactic category `sentence`, and apply rules until all syntactic categories have been resolved. For example:

expression	**apply rule**
`sentence`	`sentence: subject, verb, object.`
`subject, verb, object`	`subject: "I".`
`"I", verb, object`	`verb: "call".`
`"I", "call", object`	`object: "you".`
`"I", "call", "you"`	

This is also an example of the usage of a grammar as a *generative* device, i.e., a device that can generate sentences. Grammar rules may also be used to parse a given sentence, leading to the central start symbol `sentence`:

expression	**apply rule**
`"I", "call", "you"`	`object: "you".`
`"I", "call", object`	`verb: "call".`
`"I", verb, object`	`subject: "I".`
`subject, verb, object`	`sentence: subject, verb, object.`
`sentence`	

This example shows the usage of a grammar as a *parsing device*. Invalid sentences will fail a trial reduction to the start symbol `sentence`, for example:

expression	**apply rule**
`"I", "smell", "you"`	`object: "you".`
`"I", "smell", object`	`subject: "I".`
`subject, "smell", object`	??

Besides natural language, grammars are also in use to define artificial languages, such as programming languages. However, the descriptive capabilities of grammars go far beyond these examples, and are also used to describe all kinds of formal structures.

2.5.1 Formal grammars

The example grammar of the previous subsection can generate meaningless sentences, for example:

expression	**apply rule**
`sentence`	`sentence: subject, verb, object.`
`subject, verb, object`	`subject: "I".`
`"I", verb, object`	`verb: "call".`
`"I", "call", object`	`object: "I".`
`"I", "call", "I"`	

The reason is that the second application of the rule `object: "I".` does not make sense, as earlier in the derivation this rule was also applied. This is better stated as follows: in the context of `"I", "call"`, the rule `object: "I".` should not be applied. We need some mechanism to make rewrite rules behave like this. In AGFL, syntactic categories get a mission statement upon application of the grammar rule in the form of parameters that describe the specific content requirements.

Formal grammars provide the opportunity to study properties of grammars in an abstract way. Abstract grammars use symbols as an abstraction of words and syntactic categories. Symbols that can be rewritten are called nonterminal symbols, symbols that correspond to words are called terminal symbols. We will not further go into the details of formal grammars.

Courses on formal grammar theory focus on the essentials of generative and parsing devices. An abstract grammar is defined as a structure $\langle \mathscr{N}, \mathscr{T}, S, \mathscr{R} \rangle$, where $\mathscr{N}$ is the set of nonterminal symbols (abstraction of syntactical category), $\mathscr{T}$ is the set of terminal symbols (abstraction of words), S the start symbol (the main syntactical category, in our example `sentence`), and $\mathscr{R}$ the set of grammar rules. The example grammar of the previous subsection is represented as:

1. $\mathscr{N} = \{S,A,B,C\}$
2. $\mathscr{T} = \{a,b,c,d,e,f\}$
3. startsymbol: S
4. $\mathscr{R}$: the rules are:
    - $S : A,B,C.$
    - $A : a;b.$
    - $B : c;d.$
    - $C : e;f.$

2.5.2 Handling context

Next we consider a grammar that describes the presence of the conjugation of the verb *to walk*. The conjugation depends on the person and the count of usage. Let the parameter `PERSON` register the requested person, and the parameter `NUMBER` plurality. Then the grammar could look like:

```
sentence(PERSON,NUMBER):
  subject(PERSON,NUMBER), verb(PERSON,NUMBER).

subject(FIRST,SINGULAR):    "I".
subject(SECOND,SINGULAR):   "you".
subject(THIRD,SINGULAR):    "he"; "she", "it".

subject(FIRST,PLURAL):      "we".
subject(SECOND,PLURAL):     "you".
subject(THIRD,PLURAL):      "they".

verb(FIRST|SECOND,SINGULAR): "walk".
verb(THIRD,SINGULAR):        "walks".
verb(PERSON,PLURAL):         "walk".
```

The rule

```
sentence(PERSON,NUMBER):
  subject(PERSON,NUMBER), verb(PERSON,NUMBER).
```

describes the general construction for any value of the parameters `PERSON` and `NUMBER`. In the rule

```
subject(FIRST,SINGULAR):   "I".
```

the parameters `PERSON` and `NUMBER` are completely specified. The bar symbol (|) allows the grouping of alternatives:

```
verb(FIRST|SECOND,SINGULAR): "walk".
```

The possible substitutions for the parameters are specified by:

```
PERSON::FIRST|SECOND|THIRD.

NUMBER::SINGULAR|PLURAL.
```

A sample derivation of a sentence:

expression	**apply rule**
`sentence(THIRD,SINGULAR)`	`sentence(THIRD,SINGULAR): subject(THIRD,SINGULAR), verb(THIRD,SINGULAR).`
`subject(THIRD,SINGULAR), verb(THIRD,SINGULAR)`	`subject(THIRD,SINGULAR): "she".`
`"she", verb(THIRD,SINGULAR)`	`verb(THIRD,SINGULAR): "walks".`
`"she", "walks"`	

Another example is:

expression	**apply rule**
`sentence(THIRD,PLURAL)`	`sentence(THIRD,PLURAL): subject(THIRD,PLURAL), verb(THIRD,PLURAL).`
`subject(THIRD,PLURAL), verb(THIRD,PLURAL)`	`subject(THIRD,PLURAL): "they".`
`"they", verb(THIRD,PLURAL)`	`verb(THIRD,PLURAL): "walk".`
`"they", "walk"`	

2.5.3 ORM represented as AGFL

The information grammar displayed in figure 1.10 is represented in AGFL as:

- `Sentence:: WorksFor; Cooperator; Department; Name; Departmentname.`

- `WorkFor:: Cooperator , "works for", Department.`
- `WorkFor:: Department , "providing work to", Cooperator.`

- `Cooperator:: "Cooperator with", Name.`
- `Department:: "Department with", Name.`

- `Name:: "Name", string.`
- `Departmentname:: "Department name", string.`

For the information grammar from figure 1.11 we have the following corresponding AGFL grammar:

2.5.3.1 2

- `Sentence:: Smokes; Person; Name.`

- `Smokes:: Person , "smokes".`

- `Person:: "Person with", Name.`

- `Name:: "Name", string.`

For the schema fromfigure—2.19 we get:

- `Sentence:: Reservation; AName; Room; Time; Activity; Nr; Dh; Code; ActivityName.`

- `Reservation:: Room, "at ", Time, "is used for", Activity.`
- `AName:: Activity, "has", ActivityName.`

- `Room:: "Room with", Nr.`
- `Time:: "Time with", Dh.`
- `Activity:: "Activity with", Code.`

- `Nr:: "Nr" , string.`
- `Dh:: "Dh", string.`
- `Code:: "Code", string`
- `ActivityName:: "ActivityName", string`

2.6 The formal model

A main part of the information grammars is the overview of concepts and how they are related. This overview is called the *conceptual schema*. We will first focus on the conceptual schema, and later see how this is used as a base to construct the information grammar.

As conceptual schemata, in real applications, tend to be rather extensive, it has become common practise to present conceptual schemata graphically. A conceptual schema in the form of as diagram has shown to be a good format for discussion between domain expert and system analyst.

2.6.1 Object types

A conceptual schema consists (at least) of the following schema elements:

1. A set of label types ($\mathscr{L}$). Each label type has associated a concrete domain (such as **string**, **real** or **integer**). The concrete domain may also be an enumerated set of values.
2. A set of entity types ($\mathscr{E}$). Entities correspond to the members of the universe of discourse.
3. A set of fact types ($\mathscr{F}$). Facts describe the relations between members of the universe of discourse.

Each instance of these sets of types corresponds to some object sort. We will use the term *object type* as a generic term. Each object type will correspond to a separate syntactical category in the information grammar.

elementary object types

Label types and entity types are elementary object types. Fact types are constructed object types. The construction rules will be discussed in the next section. The graphical representations of elementary object types is displayed in figure 2.15.

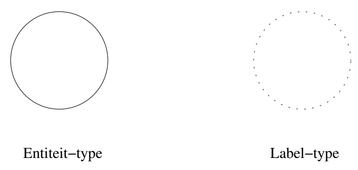


Figure 2.15: Graphical representation of elementary types

2.6.2 Construction rules

Roles are used as the primitive elements to construct new object types. The set of roles is denoted as $\mathscr{P}$. A fact type ($f \in \mathscr{F}$) basically is a set of roles ($f \subseteq \mathscr{P}$). Roles describe the kind of roles that are played in the corresponding fact type. A role is displayed graphically as follows:

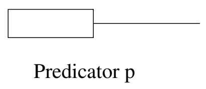


Figure 2.16. Graphical representation of a role

Each role p has associated a base type $\mathsf{Base}(p)$.

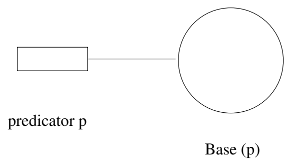


Figure 2.17. Base of a role

The fact type in which role p is used is denoted as $\mathsf{Fact}(p)$. Roles are grouped into fact types by drawing them next to each other. Sometimes the following representation is also used:

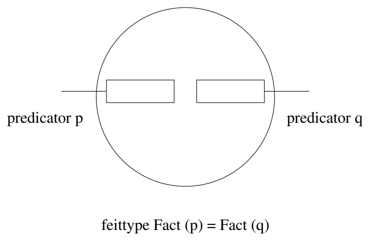


Figure 2.18. Fact type

2.6.3 Bridge types

A special sort of fact type is the bridge type. Such a fact type connects the abstract world of the information system with the concrete world, the universe of discourse. A bridge type is a binary fact type, connecting a label type with an entity type. Formally, a bridge type is a fact type with two roles p and q such that:

$$\mathsf{Base}(p) \in \mathscr{E} \wedge \mathsf{Base}(q) \in \mathscr{L}$$

Bridge types form the only possibility to go pass the border between the concrete and abstract world. This is expressed by the following rule of wellformedness of conceptual schemata:

$$\mathsf{Base}(p) \in \mathscr{L} \Rightarrow \mathsf{Fact}\text{(p) is a bridge type}$$

Bridge types have associated standard role names, leading to fixed verbalization rules for the facts stored in this object type. The structure of these verbalizations is:

- entity with label
- label of entity

2.6.4 Summary sofar

At this point, a conceptual schema consists of:

1. a set $\mathscr{L}$ of label types; each label type has associated a concrete domain.
2. a set $\mathscr{E}$ of entity type.
3. label types and entity types are elementary object types.
4. a set $\mathscr{P}$ of roles
5. a set $\mathscr{F}$ of fact types. The fact types form a partition of the set $\mathscr{P}$ of roles.
6. $\mathsf{Base}(p)$ is the base of role p, denoting the object type playing the role associated with this role.
7. $\mathsf{Fact}(p)$ is the fact type in which p is used as a constructor.

2.6.5 Example

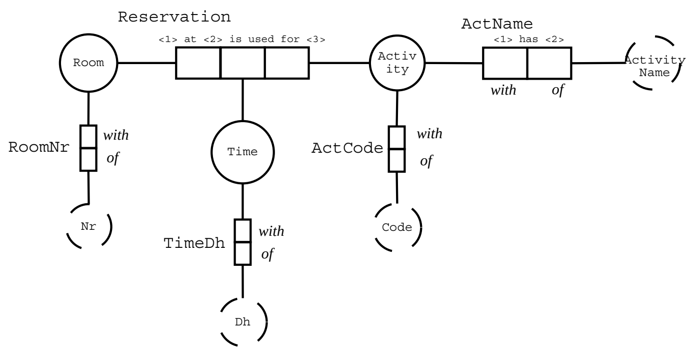
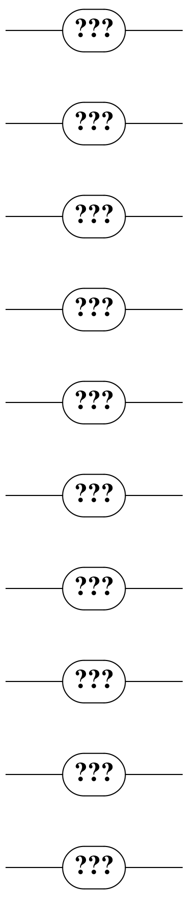


Figure 2.19: Conceptual schema for room reservation

. The example schema from figure **??** is described via syntax diagrams by:



Using AGFL, we obtain the following description:

- `Sentence:: Reservation; AName; Room; Time; Activity; Nr; Dh; Code; ActivityName.`

- `Reservation:: Room, "at ", Time, "is used for", Activity.`
- `AName:: Activity, "has", ActivityName.`

- `Room:: "Room with", Nr.`
- `Time:: "Time with", Dh.`
- `Activity:: "Activity with", Code.`

- `Nr:: "Nr" , string.`
- `Dh:: "Dh", string.`
- `Code:: "Code", string`
- `ActivityName:: "ActivityName", string`

It has the following formal description:

1. $\mathscr{L}$ = { Nr, Dh, Code, ActivityName }
2. $\mathscr{E}$ = { Room, Time, Activity }
3. $\mathscr{P}$ = { Reservation.room, Reservation.time, Reservation.act, RoomNr.with, RoomNr.of, TimeDh.with, TimeDh.of, ActCode.with, ActCode.of, ActName.with, ActName.of }

4. $\mathscr{F}$ = { Reservation: {Reservation.room, Reservation.time, Reservation.act},
   RoomNr: {RoomNr.with, RoomNr.of},
   TimeDh: {TimeDh.with, TimeDh.of},
   ActCode: {ActCode.with, ActCode.of},
   ActName: {ActName.with, ActName.of} }
   Note that $\mathscr{F}$ is a partition of the set of roles.

5. Base(p):

role	base
Reservation.room	Room
Reservation.time	Time
Reservation.act	Activity
RoomNr.with	Room
RoomNr.of	Nr
TimeDh.with	Time
TimeDh.of	Dh
ActCode.with	Activity
ActCode.of	Code
ActName.with	Activity
ActName.of	ActivityName

6. Fact(p):

role	fact type
Reservation.room	Reservation
Reservation.time	Reservation
Reservation.act	Reservation
RoomNr.with	RoomNr
RoomNr.of	RoomNr
TimeDh.with	TimeDh
TimeDh.of	TimeDh
ActCode.with	ActCode
ActCode.of	ActCode
ActName.with	ActName
ActName.of	ActName

2.6.6 Populating a conceptual schema

population

A conceptual schema describes concepts and how they relate. The state of the universe of discourse is then described as an overview of the valid elementary sentences, i.e., the actual instances of the object types. Such an overview is called a *population* of this conceptual schema. The consequence of the Closed World Assumption is that all elementary sentences not comprised in a population can be concluded to be false in the corresponding state of the universe of discourse. Formally: a population

1. assigns a set of sentences to each object type, constructed according to the construction rules as layed down in the conceptual schema
2. such that all further constraints are satisfied.

Constraints will be introduced in chapter 5. As constraints are used to exclude well-constructed populations, that nevertheless are no proper representation of any state of the universe of discourse.

Populations can be denoted as follows:

1. in a formal way, using notation from set theory
2. using a graphical format

3. using a table-notation

Figure 1.9 shows how a population is represented graphically. The table-notation lends itself best to be used as an efficient storage format of the information base in a computer:

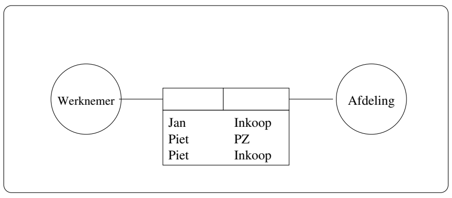


Figure 2.20. Sample population

2.7 Quality aspects for conceptual schemata

The modeling task, as described in section 2.2.2, is a process of communication and negotiation between the domain expert and the system analyst. Basically, this interaction between domain expert and system analyst leads to a set of sample sentences in ORNF. This is called the *semi-formal specification*. From the point of view of the system analyst, the constructed information grammar is correct is all this grammar can generate all sample sentences. A first point to note is that there may be several information grammars that meet the semi-formal specification. As a very trivial example, the grammar that simply generates each sample sentence directly from the start symbol of the grammar, is a correct information grammar.

The question then is: what is the best information grammar to choose? We will restrict this to the question: what is the best conceptual model given the semi-formal specification. We have the following conflicting requirements:

- the system analyst should try to avoid unnecessary concepts. In other words, the conceptual model should have a minimal number of concepts.
- the grammar should recognize all candidate concepts. The system analyst should recognize all sentence parts where the domain expert might possibly want to add variation. The system analyst should thus be capable to learn even from a very small set of examples.

Even then there may be more than one information grammar derivable from the semi-formal specification. The system analyst might want to offer the domain expert feedback questions that enable the system analyst to prefer one model above the other. The system analyst has to make the ultimate choice if the domain experts preferences fail.

2.8 Definitions overview

[informal-communication-us] –

[formal-communication-us] –

[telephone-heuristic-us] –

[fact-oriented-us] –

[action-oriented-us] –

[ornf-us] –

[agens-us] –

[patiens-us] –

[index-expression-us] –

[weak-identification-us] –

[standard-name-us] –

[strong-identification-us] –

[elementary-sentence-us] –

[deep-sentence-structure-us] –

[fully-qualified-us] –

[lexical-object-us] –

[nonlexical-object-us] –

[identifying-attribute-us] –

[bridge-type-us] –

[functional-dependency-us] –

[multivalued-dependency-us] –

[elementary-object-types-us] –

[population-us] –

Questions

Version: 01-12-2007

1. The following table gives an overview of the participants of an athletic contest, the country they represent and thier birth country.

   Athlete administration:

Athlete	Country	Birthplace
Ann Arbor	USA	USA
Bill Abbot	UK	?
Chris Lee	USA	NZ

   Give the elementary sentences that can be derived from this table. Then derive from thee sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts. Also give the associated AGFL grammar.

2. In the following table a wholesale dealer describes how many software items have been sold to retailers.

   Sales administration:

Software item	Retailer	Quantity sold
SQL+	CompuWare	330
	SoftwareLand	330
Zappo Pascal	CompuWare	330
	SoftwareLand	251
WordLight	CompuWare	200

   Give the elementary sentences that can be derived from this table. Then derive from thee sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

3. In the following project administration for each employee it is registered how many hours they have made on projects, and what expenses they had:

Project administration:

Employee	Project	Hours	Expenses
E4	P8	24	200
E4	P9	26	150
E5	P8	14	100
E5	P9	16	110
E6	P8	16	120
E6	P9	14	110

Give the elementary sentences that can be derived from this table. Then derive from thee sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

4. A university keeps track of the tute groups they organize for their students, and where and when they have their weekly meetings:

Tutorial allocations:

Tute group	Time	Room	StudentNr	StudentName
A	Mon	CS-718	302156	Blogs FB
	15.00		180064	Fletcher JB
			278155	Jackson B
			334067	Jones EP
			200140	Kawamoto T
B1	Tue	E-B18	266010	Anderson AB
	14.00		348112	Bloggs FB

Give the elementary sentences that can be derived from this table. Then derive from thee sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

5. Discuss the following statement:

if a population is not weakly identified, then there can be no unique correspondence with the corresponding UoD.

6. The personnel administration of a university registers information of each employee as follows:

Personnel administration:

EmpNr	Emp Name	Dept	Room	Phone Ext	Phone Access	Tenure/ Contract-expiry
715	Adams A	Computer Science	69-301	2345	LOC	01/31/95
720	Brown T	Biochemistry	62-406	9642	LOC	01/31/95
139	Cantor G	Mathematics	67-301	1221	INT	tenured
430	Codd EF	Computer Science	69-507	2911	INT	tenured
503	Hagar TA	Computer Science	69-507	2988	LOC	tenured
651	Jones E	Biochemistry	69-803	5003	LOC	12/31/96
770	Jones E	Mathematics	67-404	1946	LOC	12/31/95
112	Locke J	Philosophy	1-205	6600	INT	tenured
223	Mifune K	Elec. Engineering	50-215A	1111	LOC	tenured
951	Murphy B	Elec. Engineering	45-B19	2301	LOC	01/03/95
333	Russel B	Philosophy	1-206	6600	INT	tenured
654	Wirth N	Computer Science	69-603	4321	INT	tenured

Give the elementary sentences that can be derived from this table. Then derive from thee sentences an ORM schema. Discuss the correctness of this schema, and explain that this schema does not contain unnecessary concepts.

7. Beschouw de volgende zinnen:

   (a) Student Jan volgt vak DM

(b) Studente Ria volgt vak P1

(c) Studente Els heeft voor vak DM een 10 gehaald

(d) Student Cor heeft voor vak III een 7 gehaald

Maak een grammatica die in staat is (in elk geval) deze zinnen voort te brengen.

8. Sales administration:

EmpNr	Emp Name	Subject	Rating	Committees
715	Adams A	CS100 CS101	5	
430	Codd EF			
654	Wirth N	CS300		BSc-Hons CAL Advisory

9. Figuur 2.21 is een voorbeeld van de manier waarop men in een zeker UoD onderling informatie uitwisselt. Geef de elementaire verwoordingen van de elementaire feittypen die hierin liggen opgesloten. Leid vervolgens uit deze zinnen het ORM schema af. Motiveer waarom uw schema correct is.

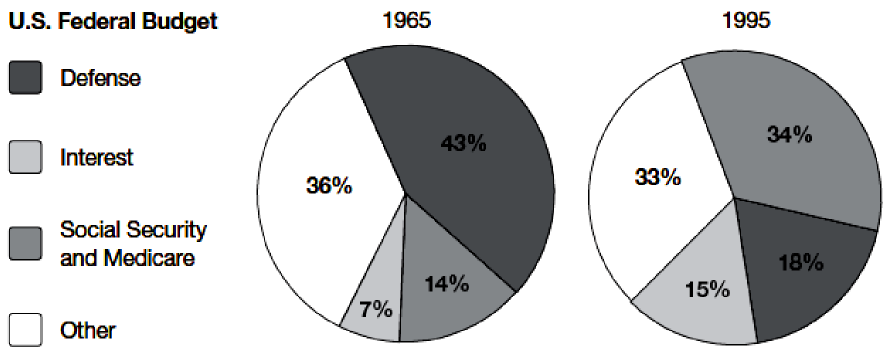


Figure 2.21: US Federal Budget expenses

Chapter 3

Object-Role Calculus - The language

Version: 31-10-07

In the previous sections we have seen how the base sentences obtained from the domain expert have been converted into a base grammar. However, this grammar is only capable to represent the way facts are described in the universe of discourse. In this chapter we extend this grammar such that is may also be used to formulate a query to be processed by the information system (see section 2.4.1).

We will assume a provisional information system with a most simple user interface. The system is waiting for a request from the user, and prompts the user by displaying a >-sign as follows:

```
 > □
```

The user is to enter a request, upon which the system will respond. Next the system will ask for a next request. A request is a (series of) sentence(s) from the grammar to be described in this chapter, followed by a full stop symbol.

The conceptual language to communicate with the information system has the following categories of sentences:

1. schema sentences: these are the sentences that describe the schema elements that constitute the conceptual model. From these sentences the information grammar is derived. *schema sentences*
2. stating sentences: a stating sentence informs the information system about new facts in the universe of discourse, or about the withdrawal of the validity of facts specified. The information base is modified accordingly. *stating sentences*
3. asking sentences: such a sentence gives the specification of an information need. The system will respond with a list of stating sentences, corresponding to the facts that satisfy the specification. *asking sentences*

3.1 Schema sentences

Initially the information system does not contain any conceptual schema specification and thus also has an empty information base. At this moment, the information system will only recognize sentences from its base grammar. This base grammar is such that the user can enter a sentence to describe a conceptual schema. In this course schema sentences will be discussed only briefly.

A schema sentence is a complex sentence that describes all schema elements successively. First the label types are introduced. Next the entity types, followed by the description of the fact types. We will not discuss these schema sentences in detail but assume they are sufficiently self-explaining. The next sample session describes the introduction of the schema displayed in figure 3.1

```
> ADD SCHEMA TYPE Example
    LABEL TYPE
      Name HAS DOMAIN String VERBALIZED AS 'Name' *; # default name

    ENTITY TYPE
      Person (Name) VERBALIZED AS 'Person' 'with' ⟨1⟩; # default name
      Department (Name);

    FACT TYPE
      Contract (
        in: Person (from) working for,
        has: Department (for) providing work to
        VERBALIZED AS ⟨1⟩ 'works for' ⟨2⟩
      );

    CONSTRAINTS
      C1: Total role on Contract.in : TOTAL ROLE Contract.in
      C2: Unique role on Contract.in : UNIQUE ROLE Contract.in

  ENDSCHEMA.
Accepted
```

This schema describes two so-called constraints. Constraints will be discussed in the next chapter. For sake of the example, we introduced two constraints referred to as C1 and C2. The first constraint requires each person to be involved in some contract, while the second constraint enforces that a person may not be involved in more than one contract at a time.

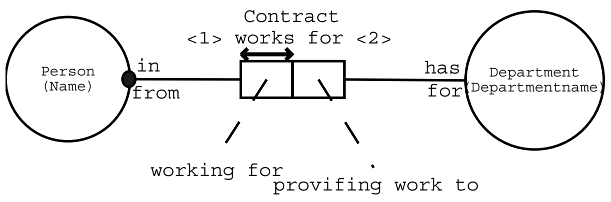


Figure 3.1: Sample schema

The information system checks the schema sentence for being well-formed, and then builds the associated information grammar. From that point the user may address the information system in terms of the newly formed language constructs.

3.2 Stating sentences

The format of stating sentences is based on the formats of the elementary sentences as specified during the analysis phase. In general, a stating sentence has one of the following formats:

- ADD $S_1, S_2, \ldots, S_n$.
- DEL $S_1, S_2, \ldots, S_n$.

where $S_1, \ldots, S_n$ are elementary sentences. Each elementary sentence verbalizes some fact that is to be recorded as a true fact by the information system. These sentences modify the contents of information base as follows:

- If the information base contains population Pop, then the population after

effect addition

ADD S_1, S_2, ..., S_n.

is the minimal extension of population Pop that contains the elementary sentences S_1, ..., S_n. If no such extension exists, then the information system reports an extension error, and does not alter the information base.

effect deletion

- If the information base contains population Pop, then the population after

  DEL S_1, S_2, ..., S_n.

  is the maximal subset of population Pop that does not contain the elementary sentences S_1, ..., S_n. If no such subset exists, then the information system reports an deletion error, and does not alter the information base.

The sentences

S_1, S_2, ..., S_n

are processed as a group. If $n > 1$, then we will refer to this group as a transaction. Transactions allow the user to enter or delete several facts with a single command. However, as we will see in the next chapter, there may be constraints that prohibit elementary facts to be entered outside the context of a transaction. For example, suppose we register both address and birth date from persons. These facts are obviously independent facts, and thus will correspond to different fact types. If we require the information system always to have registered both address and birth date, then there is no way to enter this information via single-sentence additions.

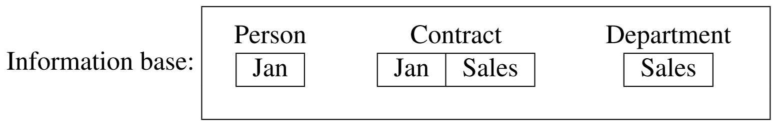
3.2.1 Addition

We demonstrate the extension of the information base as a consequence of additive stating sentences via a sample session. After entering the conceptual schema of figure 3.1, the information system has constructed its information grammar, and its information base contains an empty population of this conceptual schema. As a first fact we enter:

```
> ADD Person with Name ”Jan” works for Department with Name ”Sales”.
Accepted
```

unquote convention

The addition of this fact first requires the introduction of both Person with Name ”Jan” and Department with Name ”Sales”. We will denote the abstract object Person with Name ”Jan” as Jan, and analogously the abstract object Department with Name ”Sales” is denoted as Sales. The resulting population is schematically displayed as:



Next we enter:

```
> ADD Person with Name ”Piet”.
Rejected due to violation of constraint C1: Total role on Contract.in
```

The information system has not accepted Piet to be entered, as this would violate constraint C1 stating that each person should have a contract. Note that the information system can not itself extend the population by guessing a department for Piet. The rejection does leave the population intact. We proceed with:

```
> ADD Department with Name ”PZ”.
```

Accepted

Contrary to the previous case, there is no rule that requires each department to have some person under contract. So there is no problem in the addition of department PZ to the information base. The resulting population:

Information base:

Person	Contract		Department
Jan	Jan	Sales	Sales
			PZ

Then we enter a multi-sentence command:

```
> ADD Person with Name ”Piet” works for Department with Name ”Sales”, Person with Name ”Kees” works
    for Department with Name ”PZ”.
```
Accepted

Both sentences are processed, leading to:

Information base:

Person	Contract		Department
Jan	Jan	Sales	Sales
Piet	Piet	Sales	PZ
Kees	Kees	PZ	

Finally we enter:

```
> ADD Person with Name ”Klaas” works for Department with Name ”Admin”.
```
Accepted

The resulting population is displayed in figure 3.2.

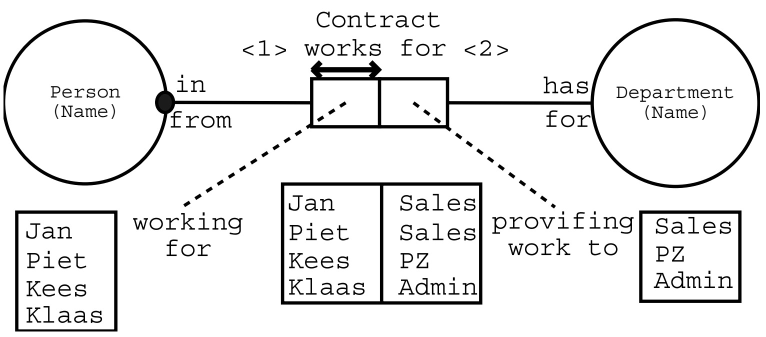


Figure 3.2: The schema populated

3.2.2 Deletion

We demonstrate the deletion of fact types via our sample session. We start with the population from figure 3.2. We start with the removal of Kees:

```
> DEL  Person with Name ”Kees”.
```
Accepted

As a consequence, all information stored about Kees is also to be deleted. This results in the following population:

Information base:

Person	Contract		Department
Jan	Jan	Sales	Sales
Piet	Piet	Sales	PZ

Next we request:

```
> DEL  Person with Name ”Piet” works for Department with Name ”Sales”.
Accepted
```

Removing this fact leaves person Piet without a contract, which would violate constraint C1. This can only be resolved by also deleting person Piet. This results in:

Information base:

Person	Contract		Department
Jan	Jan	Sales	Sales
			PZ

Next we remove the department PZ:

```
> DEL  Department with Departmentname ”PZ”.
Accepted
```

The result is obvious:

Information base:

Person	Contract		Department
Jan	Jan	Sales	Sales

Finally we also remove department Sales. Due to constraint C1, the resulting population is empty. In practice, the information system will be reluctant to carry out such long-range delete cascades. In this case, the information system might want to ask the user if the deletion of person Jan is really intended.

3.3 Asking sentences

The conceptual schema describes the concepts that are valid in the universe of discourse. This schema is derived from the structure of the elementary sentences as provided by the domain expert. Each concept thus has associated a sentence format that can be used to address this concept and its instances. In the previous section this format was used to communicate individual facts to the information system.

In this section we discuss how the elementary sentence formats can be dressed using linguistic constructs to allow the construction of semi-natural asking sentences. This leads to a general format to formulate an information need. This format is called *information descriptor*. The processing of asking sentences contains the following elements:

information descriptor

1. parse the information descriptor
2. evaluate the query
3. present its result

Example 3.3.1

If the user would be interested in the cooperators of the department Sales, then this can be done by reformulating this request in terms of the information grammar, leading to:

```
> LIST Person working for Department "Sales".
    Person with Name "Jan".
    Person with Name "Piet".
    Person with Name "Kees".
    Person with Name "Klaas".
```

Gerund

Information descriptors do not follow directly the structure of elementary sentences, more subtle constructs, derived from the elementary sentence format, are used to get better readable information descriptors. In particular, gerundium constructions will be used as they improve the readability of information descriptors.

3.4 The underlying lexicon: Names and their meaning

The retrieval grammar is constructed in two steps. First we introduce a lexicon, that is derived from the conceptual schema. In this lexicon we can find the meaningful terms and their meaning that have been introduced in the nformation grammar. Secondly we will discuss rules for sentence construction. In this section we will focus on the underlying lexicon.

lexicon

The sentence construction rules provide mechanisms to derive new facts from the information stored in the information base, the population of the conceptual schema. If we see the conceptual schema as a roadmap for connections between the elements of this schema, then a path corresponds to a simple way to combine information from the information encountered on its way. A single path makes connections between instances from one object type with instances of another object type (see figure 3.3).

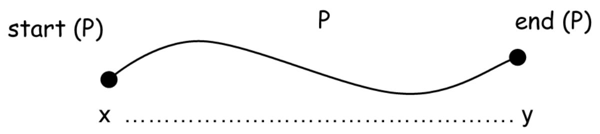


Figure 3.3: A path expression

result table

We will represent the outcome of an information descriptor as a two-column table, containing the starting and ending points of the paths that are associated with that information descriptor. The cells of this table contain the standard names of the abstract objects they refer to. The table is referred to as the result table for that information descriptor.

3.4.1 Object type names and their meaning

Names of object types are elementary ORC expressions. The name of an object type can be used to designate the population of that object type. The name

```
Person
```

refers to (yields) the set of all cooperators that are currently known to the system. The corresponding path consists of the object type named Person. Formally, let A be the name of object type X, then $\mathsf{start}(\mathsf{A}) = X$ and $\mathsf{end}(\mathsf{A}) = X$. We will also say that A has *reach* [X,X]. In the sequel we will use the function $\mathsf{ONm}(X)$ to denote the name of object type X.

reach

If we assume the population as depicted in figure 3.2, then we have the following meaning for these elementary ORC names (using the unquote convention):

Person:

from	upto
Jan	Jan
Piet	Piet
Kees	Kees
Klaas	Klaas

Another example of an elementary path constructed from the name of an object type is the name Contract. The meaning of this information descriptor is an overview of what persons work for what departments. The resulting table will present the facts from its population:

Contract:

from	upto
Jan works for Sales	Jan works for Sales
Piet works for Sales	Piet works for Sales
Kees works for PZ	Kees works for PZ
Klaas works for Admin	Klaas works for Admin

Formally the meaning of an information descriptor D is denoted as $\mathscr{R}(D)$. So we have for example:

$\mathscr{R}(\mathsf{Person}) =$

from	upto
Jan	Jan
Piet	Piet
Kees	Kees
Klaas	Klaas

3.4.2 Role names and their meaning

Roles are the basic mechanisms to build conceptual structures. A role may also be used to select a component from such a constructed object, or to find a compound object matching some component. For this purpose, each role has assigned the following two names:

1. *role name*
2. *inverse role name*

role name

A role name consists of a number of words with lower case letters, separated by spaces. The role name connects the base object of that role to the facts it is involved in. So for example the meaning of role name in is:

$\mathscr{R}(\mathsf{in}) =$

from	upto
Jan	Jan works for Sales
Piet	Piet works for Sales
Kees	Kees works for PZ
Klaas	Klaas works for Admin

So the first tuple in this table relates the object Jan to the fact instance Jan works for Sales. Formally, let r be a role with role name N, then we have:

- $\mathsf{start}(\mathsf{N}) = \mathsf{Base}(r)$
- $\mathsf{end}(\mathsf{N}) = \mathsf{Fact}(r)$.

Note that this information is not very useful on its own. It will only be meaningful in larger constructs. We will see examples later. For the inverse role name we get:

invere role name

$\mathscr{R}(\mathsf{from}) =$

from	upto
Jan works for Sales	Jan
Piet works for Sales	Piet
Kees works for PZ	Kees
Klaas works for Admin	Klaas

Formally, let r be a role with inverse role name N, then we have:

- $\mathsf{start}(\mathsf{N}) = \mathsf{Fact}(r)$
- $\mathsf{end}(\mathsf{N}) = \mathsf{Base}(r)$.

connector name

Besides their role name and inverse role name, roles may have assigned a so-called connector name. The connector name provides a mechanism to refer to fact instances seen from the point of view of some particular object. For example, in the schema of figure **??** we see that connector names

working for

and

providing work to

have been associated to the roles of fact type Contract. Both role names refer to fact instances of fact type Contract. Role names are elementary information descriptors. They associate the participating object types of the fact type as follows. The connector name working for connects those cooperators and departments that are combined via the population of the associated fact type. Note that connector name providing work to connects departments to cooperators.

Generally spoken, the connector name of role p connects the object $\mathsf{Base}(p)$ to the base object type of the other role (the co-role) of a binary fact type. As a consequence, the meaning of connector name works for is the following result table:

$\mathscr{R}(\mathsf{workingfor}) =$

from	**upto**
Jan	Sales
Piet	Sales
Kees	PZ
Klaas	Admin

Connector names are the most frequently used elements in information descriptors. The format of a connector name is obtained from the standard fact name by writing it in a gerundial format. By considering both active and passive voice, both roles can be named. Let r be a role in a binary fact type, then the other role of this fact type is denoted as $\mathsf{CoRole}(r)$. If r has assigned connector name C, then this connector has the following ending points:

co-role

- $\mathsf{start}(\mathsf{C}) = \mathsf{Base}(r)$
- $\mathsf{end}(\mathsf{C}) = \mathsf{Base}(\mathsf{CoRole}(r))$.

In figure 3.4 the various names for role names have been are summarized. For each name the corresponding path through the information structure is shown.

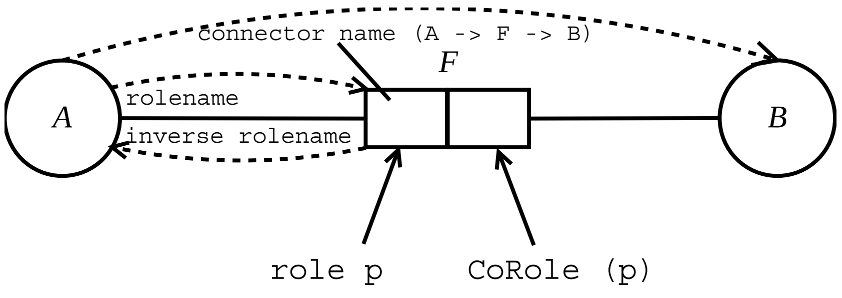


Figure 3.4: Role names

3.4.3 The treatment of constants

In ORC constants are seen as valid information descriptors. The reason for this choice is that the relation between abstract and concrete instances can be made using the concatenation of information descriptors (see next section). As a result, the meaning of a constant is a two-column table. Formally, let c be a constant, then c is an information descriptor with meaning:

$\mathscr{R}(\mathsf{c}) =$

from	**upto**
c	c

Besides names for object types and roles, constants are also described in the lexicon.

3.4.4 A sample lexicon

We consider the conceptual schema from figure 3.1 that is formally depicted in section 3.1. Note that in this figure the bridge types have been explicitly mentioned. Usually we denote the concepts by their name. For the purposes of this section, we have to distinguish between abstract objects and their names. The lexicon is an alphabetically sorted list of names. For each name the meaning is described by providing the beginning and ending of the corresponding path, and how this has to be evaluated.

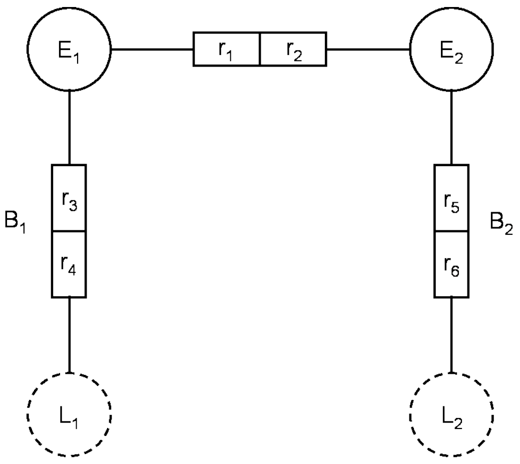


Figure 3.5:

Usually there are no special names associated with the roles form bridge types. In practise it will suffice only to have connector names available. The connector names with and of are a reasonable choice in all cases. As a consequence, these names will occur more than once in the dictionary. The resulting lexicon is displayed in table 3.1.

The multiple occurrence of a name in the lexicon is not forbidden. The interpretation of such a name is simply the union of the individual meanings. Assuming population from figure 3.2, we get:

$\mathscr{R}(\mathsf{with}) =$

from	**upto**
Jan	'Jan'
Piet	'Piet'
Kees	'Kees'
Klaas	'Klaas'
Sales	'Sales'
PZ	'PZ'
Admin	'Admin'

Name	**from**	**upto**	**meaning**
Contract	F	F	$\{\langle \mathbf{fr}: x, \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
Department	E_2	E_2	$\{\langle \mathbf{fr}: x, \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(E_2)\}$
Departmentname	L_2	L_2	$\{\langle \mathbf{fr}: x, \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(L_2)\}$
for	F	E_2	$\{\langle \mathbf{fr}: x, \mathbf{to}: x(r_2)\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
from	F	E_1	$\{\langle \mathbf{fr}: x, \mathbf{to}: x(r_1)\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
has	E_2	F	$\{\langle \mathbf{fr}: x(r_2), \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
in	E_1	F	$\{\langle \mathbf{fr}: x(r_1), \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
Name	L_1	L_1	$\{\langle \mathbf{fr}: x, \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(L_1)\}$
of	L_1	E_1	$\{\langle \mathbf{fr}: x(r_4), \mathbf{to}: x(r_3)\rangle^1 \mid x \in \mathsf{Pop}(B_1)\}$
of	L_2	E_2	$\{\langle \mathbf{fr}: x(r_6), \mathbf{to}: x(r_5)\rangle^1 \mid x \in \mathsf{Pop}(B_2)\}$
Person	E_1	E_1	$\{\langle \mathbf{fr}: x, \mathbf{to}: x\rangle^1 \mid x \in \mathsf{Pop}(E_1)\}$
providing work to	E_2	E_1	$\{\langle \mathbf{fr}: x(r_2), \mathbf{to}: x(r_1)\rangle^1 \mid x \in \mathsf{Pop}(F)\}$
with	E_1	L_1	$\{\langle \mathbf{fr}: x(r_3), \mathbf{to}: x(r_4)\rangle^1 \mid x \in \mathsf{Pop}(B_1)\}$
with	E_2	L_2	$\{\langle \mathbf{fr}: x(r_5), \mathbf{to}: x(r_6)\rangle^1 \mid x \in \mathsf{Pop}(B_2)\}$
working for	E_1	E_2	$\{\langle \mathbf{fr}: x(r_1), \mathbf{to}: x(r_2)\rangle^1 \mid x \in \mathsf{Pop}(F)\}$

Table 3.1: The lexicon for the example

3.4.5 Controlling ambiguity

A consequence of the choice to allow spaces in names associated with roles is end up with an ambiguous language. For example, suppose a lexicon contains the following:

Name	**from**	**upto**	**meaning**
⋮			
for			
⋮			
working			
⋮			
working for			
⋮			

Then the information descriptor working for can be parsed in more than one way. Either we assign meaning to this statement using the entry working for, or we interpret this sentence as the concatenation of working and for, or we simply take the combination of these both meaning interpretations.

In ORC the naming rules state that the resulting grammar may not be ambiguous (as is the case in our example). There are special techniques to efficiently check this grammar property, but these are outside the scope of this course.

3.5 Concatenation of expressions

From elementary information descriptors more complex ones can be constructed. The simplest way to generate new ORC expressions is juxta position: simply concatenating two expressions leads to the new ORC expression. For example, the expression

Person working for Department with Name "Sales"

is the concatenation of the following information descriptors: (1) Person, (2) working for, (3) Department, (4) with, (5) Name and (6) "Sales" (see figure 3.6).

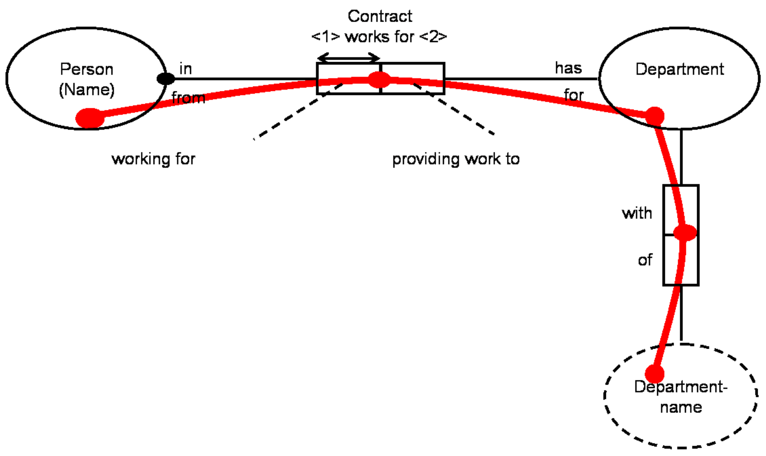


Figure 3.6: Concatenated path

The semantics of the concatenation of two information descriptors is the combination of their respective result tables. The intuition behind is that an information describes a frame for paths between two object instances. The meaning of concatenation (denoted as $\odot$) is to extend the paths from the first descriptor with those from the second one. The result registers for each path that can be constructed this way the beginning and ending points. For example:

from	upto
"jan"	"dutch"
"els"	"german"

$\odot$

from	upto
"dutch"	"teacher1"
"german"	"teacher2"

$=$

from	upto
"jan"	"teacher1"
"els"	"teacher2"

which probably provides an answer to the question what teachers can be assigned to students.

The table concatenation operator $\odot$ gets two result tables as its arguments, and produces a new result table. This operator describes the effect of concatenation of information descriptors. Let D_1 and D_2 be information descriptors, the the meaning of their concatenation $D_1 D_2$ is defined as:

$$\mathscr{R}(D_1\ D_2) = \mathscr{R}(D_1) \odot \mathscr{R}(D_2)$$

Furthermore:

- $\text{start}(D_1\ D_2) = \text{start}(D_1)$
- $\text{end}(D_1\ D_2) = \text{end}(D_2)$.

An important question at this point is: if we concatenate more than two information descriptors, is it then required to use parentheses to enforce the evaluation order. For example, the expression $2+3+4$ may be evaluated either as $(2+3)+4$ or $2-(3+4)$. Due to this independence it is justified to omit the evaluation order. Consequently, the operator $+$ is called *associative*. An example of a non-associative operator is $-$/ for example, we have $(2-3)-4=-5$, while $2-(3-4)=3$. Concatenation of information descriptors is associative. This is a consequence of the associativity of the $\odot$-operator. *associative*

Lemma 3.5.1

$(D_1D_2)D_3 = D_1(D_2D_3)$

Proof:

Two information descriptors are similar if they have the same result independent of the current population. We thus have to show that for each population of the underlying conceptual schema:

$$\mathscr{R}((D_1D_2)D_3) = \mathscr{R}(D_1(D_2D_3))$$

$\Rightarrow$ Assume $\langle \mathbf{fr:}\ a, \mathbf{to:}\ d\rangle \in \mathscr{R}((D_1D_2)D_3)$,

then for some c we have: $\langle \mathbf{fr:}\ a, \mathbf{to:}\ c\rangle \in \mathscr{R}(D_1D_2)$ and $\langle \mathbf{fr:}\ c, \mathbf{to:}\ d\rangle \in \mathscr{R}(D_3)$ (due to the definition of the $\odot$-operator)

and thus for some b we have: $\langle \mathbf{fr:}\ a, \mathbf{to:}\ b\rangle \in \mathscr{R}(D_1)$ and $\langle \mathbf{fr:}\ b, \mathbf{to:}\ c\rangle \in \mathscr{R}(D_2)$ (also by the definition of the $\odot$-operator).

We conclude: $\langle \mathbf{fr:}\ a, \mathbf{to:}\ d\rangle \in \mathscr{R}(D_1(D_2D_3))$.

$\Leftarrow$ Left as an exercise for the reader.

Note that the concatenation of information descriptors is not restricted in any sense. As a consequence, the use of ORC is not hindered when formulating a nonsense expression like

Department working for Person with Name ”Jan”

is probably due to mistakenly using the wrong role name. The problem is illustrated as follows by the following two flaws:

- end(Deparment) $\neq$ start(workingfor).
- end(workingfor) $\neq$ start(Person).

The reason is that instances of the object types named Department and Person are different in any population. Consequently, in *any* population the expression above has an empty result:

$$\mathscr{R}(\text{Department working for Person with Name ”Jan”}) = \varnothing$$

3.5.1 The result table being a multiset

A more complex example of table concatenation is the following:

from	**upto**
”jan”	”dutch”
”els”	”german”

$\odot$

from	**upto**
”dutch”	”teacher1”
”german”	”teacher2”
”german”	”teacher3”
”french”	”teacher4”

=

from	**upto**
”jan”	”teacher1”
”els”	”teacher2”
”els”	”teacher3”

In this example we see that student *els* can have both *teacher2* and *teacher3* as teacher. Mote that each entry in the result table corresponds to a possible connection between beginning point and ending point. Next suppose teacher can teach more than one course. Then this will result in double entries in the result table, as more connections are possible in this case:

from	**upto**
”jan”	”dutch”
”jan”	”german”

$\odot$

from	**upto**
”dutch”	”teacher1”
”german”	”teacher1”

=

from	**upto**
”jan”	”teacher1”
”jan”	”teacher1”

The first entry in the result table connects *jan* to *teacher1* via course *dutch*. The second entry connects them via the course *german*.

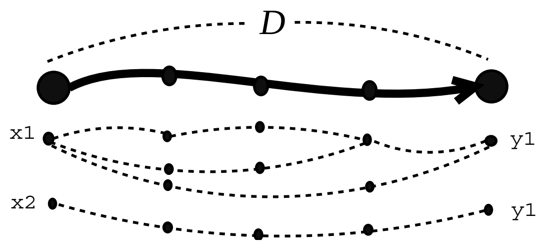


Figure 3.7: Path and population $\{\langle x1, y1\rangle^3, \langle x2, y2\rangle\}$

3.5.2 Intermezzo: multisets

occurrence frequency

Formally, the result of an ORC query is a multiset of tuples. In a multiset, each tuple has associated an *occurrence frequency*. A collection in which elements may occur more than once is called a *bag*. A convenient way is to describe a multiset by a frequency function. Formally, a multiset is a pair (A,m) where:

1. A is some set
2. $m : A \to \mathbb{N}$; $m(a)$ is called the multiplicity of a in multiset (A,m).

A conventional way to denote a multiset as a set of ordered pairs: $\{(a,m(a) \mid a \in A\}$. Some examples:

- the multiset $\{a,b,b\}$ is written as: $\{(a,1),(b,2)\}$,
- $\{a,a,b\}$ is written as $\{(a,2),(b,1)\}$,
- $\{a,b\}$ is written as $\{(a,1),(b,1)\}$.

We will write $x \in^k A$ to explicitly state that x is an element of A with occurrence frequency k.

3.5.2.1 Size of a multiset

The size of a multiset is the number of elements it contains, taking into account the multiplicity of the elements:

$$\#(A,m) = \sum_{a \in A} m(a)$$

So we have $\#\{(a,1),(b,2)\} = 3$ and $\#\{(a,1),(b,1)\} = 2$.

3.5.2.2 Subset relation

We call (B,n) a sub(multi)set of (A,m), denoted as $B,n) \subseteq (A,m)$ if all elements from (B,n) are also available in (A,m) with at least the multiplicity they have within (B,n).

$$B \subseteq A \wedge \forall_{x \in B}\,[n(x) \leq m(x)]$$

We also say that (A,m) is a super(multi)set of (B,n).

Some examples: $\{(a,1),(b,1)\} \subseteq \{(a,1),(b,1)\}$, $\{(a,1),(b,2)\} \subseteq \{(a,1),(b,3),(c,1)\}$.

3.5.2.3 Union of multisets

The union is two multisets is obtained by merging them into a new bag, respecting their multiplicity. For example: $\{(a,1),(b,3),(c,1)\} \cup \{(c,1),(d,3)\} = \{(a,1),(b,3),(c,2),(d,3)\}$. The union of two multisets thus may also be seen as their minimal common supermultiset. This operator is formally defined as:

$$(A,m) \cup (B,n) = (A \cup B, m+n)$$

where the frequency function $m+n : A \cup B \to \mathbb{N}$is defined as:

$$(m+n)(x) = \begin{cases} m(x)+n(x) & x \in A \wedge x \in B \\ m(x) & x \in A \wedge x \notin B \\ n(x) & x \notin A \wedge x \in B \end{cases}$$

3.5.2.4 Intersection of multisets

The intersection of multisets is defined as the maximal common subset of these multisets. For example, $\{(a,1),(b,3),(c,1)\} \cap \{(c,1),(d,3)\} = \{(c,1))\}$. This operator is formally defined as:

$$(A,m) \cap (B,n) = (A \cap B, \min(m,n))$$

where the frequency function $\min(m,n) : A \cap B \to \mathbb{N}$is defined as:

$$\min(m,n)(x) = \min(m(x),n(x))$$

3.5.2.5 Difference of multisets

The difference of multisets is obtained by removing from a multiset the elements that also occur in a second multiset. For example, $\{(a,1),(b,3),(c,1)\} - \{(b,1),(c,2),(d,3)\} = \{(a,1),(b,2)\}$. This operator is formally defined as:

$$(A,m) - (B,n) = (A \cap B, \max(m-n,0)$$

3.5.2.6 Cartesian product of multisets

The cartesian product of two multisets is defined as:

$$(A,m) \times (B,n) = (A \times B, m \cdot n)$$

where the frequency function $m \cdot n : A \times B \to \mathbb{N}$is defined as:

$$(m \cdot n)(x,y) = m(x) \cdot n(y)$$

3.6 Special constructs

ORC contains a number of special constructs to combine information descriptors into more complex descriptors.

3.6.1 Correlation operator

A very powerful mechanism from natural language is the possibility to use indicatives. Words like 'this', 'those', etc. are used to refer to some other construct in the same sentence, or even in another sentence. An example is the statement: if a person lives in a country, then this person will speak the languages being spoken in *that* country. Another example: an interpreter speaks the language of some country but also *another* language. Indicatives provide a powerful mechanism to improve the effectiveness of communication, but can also be a source of misinterpretation. An indicative for example may be interpreted differently by different persons, but an indicative could also become a dangling reference after the addition of a new sentence to the text.

In ORC indicatives are introduced in a very restricted way to cover for those cases where a sentence can not be expressed without the usage of indicatives. This special usage is referred to as the correlation operator. We will introduce this operator in both a positive (*that*) and a negative (*another*) way.[1]

The correlation operator couples instances from the start and end of the information descriptor. Consequently, this operator does only make sense for information descriptors that have the same beginning and ending point.

For example, in figure 3.8 we register the elementary facts about persons, their native country, the language(s) spoken in those languages, and the language(s) spoken by the registered people. An obvious question in this universe of discourse would be: *who are native speakers*? In terms of the base facts, this sentence may be reformulated in two steps as follows. First we handle the part about speaking the appropriate language: *which persons speak a language that is being used in their birth country.*

Specifying the birth country requires an indication that the country is the native country of the person we mentioned earlier in the information descriptor. This is formulated as: *which persons speak a language that is being used in the country being the birth country of that particular person.*

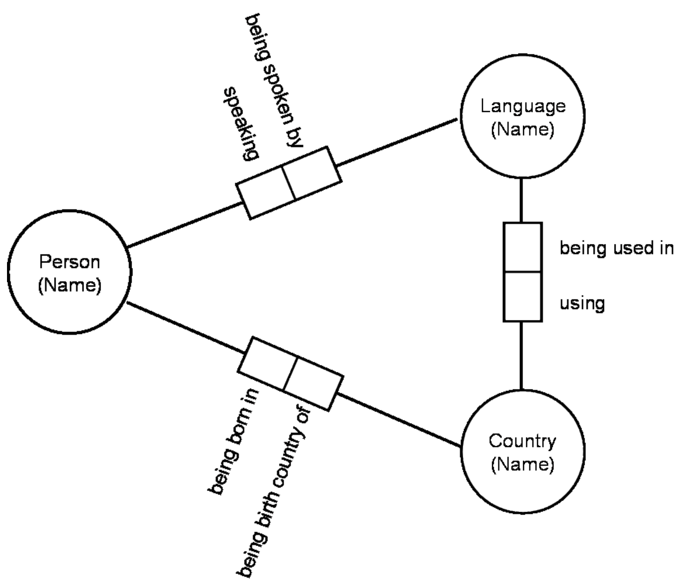


Figure 3.8: A simple interpreter administration

Next we modify the formulation via a pure syntactic transformation into the more strict format from ORC.

Person speaking Language being used in Country being birth country of THAT Person

[1]A more general usage of indicators would require a mechanism to explicitly refer to other elements in a sentence. This may be solved by the introduction of variable. Variables fall outside the scope of this course

This information descriptor describes a path that relates persons to persons. The qualification THAT Person restricts the those cases where a persons are related to themselves. Before describing the semantics formally, we derive the meaning of this sentence intuitively. For the evaluation of this expression we assume the following population:

Speaks

Jan	Flemisch
Els	German
Marie	French

Used

Flemisch	Belgium
German	Germany
French	Belgium

Birth

Jan	Belgium
Else	France
Marie	Belgium

The information descriptor Person speaking Language being used in Country being birth country of evaluates to:

$\mathscr{R}$(Person speaking Language being used in Country being birth country of)

=

from	**upto**
Jan	Jan
Jan	Marie
Marie	Jan
Marie	Marie

The qualification THAT Person restricts this table to the the reflexive tuples:

$\mathscr{R}$(Person speaking language being used in Country being birth country of THAT Person)

= $\mathscr{R}$(Person speaking language being used in Country being birth country of) $\cap$ $\mathscr{R}$(Person)

=

from	**upto**
Jan	Jan
Jan	Marie
Marie	Jan
Marie	Marie

$\cap$

from	**upto**
Jan	Jan
Marie	Marie

=

from	**upto**
Jan	Jan
Marie	Marie

Formally, let D be an ORC information descriptor with start(D) $=$ end(D) $= X$ for some object type X. Then the following sentences are also valid ORC information descriptors:

1. *positive correlation*: D THAT ONm(X), with the following meaning:

$$\mathscr{R}(\mathsf{D\ THAT\ ONm(X)}) = \left\{ \langle \textbf{fr:}\, x, \textbf{to:}\, y \rangle^n \,\middle|\, \langle \textbf{fr:}\, x, \textbf{to:}\, y \rangle \in^n \mathscr{R}(\mathsf{D}) \wedge x = y \right\}$$

2. *negative correlation*: D ANOTHER ONm(X), with the following meaning:

$$\mathscr{R}(\mathsf{D\ ANOTHER\ ONm(X)}) = \left\{ \langle \textbf{fr:}\, x, \textbf{to:}\, y \rangle^n \,\middle|\, \langle \textbf{fr:}\, x, \textbf{to:}\, y \rangle \in^n \mathscr{R}(\mathsf{D}) \wedge x \neq y \right\}$$

Next we consider an example of negative correlation. Suppose we are interested in the languages for which there is an interpreter. We reformulate this sentence using the basic sentences from our conceptual schema (figure 3.8). A first formulation is: *which languages are being spoken by a person that also speaks another language*. We refine this a step further into: *which languages are being spoken by a person speaking another language*. The negative correlation is expressed in ORC as:

Language being spoken by Person speaking ANOTHER Language

3.6.2 Set-like operators

Information descriptors can be combined into new information descriptors by a number of operators. The first group contains operators such as union, intersection and set difference. These operators come in two variants. The first variant operates on the left columns of its arguments, the second variant processes both columns.

3.6.2.1 Set-like front operators

The front versions of the set operators are obtained by performing the set operator on the left columns of its arguments. The result should however be transformed into a result table. The first column of a result table T is obtained by $\pi_1 T$ by applying the function π_1. For example:

$$\pi_1 \begin{array}{|l|l|}\hline \textbf{from} & \textbf{upto} \\ \hline \text{Jan} & \text{Jan} \\ \text{Jan} & \text{Marie} \\ \text{Marie} & \text{Jan} \\ \text{Marie} & \text{Marie} \\ \hline \end{array} = \{\text{Jan, Jan, Marie, Marie}\}$$

The transformation of a multiset into a result table is performed by the so-called square operator:

$$\{\text{Jan, Jan, Marie, Marie}\}^2 = \begin{array}{|l|l|}\hline \textbf{from} & \textbf{upto} \\ \hline \text{Jan} & \text{Jan} \\ \text{Jan} & \text{Jan} \\ \text{Marie} & \text{Marie} \\ \text{Marie} & \text{Marie} \\ \hline \end{array}$$

Using these conventions, the front versions of the set operators are introduced as:

expression	$\mathscr{R}$(expression)
P AND ALSO Q	$[\pi_1\mathscr{R}(P) \cap \pi_1\mathscr{R}(Q)]^2$
P OR OTHERWISE Q	$[\pi_1\mathscr{R}(P) \cup \pi_1\mathscr{R}(Q)]^2$
P BUT NOT Q	$[\pi_1\mathscr{R}(P) \setminus \pi_1\mathscr{R}(Q)]^2$

As an example, we consider the information descriptor

Person working for Department with Departmentname ”Sales” OR working for Department with Departmentname ”Admin”

Then we have:

$\mathscr{R}$(Person working for Department with Departmentname ”Sales”) =

from	**upto**
Jan	”Sales”
Piet	”Sales”

and

$\mathscr{R}$(working for Department with Departmentname ”Admin”) =

from	**upto**
Klaas	”Admin”

Combining these results we get:

$$\left[\pi_1 \begin{array}{|l|l|}\hline \textbf{from} & \textbf{upto}\\\hline \text{Jan} & \text{"Sales"}\\ \text{Piet} & \text{"Sales"}\\\hline\end{array} \cup \pi_1 \begin{array}{|l|l|}\hline \textbf{from} & \textbf{upto}\\\hline \text{Klaas} & \text{"Admin"}\\\hline\end{array}\right]^2$$

$$= \left[\{\mathsf{Jan},\mathsf{Piet}\} \cup \{\mathsf{Klaas}\}\right]^2$$

$$= \left[\{\mathsf{Jan},\mathsf{Piet},\mathsf{Klaas}\}\right]^2$$

$$= \begin{array}{|l|l|}\hline \textbf{from} & \textbf{upto}\\\hline \text{Jan} & \text{Jan}\\ \text{Piet} & \text{Piet}\\ \text{Klaas} & \text{Klaas}\\\hline\end{array}$$

3.6.2.2 Set-like head-tail operators

The head-tail versions of the elementary set operators are defined as:

expression	$\mathscr{R}$(expression)
P INTERSECTION Q	$\mathscr{R}(P)\cap\mathscr{R}(Q)$
P UNION Q	$\mathscr{R}(P)\cup\mathscr{R}(Q)$
P MINUS Q	$\mathscr{R}(P)\setminus\mathscr{R}(Q)$

3.6.3 THE-operator

Information descriptors relate beginning and ending points of paths through the information structure. For example, the information descriptor

> Person working for Department with Departmentname "Sales" OR working for Department with Departmentname "Admin"

evaluates to the following result tabel:

$\mathscr{R}$(Person working for Department with Departmentname "Sales") =

from	upto
Jan	"Sales"
Piet	"Sales"

If we want to restrict this result to its first column, then we use the keyword THE:

> THE Person working for Department with Departmentname "Sales" OR working for Department with Departmentname "Admin"

The result is:

$\mathscr{R}$(THE Person working for Department with Departmentname "Sales") =

from	upto
Jan	Jan
Piet	Piet

The operator THE restricts an information descriptor to its beginning point:

expression	$\mathscr{R}$(expression)
THE P	$[\pi_1\mathscr{R}(\mathsf{D})]^2$

3.6.4 DISTINCT operator

3.6.5 Arithmetic operators

Binary operators on concrete domains may be used to combine the result of information descriptors. For example, if the underlying domain is numeric, then operators like +, -, *, / are used to combine values arithmetically. We will discuss two formats for arithmetic operators.

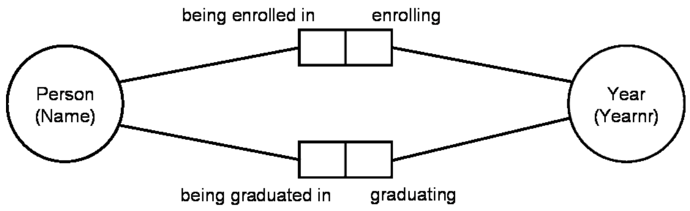


Figure 3.9: Small training administration

The first format is required for the following information need: (see figure 3.9): select the students that finished their training within nominal time. In words: *the students that are being graduated not later than 3 years after their enrollment*. An example of the second format is the computation of the period that persons have used for their training. We will discuss these formats separately. After that, we will discuss comparison operators.

3.6.5.1 Front version of arithmetic operators

We further elaborate on the ORC formulation of *the students that are being graduated not later than 3 years after their enrollment*. We first consider the subexpression *3 years after enrollment*. In terms of the conceptual schema, this may be formulated as *3 years after Yearno of Year enrolling*. The idea is that the year numbers of enrollment are to be augmented with 3. If for example Erna has been enrolled in 2003, then $\langle \textbf{fr: } 2008, \textbf{to: } Erna\rangle$ will be a tuple in the result table of this information descriptor. This particular use of the addition operator is described by the PLUS-operator, leading for this subexpression to the information descriptor:

3 PLUS Yearno of Year enrolling

The rest of the information descriptor will be discussed in section 3.6.5.3.

Formally, let OP be one of the operators PLUS, MINUS, TIMES, DIVIDED BY, then the information descriptor D_1 OP D_2 requires $\mathsf{start}(D_1)$ and $\mathsf{start}(D_2)$ to be compatible label types. The resulting expression has the same reach as D_2, and is determined by:

expression	$\mathscr{R}$(expression)
D_1 PLUS D_2	$\left\{\langle \textbf{fr: } c+x, \textbf{to: } y\rangle^k \mid c \in \pi_1\mathscr{R}(\mathsf{D}_1) \wedge \langle \textbf{fr: } x, \textbf{to: } y\rangle \in^k \mathscr{R}(\mathsf{D}_2)\right\}$
D_1 MINUS D_2	$\left\{\langle \textbf{fr: } c-x, \textbf{to: } y\rangle^k \mid c \in \pi_1\mathscr{R}(\mathsf{D}_1) \wedge \langle \textbf{fr: } x, \textbf{to: } y\rangle \in^k \mathscr{R}(\mathsf{D}_2)\right\}$
D_1 TIMES D_2	$\left\{\langle \textbf{fr: } c*x, \textbf{to: } y\rangle^k \mid c \in \pi_1\mathscr{R}(\mathsf{D}_1) \wedge \langle \textbf{fr: } x, \textbf{to: } y\rangle \in^k \mathscr{R}(\mathsf{D}_2)\right\}$
D_1 DIVIDED BY D_2	$\left\{\langle \textbf{fr: } c/x, \textbf{to: } y\rangle^k \mid c \in \pi_1\mathscr{R}(\mathsf{D}_1) \wedge \langle \textbf{fr: } x, \textbf{to: } y\rangle \in^k \mathscr{R}(\mathsf{D}_2)\right\}$

3.6.5.2 Head-tail version of arithmetic operators

Another example is the computation of the period that persons have used for their training. This period is obtained by subtracting the year of enrollment from the graduation year. The result table should relate this outcome with the corresponding student. This is brought about by:

Yearno of Year graduating Person - Yearno enrolling Person

Formally, let OP be one of the operators +, -, *, /, then the information descriptor D_1 OP D_2 requires (1) $\mathsf{start}(\mathsf{D}_1)$ and $\mathsf{start}(\mathsf{D}_2)$ to be compatible label types, and (2) $\mathsf{end}(\mathsf{D}_1)$ and $\mathsf{end}(\mathsf{D}_2)$ to be type-related. The resulting expression has the same reach as D_1 and D_2, an is determined by:

expression	$\mathscr{R}$(expression)
$\mathsf{D}_1 + \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ c+d, \mathbf{to:}\ y\rangle \mid \langle \mathbf{fr:}\ c, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2)\}$
$\mathsf{D}_1 - \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ c-d, \mathbf{to:}\ y\rangle \mid \langle \mathbf{fr:}\ c, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2)\}$
$\mathsf{D}_1 * \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ c*d, \mathbf{to:}\ y\rangle \mid \langle \mathbf{fr:}\ c, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2)\}$
$\mathsf{D}_1 / \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ c/d, \mathbf{to:}\ y\rangle \mid \langle \mathbf{fr:}\ c, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2)\}$

3.6.5.3 Comparison operators

We proceed with the example *the students that are being graduated not later than 3 years after their enrollment*. This information need is formulated as the following information descriptor:

Person being graduated in Year having Yearno $\leq$ 3 PLUS Yearno of Year enrolling THAT Person

This evaluation order of this information descriptor is displayed in figure **??**. This evaluation order may be

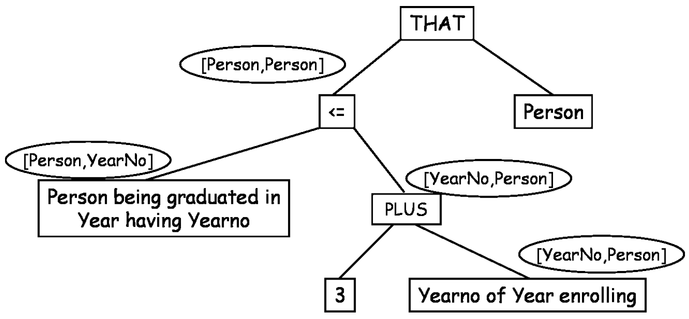


Figure 3.10: Evaluation order

obtained by applying the priority rule for ORC operators (see section 3.9).

Formally, let RELOP be one of the operators $\leq$, $<$, $\geq$, $>$, then the information descriptor D_1 RELOP D_2 requires (1) $\mathsf{end}(\mathsf{D}_1)$ and $\mathsf{start}(\mathsf{D}_2)$ to be compatible label types. The resulting expression has reach as $[\mathsf{start}(\mathsf{D}_1), \mathsf{end}(\mathsf{D}_2)]$ and is determined by:

expression	$\mathscr{R}$(expression)
$\mathsf{D}_1 \leq \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ x, \mathbf{to:}\ y\rangle \mid \exists_{c,d}\left[\langle \mathbf{fr:}\ x, \mathbf{to:}\ c\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2) \wedge c \leq d\right]\}$
$\mathsf{D}_1 < \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ x, \mathbf{to:}\ y\rangle \mid \exists_{c,d}\left[\langle \mathbf{fr:}\ x, \mathbf{to:}\ c\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2) \wedge c < d\right]\}$
$\mathsf{D}_1 \geq \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ x, \mathbf{to:}\ y\rangle \mid \exists_{c,d}\left[\langle \mathbf{fr:}\ x, \mathbf{to:}\ c\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2) \wedge c \geq d\right]\}$
$\mathsf{D}_1 > \mathsf{D}_2$	$\{\langle \mathbf{fr:}\ x, \mathbf{to:}\ y\rangle \mid \exists_{c,d}\left[\langle \mathbf{fr:}\ x, \mathbf{to:}\ c\rangle \in \mathscr{R}(\mathsf{D}_1) \wedge \langle \mathbf{fr:}\ d, \mathbf{to:}\ y\rangle \in \mathscr{R}(\mathsf{D}_2) \wedge c > d\right]\}$

3.7 Statistical Operators

Statistical operators can be used to perform statistical operations on the computational result of an information descriptor. There are two variants. The head operators directly process on the resulting multiset, while the head-tail versions first perform a grouping on this multiset, and then computing te statistical function groupwise.

3.7.1 Head Operators

3.7.1.1 Counting

The number of elements from the multiset that results from the evaluation of an information descriptor can be obtained using the COUNT operator. For example, COUNT Person provides the number of persons. Note that this number (like the handling of constants in ORC) is given as a single-row table. The semantics are formally expressed as:

$$\mathscr{R}(\text{COUNT D}) = \{\langle \textbf{fr: } \#\mathscr{R}(\text{D}), \textbf{to: } \#\mathscr{R}(\text{D}) \rangle\}$$

For example,

$\mathscr{R}(\text{COUNT with}) =$

from	**upto**
7	7

while

$\mathscr{R}(\text{COUNT Department working for Person with Name "Jan"}) =$

from	**upto**
0	0

3.7.2 Head-Tail Operators

A more subtle method of counting first groups the multiset resulting from the evaluation of an information descriptor according to the first column, and then

$$\mathscr{R}(\text{NUMBER } D) = \left\{ \langle \textbf{fr: } n, \textbf{to: } y \rangle^n \;\middle|\; n = \sum \left\{ k \;\middle|\; \langle \textbf{fr: } x, \textbf{to: } y \rangle \in^k \mathscr{R}(\text{D}) \right\} \right\}$$

$\mathscr{R}(\text{NUMBER Person being born in Country using Language})$:

from	**upto**
Jan	Flemish
Jan	French
Marie	Flemish
Marie	French

3.7.3 Head-tail version of comparison operators

$$\mathscr{R}(D_1 \text{ RELOP}_2\ D_2) = \left\{ \langle \textbf{fr: } x, \textbf{to: } n \rangle^k \;\middle|\; \langle \textbf{fr: } x, \textbf{to: } n \rangle^k \in \mathscr{R}(D_1) \wedge \forall_{\langle \textbf{fr: } k, \textbf{to: } y \rangle \in \mathscr{R}(D_2)} [n \text{ RELOP } k] \right\}$$

3.8 Specifying conditions

The language ORC provides the information descriptor as a mechanism to describe a set of information derived from the current population. In this section we will use information descriptors to describe a constraint.

3.8.1 Boolean constants

The basis for predicates is formed by the boolean values, which are introduced as special zero-adic operators:

expression	$\mathscr{R}$(expression)
FALSE	$\varnothing$

As a consequence, if an information descriptor is used in a boolean context, and the descriptor evaluates to a non-empty result table, then this descriptor has a non-false value, and thus a true-value.

3.8.2 Quantification

A information descriptor is coerced into a condition by the NO-operator. For example, in figure 3.11 we want to express the rule that a person may not work in a project and also be the manager of that project. The information descriptor Cooperator is managing Project AND working for Project describes all cooperators that break this rule. The following rule states that no cooperator is allowed to this infringement:

NO Cooperator is managing Project AND working for Project

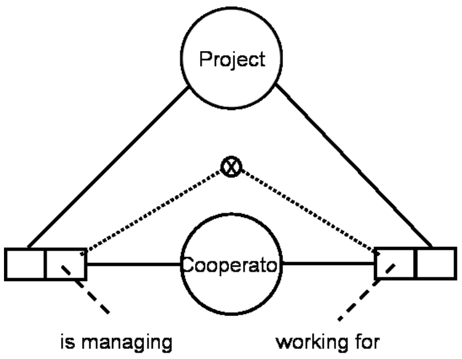


Figure 3.11: Work management

Formally this is expressed as:

expression	$\mathscr{R}$(expression)
NO D	$\begin{cases} \mathscr{R}(\text{TRUE}) & \text{if } \mathscr{R}(D) = \varnothing \\ \mathscr{R}(\text{FALSE}) & \text{otherwise} \end{cases}$

3.8.3 Implication

As an example we consider the so-called *conformity rule*. This rule states for example (see figure 3.1) that for each cooperator-department tuple in the population of fact type *Contract* the cooperator should be a member of the population of object type *Person*. This rule can be expressed in ORC as follows:

Cooperator working for Department IS ALSO Cooperator

The ORC operator IS ALSO tests the subset relation on the first columns of both result tables. As entries may occur more than once in these columns, the subset relation requires:

for each instance the number of occurrences in 1st column of the left table $\leq$ the number of occurrences of that instance in 1st column of the right table

For the rule above we get:

$\mathscr{R}$(Cooperator working for Department) =

from	upto
Jan	Sales
Piet	Sales
Kees	PZ
Klaas	Admin

The first column entries are obtained by:

π_1

from	upto
Jan	Sales
Piet	Sales
Kees	PZ
Klaas	Admin

$= \{$Jan, Piet, Kees, Klaas$\}$

The evaluation of the righthand side leads to:

$$\pi_1\mathscr{R}(\text{Cooperator}) = \{\text{Jan, Piet, Kees, Klaas}\}$$

The keyword IS ALSO compares the instances of the first entry of the resulting table. The operator IMPLIES is a tuple-wise comparison. An example is the condition *all persons speak all national languages from their mother country*. Using the IMPLIES construct, this sentence can be formulated, in terms of the conceptual schema from figure 3.8, as follows:

Person being born in Country using Language IMPLIES Person speaking Language

The lefthand side evaluates to:

$\mathscr{R}$(Person being born in Country using Language):

from	upto
Jan	Flemish
Jan	French
Marie	Flemish
Marie	French

The righthand side yields:

$\mathscr{R}$(Person speaking Language):

from	upto
Jan	Flemish
Els	German
Marie	French

The ORC condition requires:

from	upto
Jan	Flemish
Jan	French
Marie	Flemish
Marie	French

$\subseteq$

from	upto
Jan	Flemish
Els	German
Marie	French

which obviously is not the case as Jan does not speak French and Marie fails on Flemish. Formally these constructs are defined as:

expression	$\mathscr{R}$(expression)
D_1 IS ALSO D_2	$\mathscr{R}$(THE D_1 IMPLIES THE D_2)
D_1 IMPLIES D_2	$\begin{cases} \mathscr{R}(\text{TRUE}) & \text{if } \mathscr{R}(D_1) \subseteq \mathscr{R}(D_2) \\ \mathscr{R}(\text{FALSE}) & \text{otherwise} \end{cases}$

Lemma 3.8.1 *Negative-positive formulation transformation*
Let X be an object type with name N, and D an information descriptor from N to N (having reach $[\mathsf{N},\mathsf{N}]$), then the information descriptor

NO N D ANOTHER N

is equivalent with

N D IMPLIES N

Proof:

$\rightarrow$ Suppose NO N D ANOTHER N, then $\mathscr{R}(\text{N D ANOTHER N})$ is empty in each population of the underlying conceptual schema. So any tuple from $\mathscr{R}(\text{N D})$ should connect equal instances from X, and thus this tuple is also element of $\mathscr{R}(\text{N})$.

$\leftarrow$ Suppose N D IMPLIES N, then no tuple resulting from N D can connect different instances from N, and thus NO N D ANOTHER N.

3.9 Priorities

Priorities have been introduced to disambiguate expressions. For example, the expression $2+3*5$ is to be evaluated as $2+(3*5)$ as the operator $*$ has a higher priority as the operator $+$. Thanks to the priority mechanism, parentheses can be omitted in many cases in expressions.

For example, the $+$-operator and the $-$-operator have the same priority. As the $+$-operator is associative, the expression $2+3+4$ may be evaluated as $2+(3+4)$ or as $(2+3)+4$. The $-$-operator is not associative, for the evaluation of $2-3-4$ the evaluation order has to be specified. IN order to avoid parenthese also in this case, the convention is that the evaluation order is *left associative*, meaning that the expression $2-3-4$ is to be evaluated from left to right: $(2-3)-4$.

In ORC the same techniques are used. Consider for example the information descriptor

NO Cooperator is managing Project AND working for THAT Project

The evaluation order of this expression is:

NO (((Cooperator is managing Project) AND ALSO (working for)) THAT Project)

This evaluation order is obtained from the following priority convention:

	operator
high	concatenation
	*, /, TIMES, DIVIDED BY
	+, -, PLUS, MINUS
	`<, <=, >, >=, =, <>`
	AND ALSO, OR OTHERWISE, BUT NOT, $\cup$, $\cap$, $-$
	THAT, ANOTHER
	IS ALSO, IMPLIES
low	NO

3.10 ORC overview

3.10.1 Constructs

construct	meaning
D_1 D_2	$\mathscr{R}(D_1) \odot \mathscr{R}(D_2)$
THE D	Restriction to head
DISTINCT D	Duplicate removal
D_1 INTERSECTION D_2	$\mathscr{R}(D_1) \cap \mathscr{R}(D_2)$
D_1 UNION D_2	$\mathscr{R}(D_1) \cup \mathscr{R}(D_2)$
D_1 MINUS D_2	$\mathscr{R}(D_1) - \mathscr{R}(D_2)$
D_1 AND {ALSO} D_2	(THE D_1) INTERSECTION (THE D_2)
D_1 OR {OTHERWISE} D_2	(THE D_1) UNION (THE D_2)
D_1 BUT NOT D_2	(THE D_1) MINUS (THE D_2)
D_1 THAT D_2	D_1 INTERSECTION THE D_2
D_1 ANOTHER D_2	(D_1 MINUS (THE D_1)) THAT D_2
*, /, +, -	concatenating arithmetic operators
TIMES, DIVIDED BY, PLUS, MINUS	head-tail arithmetic operators
$<, \leq, >, \geq, =, \neq$	concatenating comparison operators
$<_2, \leq_2, >_2, \geq_2$	head-tail comparison operators
COUNT, SUM, AVERAGE	Overall variant
.. GROUPWISE	Grouping variant
D_1 IMPLIES D_2	$\big(\mathscr{R}(D_1) \subseteq \mathscr{R}(D_2)\big)?$
D_1 IS ALSO D_2	(THE D_1) IMPLIES (THE D_2)
NO D	$\big(\mathscr{R}(D) = \varnothing\big)?$

3.10.2 Priorities

	operator(s)
high	concatenation
	*, /, TIMES, DIVIDED BY
	+, -, PLUS, MINUS
	$<, \leq, >, \geq, =, \neq,$ $<_2, \leq_2, >_2, \geq_2, \neq_2$
	AND {ALSO}, OR {OTHERWISE}, BUT NOT, $\cup$, $\cap$, $-$
	THAT, ANOTHER
	THE, DISTINCT
	COUNT, SUM, AVERAGE, GROUPWISE variants
	IS ALSO, IMPLIES
low	NO

3.11 An extended sample session: the presidential database

In this section we introduce ORC by means of an extended sample session. This session is base on the so-called presidential database has been used in literature as a base for comparison between modeling techniques. We start from the following sample tables. Table 3.2 contains the administrations, their presidents and their vice-presidents. Relevant information about presidents of the USA is being recorded in table 3.3.

Personal information about presidents is stored in table 3.4 Hobbies of presidents are described in table 3.5. The presidential contribution of states is described in table 3.6. This leads to the conceptual schema from figure 3.12.

3.11.1 Elementary sentences and juxta-position

We start with a number of examples showing the usage of the lexicon and the how simple sentences can be formed by juxta-position.

1. *Show all presidents*
   The object type *President* is populated by all presidents. To list all presidents, we use:

   ```
   LIST President
   ```

   The LIST command will display the resulting presidents by their standard names.

2. *Who are the president spouses?*
   This is expressed as the concatenation of elementary names from the conceptual schema:

   ```
   LIST Person being spouse of President
   ```

Administration table		
Administration number	President name	Vice president name
1	Washington G	Adams J
2	Washington G	Adams J
3	Adams J	Jefferson T
4	Jefferson T	Burr A
5	Jefferson T	Clinton G
6	Madison J	Clinton G
7	Madison J	Gerry E
8	Monroe J	Tompkins D
9	Monroe J	Tompkins D
10	Adams J Q	Calhoun J
11	Jackson A	Calhoun J
12	Jackson A	Van Buren M
13	Van Buren M	Johnson R M
14	Harrison W H	Tyler J
15	Polk J K	Dallas G M
16	Taylor Z	Fillmore M
17	Pierce F	De Vane King
18	Buchanan J	Breckinridge
19	Lincoln A	Hamlin H
20	Lincoln A	Johnson A
21	Grant U S	Colfax S
22	Grant U S	Wilson H
23	Hayes R B	Wheeler W
24	Garfield J A	Arthur C A
25	Cleveland G	Hendricks T A
26	Harrison B	Morton L P
27	Cleveland G	Stevenson A E
28	McKinley W	Hobart G A
29	McKinley W	Roosevelt T
30	Roosevelt T	Fairbanks C W
31	Taft W H	Sherman J S
32	Wilson W	Marshall T R
33	Wilson W	Marshall T R
34	Harding W G	Coolidge C
35	Coolidge C	Dawes C G
36	Hoover H C	Curtis C
37	Roosevelt F D	Garner J N
38	Roosevelt F D	Garner J N
39	Roosevelt F D	Wallace H A
40	Roosevelt F D	Truman H S
41	Truman H S	Barkley A W
42	Eisenhower D D	Nixon R M
43	Eisenhower D D	Nixon R M
44	Kennedy J F	Johnson L B
45	Johnson L B	Humphrey H H
46	Nixon R M	Agnew S T
47	Nixon R M	Agnew S T
47	Nixon R M	Ford G R
47	Ford G R	Rockefeller N
48	Carter J E	Mondale W F
49	Reagan R	Bush G

Table 3.2: Administration table

Presidential information					
President name	Birth year	Years served	Death age	Party	State born
Washington G	1732	7	67	Federalist	Virginia
Adams J	1735	4	90	Federalist	Massachusetts
Jefferson T	1743	8	83	Demo-Rep	Virginia
Madison J	1751	8	85	Demo-Rep	Virginia
Monroe J	1758	8	73	Demo-Rep	Virginia
Adams J Q	1767	4	80	Demo-Rep	Massachusetts
Jackson A	1767	8	78	Democratic	South Carolina
Van Buren M	1782	4	79	Democratic	New York
Harrison W H	1773	0	68	Whig	Virginia
Tyler J	1790	3	71	Whig	Virginia
Polk J K	1795	4	53	Democratic	North Carolina
Taylor Z	1784	1	65	Whig	Virginia
Fillmore M	1800	2	74	Whig	New York
Pierce F	1804	4	64	Democratic	New Hampshire
Buchanan J	1791	4	77	Democratic	Pennsylvania
Lincoln A	1809	4	56	Republican	Kentucky
Johnson A	1808	3	66	Democratic	North Carolina
Grant U S	1822	8	63	Republican	Ohio
Hayes R B	1822	4	70	Republican	Ohio
Garfield J A	1831	0	49	Republican	Ohio
Arthur C A	1830	3	56	Republican	Vermont
Cleveland G	1837	8	71	Democratic	New Jersey
Harrison B	1833	4	67	Republican	Ohio
McKinley W	1843	4	58	Republican	Ohio
Roosevelt T	1858	7	60	Republican	New York
Taft W H	1857	4	72	Republican	Ohio
Wilson W	1856	8	67	Democratic	Virginia
Harding W G	1865	2	57	Republican	Ohio
Coolidge C	1872	5	60	Republican	Vermont
Hoover H C	1874	4	90	Republican	Iowa
Roosevelt F D	1882	12	63	Democratic	New York
Truman H S	1884	7	88	Democratic	Missouri
Eisenhower D D	1890	8	79	Republican	Texas
Kennedy J F	1917	2	46	Democratic	Massachusetts
Johnson L B	1908	5	65	Democratic	Texas
Nixon R M	1913	5	?	Republican	California
Ford G R	1913	2	?	Republican	Nebraska
Carter J E	1924	4	?	Democratic	Georgia
Reagan R	1911	3	?	Republican	Illinois

Table 3.3: Presidential information

Personal information					
President name	Spouse name	President age	Spouse age	Number of children	Marriage year
Washington G	Custis M D	26	27	0	1759
Adams J	Smith A	28	19	5	1764
Jefferson T	Skelton M W	28	23	6	1772
Madison J	Todd D D P	43	26	0	1794
Monroe J	Kortright E	27	17	3	1786
Adams J Q	Johnson L C	30	22	4	1797
Jackson A	Robards R D	26	26	0	1794
Van Buren M	Hoes H	24	23	4	1807
Harrison W H	Symmes A T	22	20	10	1795
Tyler J	Christian L	23	22	8	1813
Tyler J	Gardiner J	54	24	7	1844
Polk J K	Childress S	28	20	0	1824
Taylor Z	Smith M M	25	21	6	1810
Fillmore M	Powers A	26	27	2	1826
Fillmore M	McIntosh C C	58	44	0	1858
Pierce F	Appleton J M	29	28	3	1834
Lincoln A	Todd M	33	23	4	1842
Johnson A	McCardle E	18	16	5	1827
Grant U S	Dent J B	26	22	4	1848
Hayes R B	Webb L W	30	21	8	1852
Garfield J A	Rudolph L	26	26	7	1858
Arthur C A	Herndon E L	29	22	3	1859
Cleveland G	Folson F	49	21	5	1886
Harrison B	Scott C L	20	21	2	1853
Harrison B	Dimmick M S L	62	37	1	1896
McKinley W	Saxton I	27	23	2	1871
Roosevelt T	Lee A H	22	19	1	1880
Roosevelt T	Carow E K	28	25	5	1886
Taft W H	Herron H	28	25	3	1886
Wilson W	Axson E L	28	25	3	1885
Wilson W	Galt E B	58	43	0	1915
Harding W G	De Wolfe F K	25	30	0	1891
Coolidge C	Goodhue G A	33	26	2	1905
Hoover H C	Henry L	24	23	2	1899
Roosevelt F D	Roosevelt A E	23	20	6	1905
Truman H S	Wallace E V	35	34	1	1919
Eisenhower D D	Doud G	25	19	2	1916
Kennedy J F	Bouvier J L	36	24	3	1953
Johnson L B	Taylor C A	26	21	2	1934
Nixon R M	Ryan T C	27	28	2	1940
Ford G R	Warren E B	35	30	4	1948
Carter J E	Smith R	21	18	4	1946
Reagan R	Wyman J	28	25	2	1940
Reagan R	Davis N	41	28	2	1952

Table 3.4: Personal information

Presidential hobbies	
President name	Hobby
Adams J Q	Billiards
	Swimming
	Walking
Arthur C A	Fishing
Cleveland G	Fishing
Coolidge C	Fishing
	Golf
	Indian Clubs
	Mechanical Horse
	Pitching Hay
Eisenhower D D	Bridge
	Golf
	Hunting
	Painting
	Fishing
Garfield J A	Billiards
Harding W G	Golf
	Poker
	Riding
Harrison B	Hunting
Hayes R B	Croquet
	Driving
	Shooting
Hoover H C	Fishing
	Medicine Ball
Jackson A	Riding
Jefferson T	Fishing
	Riding
Johnson L B	Riding
Kennedy J F	Sailing
	Swimming
	Touch Football
Lincoln A	Walking
McKinley W	Riding
	Swimming
	Walking
Nixon R M	Golf
Roosevelt F D	Fishing
	Sailing
	Swimming
Roosevelt T	Boxing
	Hunting
	Jujitsu
	Riding
	Shooting
	Tennis
	Wrestling
Taft W H	Golf
	Riding
Taylor Z	Riding
Truman H S	Fishing
	Poker
	Walking
Van Buren M	Riding
Washington G	Fishing
	Riding
Wilson W	Golf
	Riding
	Walking

Table 3.5: Presidential hobbies

State name	Administration entered	Year entered
Massachusetts	?	1776
Pensylvania	?	1776
Virginia	?	1776
Connecticut	?	1776
South Carolina	?	1776
Maryland	?	1776
New Jersey	?	1776
Georgia	?	1776
New Hampshire	?	1776
Delaware	?	1776
New York	?	1776
North Carolina	?	1776
Rhode Island	?	1776
Vermont	1	1791
Kentucky	1	1792
Tennessee	2	1796
Ohio	4	1803
Louisianna	6	1812
Indiana	7	1816
Mississippi	8	1817
Illinois	8	1818
Alabama	8	1819
Maine	8	1820
Missouri	9	1821
Arkansas	12	1836
Michigan	12	1837
Florida	14	1845
Texas	15	1845
Iowa	15	1846
Wisconsin	15	1848
California	16	1850
Minnesota	18	1858
Oregon	18	1859
Kansas	18	1861
West Virginia	19	1863
Nevada	19	1864
Nebraska	20	1867
Colorado	22	1876
North Dakota	26	1889
South Dakota	26	1889
Montana	26	1889
Washington	26	1889
Idaho	26	1890
Wyoming	26	1890
Utah	27	1896
Oklahoma	30	1907
New Mexico	31	1912
Arizona	31	1912
Alaska	43	1959
Hawaii	43	1959

Table 3.6: Presidential information

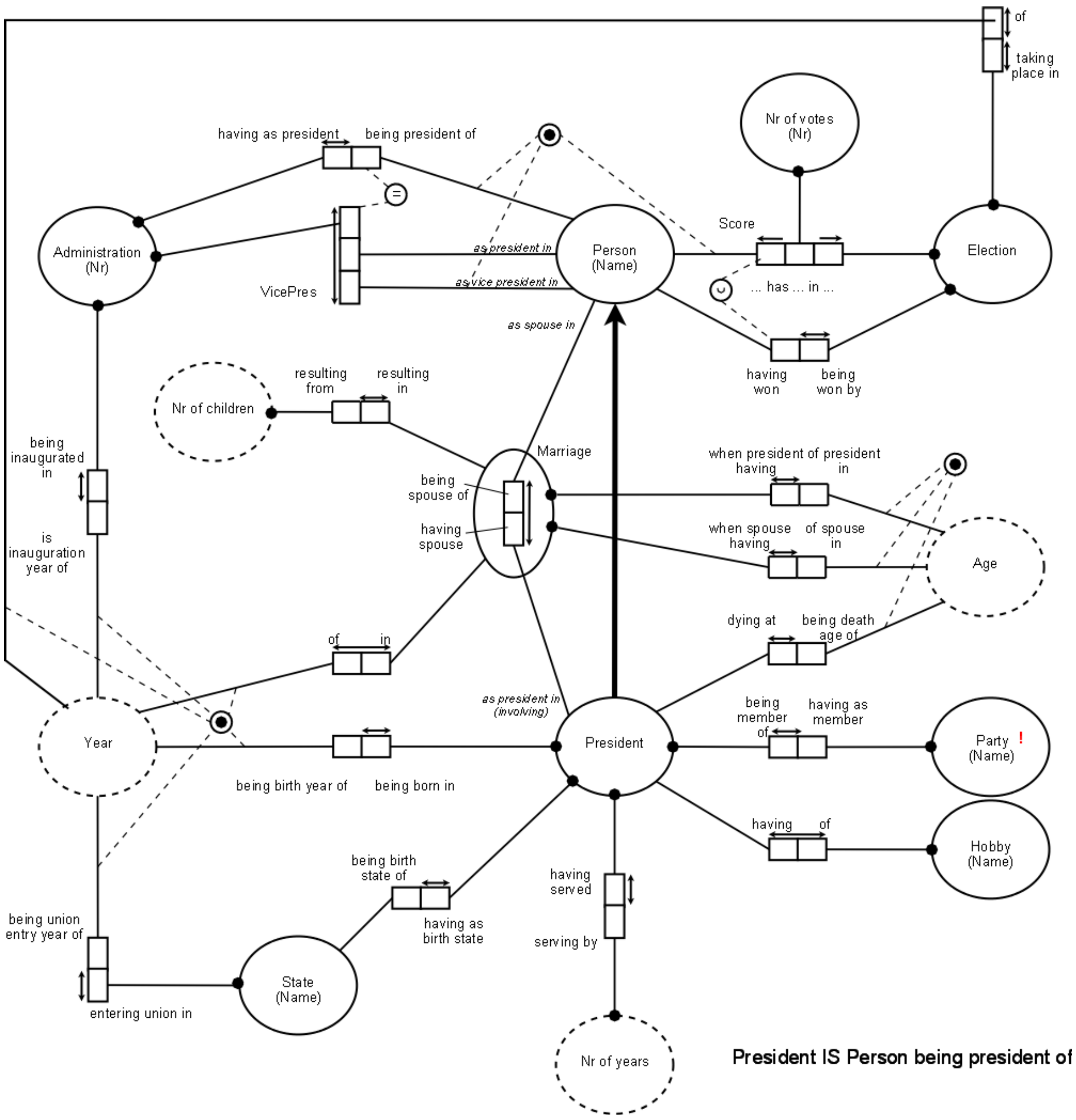


Figure 3.12: Presidential Database fragment

A person may be spouse of several presidents, and will then occur more than once in the list of president spouses. If we dont want duplicates in the list, we require:

```
LIST DISTINCT Person being spouse of President
```

3. *Which presidents have been married?*
   This is answered by:

   ```
   LIST President having spouse
   ```

   In this case president Harrison for example, will result twice as this president married two times. To remove these duplicates, we modify the query into:

   ```
   LIST DISTINCT President having spouse
   ```

4. *Who is the spouse of president Johnson?*
   In this query we add the usage of a label value:

   ```
   LIST Person being spouse of President "Johnson"
   ```

3.11.2 Statistical functions

1. *How many individual presidents were there?*
   The COUNT operator counts the number of facts that are derived from the information descriptor, taking duplicates into account: *SQL 6.1.8*

   ```
   LIST COUNT President
   ```

2. *How many presidential marriages were there altogether?* *SQL 6.1.9*

   ```
   COUNT Marriage
   ```

   Note that this number will not be equal to the number married presidents.

3. *How many presidents have been married?*
   This is answered by:

   ```
   LIST COUNT DISTINCT President having spouse
   ```

4. *Show the average age at death of deceased presidents.*
   The AVERAGE operator takes the average value of the results produced. Note that this operator takes duplicates into account. so we get: *SQL 6.1.1*

   ```
   LIST AVERAGE Age being death age of President
   ```

   Note that (1) presidents can occur at most once in the results from Age being death age of President. (2) the information descriptor Age being death age of President will not produce presidents that are still alive.

5. *Show oldest death age at death of a president.* *SQL 6.1.3,4,5*

   ```
   LIST MAXIMUM Age being death age of President
   ```

6. *What is the total number of children resulting from presidential maariages?*

   ```
   LIST SUM Nr of children resulting from Marriage
   ```

There is also a grouping form for statistical functions.

1. *What is number of presidents for each party?*
   The counting is supposed to group the membership tuples according to the party involved, and then to count the tuples per group.

   ```
   LIST COUNT GROUPWISE President being member of Party
   ```

   The information descriptor COUNT GROUPWISE President being member of Party has the following result:

from	upto
4	Party with Name ”Demo-Rep”
13	Party with Name ”Democratic”
2	Party with Name ”Federalist”
16	Party with Name ”Republican”
4	Party with Name ”Whig”

   So the counting is done group wise, grouping by parties. The LIST operator will transform this into the following answer:

   ```
   4
   13
   2
   16
   4
   ```

2. *Show the total number of number of children for each marriage.*
   The number of children resulting from a marriage is described as Nr of children resulting from Marriage. These facts are to be grouped by marriage, and the number of children is summed per marriage: *SQL 6.1.2*

   ```
   SUM GROUPWISE Nr of children resulting from Marriage
   ```

3.11.3 Multi-valued result

The construction

$D_1, D_2, \ldots D_k$ FROM D

combines the results of corresponding results from $D_1, D_2, \ldots, D_k$, using D as a selection condition.

1. *List the name, birth year and spouse names of all married presidents.*
   SQL 9.1.2

   ```
   Name of, Nr of Year being birth year of, Name of Person being spouse of FROM
   President having spouse
   ```

2. *Show name and birth year of all presidents.*

   ```
   LIST Name of, Year being birth year of FROM President
   ```

   This may also be formulated as

   ```
   LIST Name of, Year being birth year of
   ```

   as President does not impose any selection restriction.

3. *Show name and birth year of all presidents having served 8 years.*

```
LIST Name of, Year being birth year of FROM President having served Nr of years 8
```

4. *Show the total number of marriages and the total of the number of children of all presidents.* *SQL 6.1.2*

```
LIST COUNT Marriage, SUM GROUPWISE Nr of children resulting from FROM Marriage
```

5. *For each party, count the number of presidents who belonged to this party. List the name of each party, together with this count.* *SQL 8.1.1*

```
LIST NUMBER GROUPWISE President being member of, Name of FROM Party
```

The following will be displayed:

```
4, Party with Name ”Demo-Rep”
13, Party with Name ”Democratic”
2, Party with Name ”Federalist”
16, Party with Name ”Republican”
4, Party with Name ”Whig”
```

6. *For each state, count the number of presidents born in that state. List the state name together with this count.* *SQL 8.1.2*
   Analogous to the previous example:

```
LIST COUNT GROUPWISE President having as birth state, Name of FROM State
```

The result displayed is:

```
1, State with Name ”California”
1, State with Name ”Georgia”
1, State with Name ”Illinois”
1, State with Name ”Iowa”
1, State with Name ”Kentucky”
3, State with Name ”Massachusetts”
1, State with Name ”Missouri”
1, State with Name ”Nebraska”
1, State with Name ”New Hampshire”
1, State with Name ”New Jersey”
4, State with Name ”New York”
2, State with Name ”North Carolina”
7, State with Name ”Ohio”
1, State with Name ”Pennsylvania”
1, State with Name ”South Carolina”
2, State with Name ”Texas”
2, State with Name ”Vermont”
8, State with Name ”Virginia”
```

3.11.4 Combining information descriptors

In this section we discuss a number of operators that enable us to combine selection criteria.

1. *Which democratic presidents were born in Texas?*
   We reformulate this question to bring it in ORNF, using the grammar as defined by the conceptual schema. The presidents we are looking for in this information need should satisfy two conditions: (1) being democratic and (2) being born in Texas. Being a democratic presidents is expressed as President being member of Party with name ”Democratic”. The requirement to presidents being born in Texas is formulated as President having as birth state State with Name ”Texas”. The operator AND ALSO is used to combine these two requirements.

   LIST President having as birth state State with Name ”Texas” AND ALSO being member of Party with Name ”Democratic”

   This may be simplified as follows:

   LIST President having as birth state State with Name ”Texas” AND ALSO being member of Party with Name ”Democratic”

3.11.5 Complex examples

1. *Show the age at death of the Democratic president who died the oldest.*
   LIST Age beging death age of President (dying at Age = MAXIMUM Age being death age AND ALSO being member of Party with Name ”Democratic”) *SQL 6.1.3,4,5*

2. *Show the age at death of the president who died the youngest.*
   LIST Age beging death age of President dying at Age = MINIMUM Age being death age *SQL 6.1.6*

3. *For each party, calculate the total number of years served by presidents of that party, the number of presidents, and the average number of years served. List the party, total number of years served, number of presidents, and average number of years served.* *SQL 8.1.3*

   ```
   LIST Name of,
        SUM GROUPWISE Number of years serving by President being member of,
        COUNT GROUPWISE President being member of,
        AVERAGE GROUPWISE Number of years serving by President being member of
        FROM Party
   ```

   We first consider the information descriptors in isolation:

   (a) The information descriptor Name of a party produces the result:

from	upto
”Demo-Rep”	Party with Name ”Demo-Rep”
”Democratic”	Party with Name ”Democratic”
”Federalist”	Party with Name ”Federalist”
”Republican”	Party with Name ”Republican”
”Whig”	Party with Name ”Whig”

   (b) The information descriptor SUM GROUPWISE Number of years serving by President being member of produces the result:

from	upto
28	Party with Name ”Demo-Rep”
73	Party with Name ”Democratic”
11	Party with Name ”Federalist”
67	Party with Name ”Republican”
6	Party with Name ”Whig”

   (c) The information descriptor COUNT GROUPWISE President being member of produces the result:

from	**upto**
4	Party with Name ”Demo-Rep”
13	Party with Name ”Democratic”
2	Party with Name ”Federalist”
16	Party with Name ”Republican”
4	Party with Name ”Whig”

(d) The information descriptor AVERAGE GROUPWISE Number of years serving by President being member of produces the result:

from	**upto**
7.0	Party with Name ”Demo-Rep”
5.6	Party with Name ”Democratic”
5.5	Party with Name ”Federalist”
4.2	Party with Name ”Republican”
1.5	Party with Name ”Whig”

Combining these results leads to the answer displayed:

```
”Demo-Rep”, 28, 4, 7.0
”Democratic”, 73, 13, 5.6
”Federalist”, 11, 2, 5.5
”Republican”, 67, 16, 4.2
”Whig” , 6, 4, 1.5
```

3.11.6 Arithmetic and relational operators

1. *Find those deceased politicians who served more than 10% of their lives as president. List their names and this percentage.* *SQL 7.1.3*
We use the LET mechanism to simplify this query.

    LET Percentage of BE Nr of years serving by / Age being death age of

    Then p Percentage $pres$ if p is the precentage that a president served. Note that this relation only is defined for deceased presidents, as only those presidents occurr in Age being death age of.

    The restriction to percentages of more than 10% is obtained by the descriptor Percentage $>_2$ 0.1. Note that the use of the head-tail operator is required as we need the relation between the percentage and the associated president.

    The answer now becomes:

    Percentage $>_2$ 0.1, Name of
    OF President

2. *Give the name and age of president and spouse for those marriages where the president was at least 5 years older than the spouse and the spouse was less than 20 years old.* *SQL 7.1.4*

    Name of President as president in, Age of president in,
    Name of Person as spouse in, Age of spouse in
    OF Marriage when president having Age $\geq$ 5 + Age of spouse in
        THAT Marriage when spouse having Age $>$ 20

3. *Count the presidents who were members of the same party and who were born in the same state. List party, state of birth, and this count.* *SQL 8.1.4*
First we introduce the binary relation that presidents are member of the same party:

    President being member of Party having as member President

This relation both is reflexive and symmetric. An asymmetric version is created by using alphabetic order of president name as a restriction:

```
President being member of Party having as member President
INTERSECTION
President having Name < Name of President
```

Next we add the condition that these presidents are to be born in the same state:

```
President being member of Party having as member President
INTERSECTION
President having Name < Name of President
INTERSECTION
President being born in State being birth place of President
```

This relation has to be counted according to party. Therefore we extend the relation to the party involved:

```
(President being member of Party having as member President
INTERSECTION
President having Name < Name of President
INTERSECTION
President being born in State being birth place of President
) being member of Party
```

The resulting information descriptor thus relates the required presidents to their party. Next we count the number of presidents for each party by:

```
NUMBER (President being member of Party having as member President
        INTERSECTION
        President having Name < Name of President
        ) being member of Party
```

The result then is obtained as:

```
NUMBER (President being member of Party having as member President
        INTERSECTION
        President having Name < Name of President
        ) being member of,
Name of
FROM Party
```

4. *For each birth state / party combination, count the presidents who were born in that state and who were members of that party. List state, party and this count.* *SQL 8.1.5*
The combination of birth states and parties is made by introducing the following derived fact type:

```
LET Combinations BE being state in:State being birth state of, being party in:Party
```

This fact type is diplayed in the following figure:

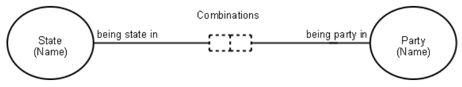


Figure 3.13. Derived facttype *Combinations*

Then we get:

```
Name of State being state in,
Name of Party being party in,
NUMBER President being born in State being state in
        INTERSECTION
        President being member of Party being party in
OF Combinations
```

5. *For each party, list the party name and the number of presidents born after the year 1850.* *SQL 8.1.6*
The information descriptor President being born in Year $>$ 1850 relates presidents to their birth year, if born after 1850. Using the operator THE, this relation is transformed to a binary relation between those presidents.

   This allows us to extend the path, making a relation between those presidents and the parties they are member of: (THE President being born in Year $>$ 1850) being member of. The query then is:

   (THE President being born in Year $>$ 1850) being member of,
   Name of
   OF Party

6. *List the names of presidents and the number of their marriages for those presidents who married more than once.* *SQL 8.1.7*
The construction NUMBER Marriage involving computes the number of marriages for each president. We add the restriction on this of the number of marriages being greater than 1: NUMBER Marriage involving $>_2$ 1 This leads to the query:

   Name of,
   NUMBER Marriage involving $>_2$ 1
   OF President

7. *Find those presidents who married at least twice and whose maximum number of children in any of their marriages exceeds their minimum number or children by at least 2. List their names, and the maximum and minimum number of children.* *SQL 8.1.9*
Presidents that married at least twice are characterized by NUMBER President as president in Marriage $>_2$ 1 For these presidents we require for their number of children:

   (NUMBER President as president in Marriage $>_2$ 1)
   AND ALSO
   (NUMBER President)

3.11.7 Rest

1. *Which presidents were no more than 10% older than their spouse(s) at the time of marriage! List their name, their age at marriage, their spouse's age at marriage and the ratio of the president's age to his spouse's age as a decimal number.* *SQL 7.1.5,6,7*

   STATE OF BIRTH??

2. *For those parties which had more than 8 presidents born after 1850, list the names of the parties and the corresponding number of presidents born after 1850.* *SQL 8.1.8*
The required answer:

   $$\{(pn, n) \mid \exists_p [pn \text{ is name of party } p \wedge n \text{ n is number of presidents from that party born after 1850}]\}$$

   First we construct a relate parties to presidents born after 1850:

   Party having as member President being born in Year $>$ 1850

   Each tuple $\langle \mathbf{fr:}\ p, \mathbf{to:}\ 1850 \rangle$ in the result of this information descriptor corresponds to a particular president from that part p being born after 1850. We number for each party, and require the number to be greater than 8:

   (NUMBER Party having as member President being born in Year $>$ 1850) $>_2$ 8

Each tuple $\langle$**fr:** p, **to:** $n\rangle$ corresponds to a unique party p, having n presidents born after 1850 (where $n > 8$). As we will need both party and number of this relation, we use the LET-construct to make this relation accessible from both sides:

```
LET PartNr BE
being party in: (NUMBER Party having as member President being born in Year > 1850)
>2 8 :being number in
```

The query then is obtained as:

```
being party in, being number in OF PartNr
```

3. *If we want to produce as a result a table with 4 columns, where each row consists of the president name of the P_TABLE and his birth year followed by the president name of M_TABLE and spouse name, for all possible combinations, we have to write the following SQL query:* *SQL 9.1.1*

4. *List names, birth years, marriage years and spouses of all married presidents. Order by president name.* *SQL 9.1.3*

5. *List president name, birth year, the administrations served as president and the vice presidents in each administration, in order of administration number.* *SQL 9.1.4*

6. *List the names, birth years and hobbies of all presidents born before 1800. Order by birth year and president name.* *SQL 9.1.5*

7. *List name, birth year, marriage years and spouse names of those presidents who were born before 1776 and married before 1800. Order the list on president name in ascending order.* *SQL 9.1.6*

8. *List the name, birth year, age at marriage, spouse's age at marriage and name, for all presidents who married when they were less than 20 years old, or who married a spouse less than 18 years of age.* *SQL 9.1.7*

9. *For each president with more than three children, list their name, their birth year and the number of children from all marriages. Order by number of children in descending order, and then by name.* *SQL 9.1.8*

10. *Show the married presidents who have more than 3 hobbies and show in the same view for each president as well the number of marriages as the number of hobbies, ordered by decreasing number of hobbies.* *SQL 9.2*

11. *List all the facts of those presidential marriages which resulted in a number of children that is greater than the average number of children per presidential marriage.* *SQL 10.1.1*

    ```
    Marriage resulting in Nr of children with Nr > AVERAGE Nr of Nr of Children resulting from
    Marriage
    ```

    The total role constraint justifies the following simplification:

    ```
    Marriage resulting in Nr of children with Nr > AVERAGE Nr of Nr of Children
    ```

12. *Show the name and age of the president who died the youngest.* *SQL 10.1.2*

```
Name of, Age being death age of OF
President dying at Age with Number of years = MINIMUM Number of years of Age being
death age of
```

Using the total role constraint on the role *being death age of* this can be simplified as:

```
Name of, Age being death age of OF
President dying at Age with Number of years = MINIMUM Number of years of Age
```

13. *List the hobbies and names of all those presidents who served 8 years or longer.* *SQL 10.1.3*

```
Name of Hobby of, Name of OF
President having served Nr of years with Nr >= 8
```

14. *Which presidents never won a election?* *SQL 10.1.4*

```
President BUT NOT having won Election
```

15. *List the hobbies of presidents who served for (a) 12 years or more, (b) 8 years or more.* *SQL 10.1.5*

```
Name of Hobby of OF
President having served Nr of years with Nr >= 12 AND ALSO having served Nr of years
with Nr >= 8
```

16. *Which state provided the largest number of presidents, and what is that number?* *SQL 10.2.1*

17. *Find those states which entered the union before President Washington was inaugurated.* *SQL 10.2.2,3,4*
The problem for this information need is that President Washington has been inaugurated more than once.

```
State entering union in Year with Nr < ALL Nr of Year being inauguration year of President
with Name ”Washington G.”
```

18. *List all the facts available about presidents who were inaugurated after Hawaii entered the union.* *SQL 10.2.5*

```
EVERYTHING OF
President being inaugurated in Year with Nr > Nr of Year being union entry year of State
with Name ”Hawai”

Name of, Nr of years being death age of OF
President in Marriage ???????
```

19. *Find those states which entered the union the same year as President Eisenhower was born.* *SQL 10.2.7*

```
State entering union is Year being birthyear of President with Name ”Eisenhower D D”
```

20. *Show for each state which president was the first-born out of that state (limit yourself to the period between 1800 and 1850).* *SQL 10??*

21. *For each president who was born in a year in which at least one other president was born, list his name and birth year.* *SQL 11.1.1*

```
Name of, Nr of Year being birth year of OF
President being born in Year being birth year of ANOTHER President
```

Niet van toepassing   Schema niet voorhanden

22. *List the name and birth year of those presidents who were inaugurated at least once within 45 years of their birth year.* *SQL 12.1.1*

```
Name of, Nr of Year being birth year of OF
President being inaugurated in Year with Nr ¡= 45 + Nr of Year being birthyear of THAT
President
```

23. *List the election year and winner of those elections in which the winner received more than 80% of the votes in that election.* *SQL 12.1.2*

3.12 Definitions overview

Schema sentence – Sentence that describes the schema elements that constitute the conceptual model. From these sentences the information grammar is derived.

In the context of these lecture notes this is refined to:
Sentence that describes the schema elements that constitute the conceptual model. From these sentences the information grammar is derived.

Stating sentence – Sentence to inform the information system about new facts in the universe of discourse (see effect of addition), or about the withdrawal of the validity of facts specified (see effect of deletion).

In the context of these lecture notes this is refined to:
Sentence to inform the information system about new facts in the universe of discourse (see effect of addition), or about the withdrawal of the validity of facts specified (see effect of deletion).

Asking sentence – Specification of an information need. The system will respond with a list of stating sentences, corresponding to the facts that satisfy the specification.

In the context of these lecture notes this is refined to:
Specification of an information need. The system will respond with a list of stating sentences, corresponding to the facts that satisfy the specification.

Effect of addition – Adding facts leads to the minimal extension of the information base (population) containing those facts. If no such extension exists, then the information system reports an extension error, and does not alter the information base.

In the context of these lecture notes this is refined to:
Adding facts leads to the minimal extension of the information base (population) containing those facts. If no such extension exists, then the information system reports an extension error, and does not alter the information base.

Effect of deletion – Deleting facts leads to the maximal subset the information base (population) not containing those facts. If no such subset exists, then the information system reports an deletion error, and does not alter the information base.

In the context of these lecture notes this is refined to:
Deleting facts leads to the maximal subset the information base (population) not containing those facts. If no such subset exists, then the information system reports an deletion error, and does not alter the information base.

Unquote convention – Convention to denote an abstract object instance by its unquoted name. So for example *Jan* refers to Person with Name 'Jan'.

In the context of these lecture notes this is refined to:
Convention to denote an abstract object instance by its unquoted name. So for example *Jan* refers to Person with Name 'Jan'.

Information descriptor – General format of information need description in Lisa-D.

In the context of these lecture notes this is refined to:
General format of information need description in Lisa-D.

Lexicon – The Lisa-D lexicon describes the elementary language constructions.

In the context of these lecture notes this is refined to:
The Lisa-D lexicon describes the elementary language constructions.

Result table – The evaluation result of a Lisa-D information descriptors. This result has the form of a inhomogeneous binary relation that may contain tuples more than once.

In the context of these lecture notes this is refined to:
The evaluation result of a Lisa-D information descriptors. This result has the form of a inhomogeneous binary relation that may contain tuples more than once.

Role name – The name of a role that corresponds to the path from the base of that role to the corresponding fact type.

In the context of these lecture notes this is refined to:
The name of a role that corresponds to the path from the base of that role to the corresponding fact type.

Inverse role name – The name of a role that corresponds to the path from the corresponding fact type to the base of that role.

In the context of these lecture notes this is refined to:
The name of a role that corresponds to the path from the corresponding fact type to the base of that role.

Connector name – The name of a role that describes a path through the corresponding fact type via this role.

In the context of these lecture notes this is refined to:
The name of a role that describes a path through the corresponding fact type via this role.

Questions

Version: 01-12-2007

1. Proof Lemma 3.5.1 (page 76).
2. Consider the populated schema from figure 3.2, and evaluate the following information descriptors:
   (a) Person working for Department providing work to Person with Name "Jan"
   (b) Person in Contract from Person with Name "Kees"
3. Proof the equivalence of the following information descriptors:
   (a) Person working for Department
   (b) Person in Contract for Department
   (c) working for
4. Consider the populated schema from figure 3.14, and provide a Lisa-D formulation for the following information needs:

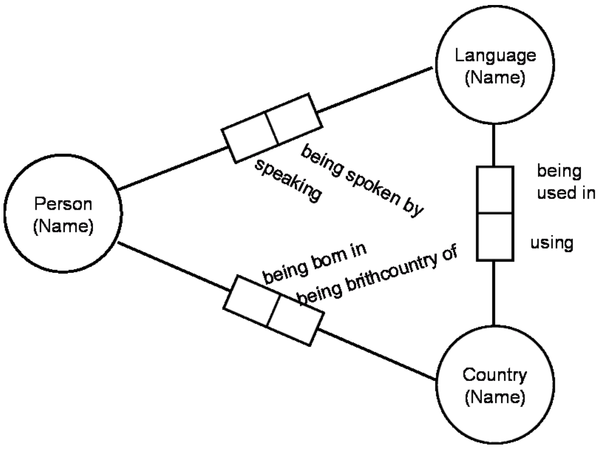


Figure 3.14: Interpreter model

(a) Which persons are speaking Greek?

(b) Who is born in France and speaks German?

(c) What languages are spoken in the country where Piet has been born?

(d) What languages are spoken by both Piet and Els?

(e) What languages are being spoken in a polyglot country?

5. Consider the schema from figure 3.2, and proof the equivalence of the following information descriptors:

    (a) Person working for Department

    (b) Person in Contract for Department

    (c) working for

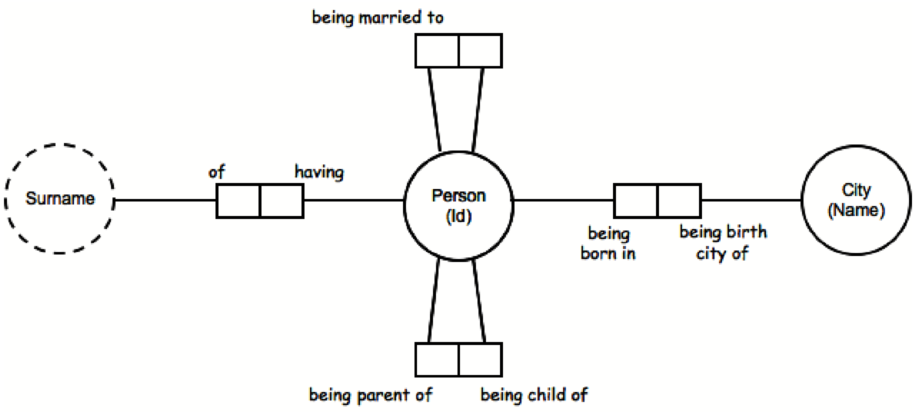


Figure 3.15: Marriage and Parentship

6. Consider the conceptual schema from figure 3.15. Formulate the following information needs in terms of ORC.

    (a) Give the surnames of all persons being married with someone born in Arnhem.

    (b) Give the surnames of all persons being married with someone born not in Arnhem.

(c) Give the surname of all persons being born in Nijmegen, married to someone born in Arnhem, and having a grandchild born in Wijchen.

(d) Give the surname of all persons being born in a different city than the one they are married to.

(e) Give the surnames of the persons with children born in an different city.

7. For each of the following statements, either provide a proof a give a counter example:

   (a) D1 AND D2 = D2 AND D1

   (b) D1 BUT NOT D2 = D2 BUT NOT D1

   (c) D1 BUT NOT (D2 AND D3) = (D1 BUT NOT D2) AND (D1 BUT NOT D3)

8. (tentamen 2005)

   We beschouwen een administratie waarin gegevens van studenten worden opgeslagen met de opleiding die zij volgen. Studenten worden geïdentificeerd met een studentnummer. Verder worden gegevens van docenten opgeslagen. Docenten worden, anders dan studenten, geïdentificeerd met een sofinummer. Omdat Duits een belangrijke taal is voor deze instelling, worden studenten Duits ingeschakeld om bijles Duits te geven aan andere studenten. Van zowel studenten die bijles geven, als van docenten, wordt het salaris geadministreerd. Enige voorbeeldtabellen zijn:

Volgt:

Student	Naam student	Opleiding
S123456	Blair	Engels
S123457	Bohr	Natuurkunde
S123458	Kohl	Duits
S123459	Schmidt	Duits

Bijles:

Geeft bijles	Krijgt bijles
S123458	S123456
S123458	S123459
S123459	S123456

Docenten:

Docent	Telnr
02934124	024-3653652
	06-12351672
09657829	024-3765431

Verdiensten:

Persoon	Boekjaar	Salaris
02934124	2004	20000
02934124	2005	21000
09657829	2005	30000
S123458	2005	10000
S123459	2005	5000

   (a) Geef de elementaire verwoordingen van de elementaire feittypen die hierin liggen opgesloten. Leid vervolgens uit deze zinnen het ORM schema af.

   (b) Formuleer met behulp van ORC:

      i. Hoe heten de studenten die bijles Duits krijgen van student Kohl?

      ii. Zijn er studenten Duits die bijles Duits krijgen?

      iii. Hoe heten de studenten die bijles Duits geven, maar zelf ook bijles krijgen in dat vak?

Chapter 4

Object-Role Calculus - Formal Reasoning

4.1 Business rules

Version: 18-11-05

Business rules may be seen as the basic properties of some (business oriented) universe of discourse. In this context we will refer to this discourse also as business area. These rules form the basic properties in the mission of that business area. When reasoning about a business area, they are the axioms. Such reasoning typically is done by the managers of that business area.

Business rules correspond to ORC conditions. The advantage of using ORC is that this language is easily understood by a the managers. For simple cases, managers can transform a natural language expression into the requirements of this controlled language.

When reasoning about the application domain, managers use a reasoning style that is like reasoning with ORC. Business rules are the domain axioms. Using the reasoning rules, domain properties can be derived. Reasoning with ORC is just as formal as formal reasoning in logics.

Logics provide a more compact representation, hiding irrelevant details. ORC is not compact, but this is the very reason that is more easily read by domain experts.

In this course, we will make our point by giving an example.

The base rule for formal reasoning is called *modus ponens*. This rule states: if we know (have proven) (1) x and (2) from x can be concluded y (or: $x \Rightarrow y$, then we also have proven y. When reasoning within ORC, this rule of reasoning is not directly applicable. We will introduce two reasoning rules instead:

1. The first implication rule
2. The second implication rule

4.1.1 The first implication rule

The first implication rule states:

- If we have proven both:
  - D1 IS ALSO D2 D3
  - D3 IS ALSO D4
- Then we can conclude: D1 IS ALSO D2 D4

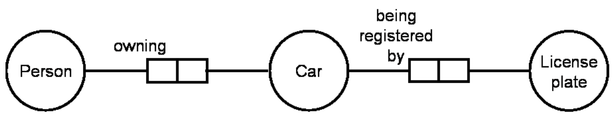


Figure 4.1: Cars and License

4.1.1.1 Examples

As an example, we consider the conceptual schema of figure 4.1. Suppose we are given the following rules:

- Each person must own a car: Person IS ALSO owning Car
- Each car must be registered by a license plate: Car IS ALSO being registered by License plate

We want to prove from these rules that each person has a license plate:

> Person IS ALSO owning Car being registered by License plate

using the formal reasoning system! We apply the first implication rule by setting:

> D1**:** Person
> D2**:** owning
> D3**:** Car
> D4**:** being registered by License plate

Then we have

> D1 IS ALSO D2 D3**:** Person IS ALSO owning Car
> D3 IS ALSO D4**:** Car IS ALSO being registered by License plate

and thus we conclude:

> D1 IS ALSO D2 D4**:** Person IS ALSO owning Car being registered by License plate

We give another example. Suppose we are given (have proven) the following rules:

- Each car must be registered by a license plate:

  being manufactured in Country from ”EU” IS ALSO satisfying seatbelt rules

- People may only own a car that has been produced in their home country:

  Person IS ALSO owning Car being manufactured in Country from ”EU”

We want to prove from this knowledge that each person satisfies the seatbelt rules:

> Person IS ALSO owning Car satisfying seatbelt rules

We apply the first implication rule by setting:

> D1**:** Person
> D2**:** owning Car
> D3**:** being manufactured in Country from ”EU”
> D4**:** satisfying seatbelt rules

Then we have

> D1 IS ALSO D2 D3**:** Person IS ALSO owning Car being manufactured in Country from ”EU”
> D3 IS ALSO D4**:** being manufactured in Country from ”EU” IS ALSO satisfying seatbelt rules

and thus we conclude:

> D1 IS ALSO D2 D4**:** Person IS ALSO owning Car satisfying seatbelt rules

4.1.1.2 Proof of first implication rule

In this section we prove the validity of the first implication rule. Assume:

- D1 IS ALSO D2 D3
- D3 IS ALSO D4

Then we have to prove: D1 IS ALSO D2 D4

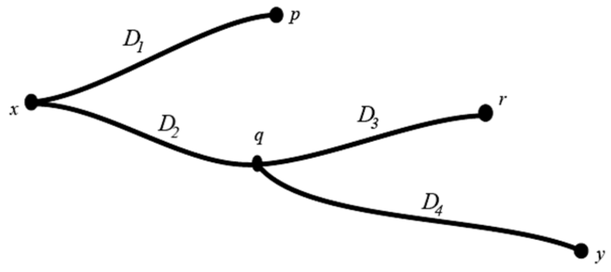


Figure 4.2: Displaying the situation

Proof:

Suppose $\langle \mathbf{fr:}\, x, \mathbf{to:}\, y\rangle \in \mathscr{R}(D_1)$. From D1 IS ALSO D2 D3 we conclude that there are q and r such that:

- $\langle \mathbf{fr:}\, x, \mathbf{to:}\, q\rangle \in \mathscr{R}(D_2)$
- $\langle \mathbf{fr:}\, q, \mathbf{to:}\, r\rangle \in \mathscr{R}(D_3)$

From D3 IS ALSO D4 we conclude that for some y we have:

- $\langle \mathbf{fr:}\, q, \mathbf{to:}\, y\rangle \in \mathscr{R}(D_4)$

We conclude: $\langle \mathbf{fr:}\, x, \mathbf{to:}\, y\rangle \in \mathscr{R}(D_2 D_4)$. And thus we have proven: D1 IS ALSO D2 D4.

4.1.2 The second implication rule

The second implication rule is the interpretation of modus ponens employing both head and tail of the result of an ORC expression.

- If we have proven both:
  - D1 D2 D3 IMPLIES D4
  - D5 IMPLIES D2
- Then we can conclude: D1 D5 D3 IMPLIES D4

4.1.2.1 Example

As an example, we focus on uniqueness within binary relationships. Uniqueness for ownership of a car is expressed as:

Car being owned by Person owning Car IMPLIES Car

Uniqueness for license plate registration is expressed as:

License plate registering Car being registered by License plate IMPLIES License plate

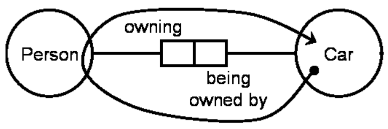


Figure 4.3: Displaying the situation

Combining these rules, we can conclude that each license plate has a unique owner by applying the second implication rule. We choose as follows:

D1: License plate registering
D2: Car
D3: being registered by License plate
D4: License plate
D5: Car being owned by Person owning Car

By applying the second implication rule we conclude:

License plate registering Car being owned by Person owning Car being registered by License plate IMPLIES License plate

Using the Negative-positive formulation transformation (lemma 3.8.1), we can transform this into the following negatively formulated sentence:

NO License plate is registering Car being owned by Person owning Car being registered by ANOTHER License plate

4.1.2.2 Proof of first implication rule

In this section we prove the validity of the first implication rule. Assume:

- D1 D2 D3 IMPLIES D4
- D5 IMPLIES D2

Then we have to prove: D1 D5 D3 IMPLIES D4

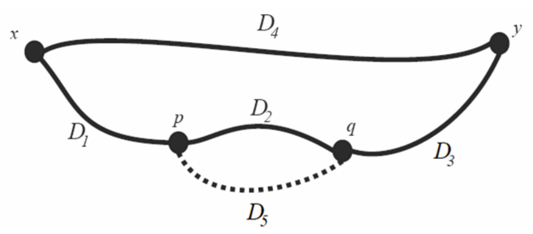


Figure 4.4: Second Implication Rule

Proof:

Assume D1 D2 D3 IMPLIES D4 and D5 IMPLIES D2, and let $\langle \textbf{fr:}\, x, \textbf{to:}\, y\rangle \in \mathscr{R}(D_1 D_5 D_3)$ then for some p and q we have (see figure 4.4):

- $\langle \textbf{fr:}\, x, \textbf{to:}\, p\rangle \in \mathscr{R}(D_1)$
- $\langle \textbf{fr:}\, p, \textbf{to:}\, q\rangle \in \mathscr{R}(D_5)$
- $\langle \textbf{fr:}\, q, \textbf{to:}\, y\rangle \in \mathscr{R}(D_2)$

From D5 IMPLIES D2 we conclude $\langle \textbf{fr:}\, p, \textbf{to:}\, q\rangle \in \mathscr{R}(D_2)$. As a consequence: $\langle \textbf{fr:}\, x, \textbf{to:}\, y\rangle \in \mathscr{R}(D_1 D_2 D_3)$. From D1 D2 D3 IMPLIES D4 we conclude: $\langle \textbf{fr:}\, x, \textbf{to:}\, y\rangle \in \mathscr{R}(D_4)$.

4.2 Combining

In this section we show how information descriptors can be extended to handle multiple aspects. For example, we are interested in the names of all persons living in Brussels. If a person owns a car, then we like to see the brands all cars owned by that person and also all insurance claims from that person for that car. The main structure of the information descriptor is:

```
"required personal properties" FROM Person living in City with Name 'Brussels'
```

The required properties consist of the name of the person and the car brand and insurance properties:

```
Name of Person,
"car brand and insurance properties"
FROM Person living in City with Cityname 'Brussels'
```

If a person owns a car, then the brands of all cars are to be provided, and the insurance data list:

```
Name of Person,
POSSIBLY (Brand of Car,
          ALL Claims from Insurance registered on Car
          FROM being owned by)
FROM Person living in City with Cityname 'Brussels'
```

The main construct is a list of properties $\mathsf{P}_1, \ldots, \mathsf{P}_k$ bound by a specifier D:

$\mathsf{P}_1, \ldots, \mathsf{P}_k$ FROM D

Suppose P_i have reach $[\mathsf{X_i}, \mathsf{Y}]$, and D has reach $[\mathsf{Y}, \mathsf{Z}]$. Then the grouping construct has reach $[\mathsf{X_1} \times \ldots \times \mathsf{X_k}, \mathsf{Z}]$. The keywords POSSIBLY and ALL indicate an optional property and a set property respectively. In the example above, the inner grouping construct

```
Brand of Car,
ALL Claims from Insurance registered on Car
FROM being owned by
```

has reach $[\mathsf{Brand} \times \mathsf{Claim}, \mathsf{Person}]$. As a consequence, the whole grouping construct has reach:

$$[\mathsf{Name} \times \mathsf{Brand} \times \mathsf{Claim}, \mathsf{Cityname}]$$

We will omit the formal definition of the grouping construct, as it falls outside the scope of this course.

4.3 Definitions overview

Schema sentence – Sentence that describes the schema elements that constitute the conceptual model. From these sentences the information grammar is derived.

In the context of these lecture notes this is refined to:
Sentence that describes the schema elements that constitute the conceptual model. From these sentences the information grammar is derived.

Stating sentence – Sentence to inform the information system about new facts in the universe of discourse (see effect of addition), or about the withdrawal of the validity of facts specified (see effect of deletion).

In the context of these lecture notes this is refined to:
Sentence to inform the information system about new facts in the universe of discourse (see effect of addition), or about the withdrawal of the validity of facts specified (see effect of deletion).

Asking sentence – Specification of an information need. The system will respond with a list of stating sentences, corresponding to the facts that satisfy the specification.

In the context of these lecture notes this is refined to:
Specification of an information need. The system will respond with a list of stating sentences, corresponding to the facts that satisfy the specification.

Effect of addition – Adding facts leads to the minimal extension of the information base (population) containing those facts. If no such extension exists, then the information system reports an extension error, and does not alter the information base.

In the context of these lecture notes this is refined to:
Adding facts leads to the minimal extension of the information base (population) containing those facts. If no such extension exists, then the information system reports an extension error, and does not alter the information base.

Effect of deletion – Deleting facts leads to the maximal subset the information base (population) not containing those facts. If no such subset exists, then the information system reports an deletion error, and does not alter the information base.

In the context of these lecture notes this is refined to:
Deleting facts leads to the maximal subset the information base (population) not containing those facts. If no such subset exists, then the information system reports an deletion error, and does not alter the information base.

Unquote convention – Convention to denote an abstract object instance by its unquoted name. So for example *Jan* refers to Person with Name 'Jan'.

In the context of these lecture notes this is refined to:
Convention to denote an abstract object instance by its unquoted name. So for example *Jan* refers to Person with Name 'Jan'.

Information descriptor – General format of information need description in Lisa-D.

In the context of these lecture notes this is refined to:
General format of information need description in Lisa-D.

Lexicon – The Lisa-D lexicon describes the elementary language constructions.

In the context of these lecture notes this is refined to:
The Lisa-D lexicon describes the elementary language constructions.

Result table – The evaluation result of a Lisa-D information descriptors. This result has the form of a inhomogeneous binary relation that may contain tuples more than once.

In the context of these lecture notes this is refined to:
The evaluation result of a Lisa-D information descriptors. This result has the form of a inhomogeneous binary relation that may contain tuples more than once.

Role name – The name of a role that corresponds to the path from the base of that role to the corresponding fact type.

In the context of these lecture notes this is refined to:
The name of a role that corresponds to the path from the base of that role to the corresponding fact type.

Inverse role name – The name of a role that corresponds to the path from the corresponding fact type to the base of that role.

In the context of these lecture notes this is refined to:
The name of a role that corresponds to the path from the corresponding fact type to the base of that role.

Connector name – The name of a role that describes a path through the corresponding fact type via this role.

In the context of these lecture notes this is refined to:
The name of a role that describes a path through the corresponding fact type via this role.

Questions

Version: 01-12-2007

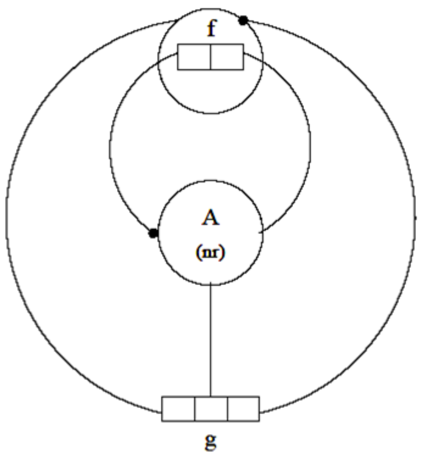


Figure 4.5: Schema for exercise 1

1. Is the schema from figure 6.13 populatable?
2. Express the relationship between the following kinds of constraints:
   (a) total role and frequency constraint
   (b) unique role and frequency constraint

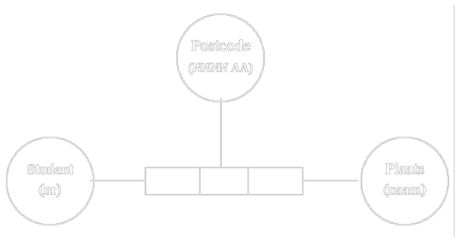


Figure 4.6: Missing dependencies

3. Consider the schema from figure 6.14. Which dependencies seem to be missing in the following schema fragment? What is the effect on populatability after adding these dependencies?
4. Is it possible that a schema is strongly identified but not populatable?
5. What is wrong in the following schema from figure 6.15?

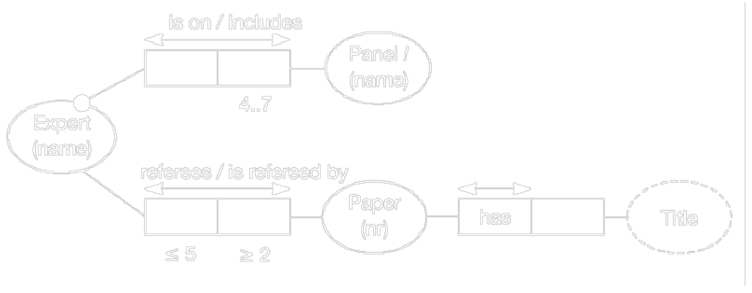


Figure 4.7:

6. Give an example of fact types for each of the following kinds of behavior:
   (a) reflexive
   (b) symmetric
   (c) transitive
   (d) irreflexive
   (e) asymmetric
   (f) antisymmetric

Chapter 5

Advanced modeling constructs

Version: 01-12-05

5.1 Higher-order fact types

Sofar we have mainly seen unary and binary fact types. In this section we focus on higher-order fact types. Consider the following sample sentence:

> Student with StudentNr 12345 gets for Course with Coursename ”Domain Modeling” the Mark with Number 8

It may be easily verified that this sentence is elementary, and can not be split without loosing information. The resulting schema fragment, a ternary fact type, is displayed in figure 5.1.

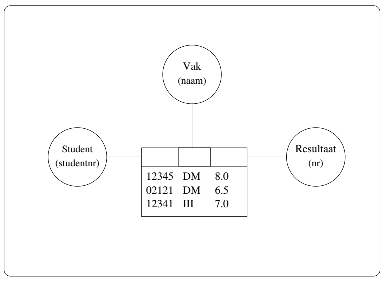


Figure 5.1: Ternary fact type

A slight modification of the sample sentence is:

> Student with StudentNr 12345 takes Course with Coursename ”Domain Modeling” and gets the Mark with Number 8

This sentence is *not* elementary, and can be split into the following sentences:

1. A sentence of type *Registration*:
   Student with StudentNr 12345 takes Course with Coursename ”Domain Modeling”
2. A sentence of type *Result*:
   Registration(*Student with StudentNr 12345 takes Course with Coursename ”Domain Modeling”*) gets the Mark with Number 8

Note that this latter sentence is a statement involving 2 roles! The first role refers to an elementary sentence of type *Registration*. In the next section we will see how this kind of object referencing can be handled. Furthermore, in this reformulated version, a registration can be done even when the mark is not (yet) available.

5.1.1 Objectification

The mechanism to make it possible that sentences themselves can play a role in more complex sentences is called objectification. This means that fact types can be the base of a role. Parsing the sentence Student with StudentNr 12345 takes Course with Coursename ”Domain Modeling” leads to the following populated schema fragment:

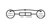


Figure 5.2. Processing the registration

Adding the second sentence Registration(*Student with StudentNr 12345 takes Course with Coursename ”Domain Modeling”*) gets the Mark with Number 8 extends this populated schema into:

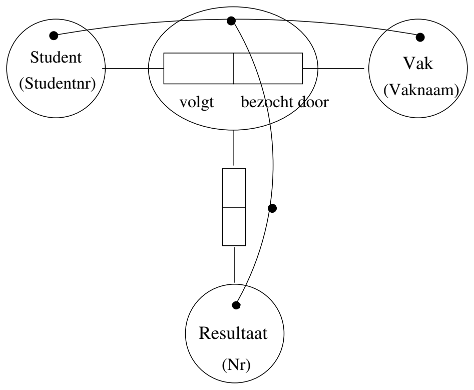


Figure 5.3. Adding the mark

The resulting conceptual schema fragment is:

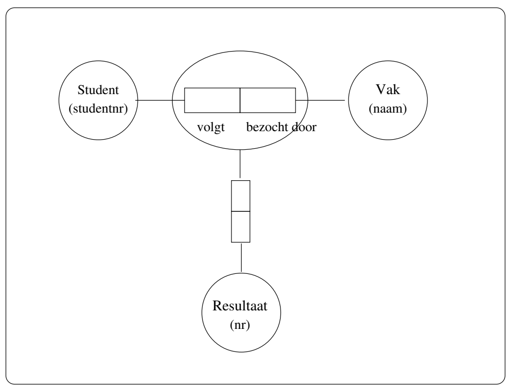


Figure 5.4. The resulting schema fragment

5.1.2 When to objectify

– to be done

5.2 The meaning of a schema

A conceptual schema describes what kind of facts may be stored about the universe of discourse. Some facts are known to be valid, other facts are known to be invalid. In figure 5.5 this situation is displayed. This figure shows a third category of sentences, sentences that are not known to be valid or invalid. Such indeterminate sentences are caused by incomplete knowledge of the underlying domain.

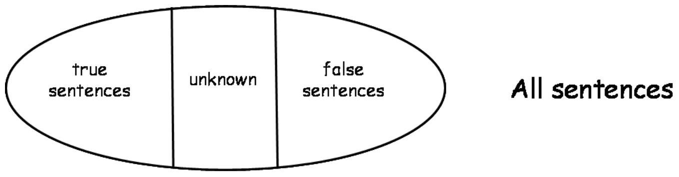


Figure 5.5: Knowledge of valid and invalid sentences

Incomplete knowledge is a complicating factor. For example, yes-no questions about the state of affairs is unknown in case relevant information is missing. This requires the use of a 3-valued logic system.

In this course we will exclude incomplete or missing knowledge. As a consequence, we assume each sentence to express a valid fact in the underlying domain, or an invalid one. This is shown in figure 5.6. This is referred to as the *Closed World Assumption*. When confronted with incomplete sentences in ORM, we will try to handle this incompleteness by adding special fact types that contain the available information for that case. We will thus choose incomplete sentences, rather than incomplete knowledge

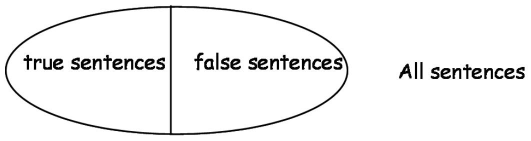


Figure 5.6: No missing knowledge

A consequence of this choice is that each state of the universe of discourse is uniquely characterized by the set of facts that hold at that moment. The state thus may be represented by storing all those facts. At this point we summarize the underlying principles for ORM:

Principle 1: complete knowledge

Principle 2: closed world assumption

Principle 3: All facts that can be stated about the universe of discourse are composed from a limited number of elementary facts.

The conceptual schema describes these elementary facts are described by the concepts and their relations. The latter principle is the motivation to to simply store the elementary facts, and to reconstruct all other facts by applying reasoning rules. Examples of such reasoning languages are: ORC and SQL.

Storing elementary facts means the assignment of facts to all elements of the conceptual schema. Such an assignment is called a *population* of the conceptual schema. Let Σ be a conceptual schema, and P a population of this schema. Then we will shortly denote this as: $\mathsf{IsPop}(\Sigma, P)$. The semantics of a conceptual schema is the set of all possible populations:

$$\mathscr{M}(\Sigma) = \{P \mid \mathsf{IsPop}(\Sigma, P)\}$$

Sofar, the conceptual schema describes only structural communication aspects. Exclusion of particular instances is done via *constraints*. Constraints are rules that are required by the condition $\mathsf{IsPop}(\Sigma, P)$. This strategy to describe a set of populations is called the *exclusion principle*. According to this principle, we make a two-step description:

1. describe structural properties
2. exclude unwanted situations

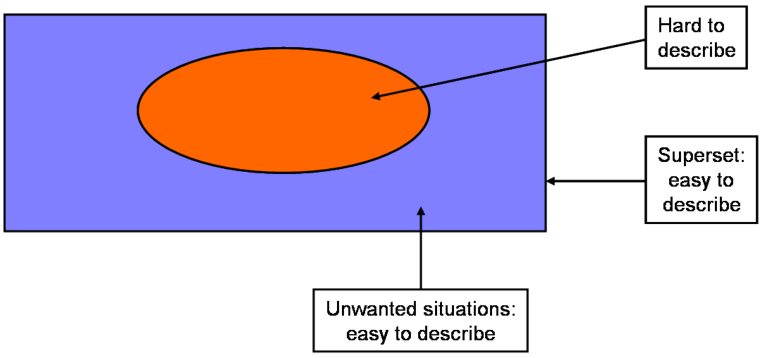

Hard to describe
Superset: easy to describe
Unwanted situations: easy to describe

Figure 5.7: Exclusion Principle

5.3 Constraints

The conceptual model describes the concepts that are relevant in some universe of discourse and their relations. The semantics of a conceptual schema is the set of populations that can be associated with the schema.

The conceptual schema sofar describes the general structure of elementary sentences. This does not prohibit sentences that are unwanted for some reason. For example, consider the following population of the conceptual schema fragment from figure 5.8:

- Product with Name ”pluto” has Productcode 4123
- Product with Name ”trix” has Productcode 9231
- Product with Name ”country” has Productcode 4123

This population obviously is invalid! Constraints are used to exclude populations with unwanted features.

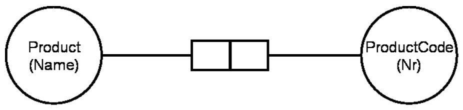


Figure 5.8: Sample schema fragment

The following constraints are very frequent, and have a special graphical notation.

- Uniqueness constraint
- Total role constraint

These constraints play a fundamental role during data with regards to the identity integrity of objects.

5.3.1 Unique role constraint

Reconsider the sample population of the schema fragment from figure 5.8. Although these sentences are well-formed, their combination does not make sense as product codes (usually) should uniquely determine the associated product. This restriction can be enforced by the following rule: *no product code may appear*

unique role constraint

more than once in the relation Coding. This rule is refer to as a *unique role constraint*, also referred to as uniqueness constraint.

A unique role constraint is denoted graphically by putting an arrow above the involved roles (for a horizontally drawn fact type), or an arrow next to the involved roles (for a vertically drawn fact type). I

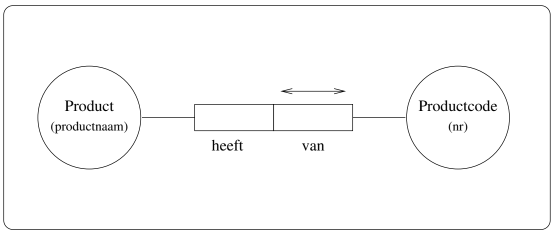


Figure 5.9. Adding a uniqueness constraint

A unique role constraint may involve more than one role, for example:

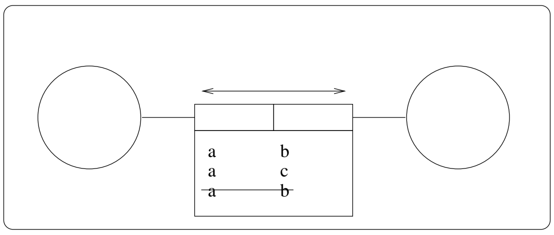


Figure 5.10. Restriction

In this example, an invalid tuple is stroken out. This example suggests a simple method to determine the existence of a unique constraint. The system expert offers the domain expert a significant sample population for validation. The domain expert responds by striking out a minimal number of tuples such that a correct population results. For example, suppose the system analyst responded as follows:

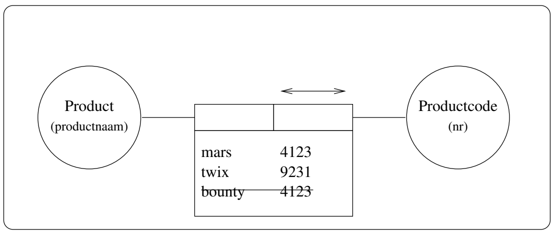


Figure 5.11. Excluded tuple

Then the system analyst concludes the existence of a unique role constraint.

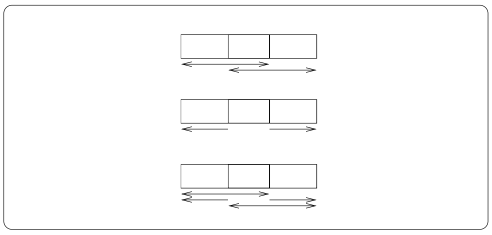


Figure 5.12. Multi-role uniqueness constraint

5.3.1.1 Formal interpretation

Consider the following uniqueness constraint:

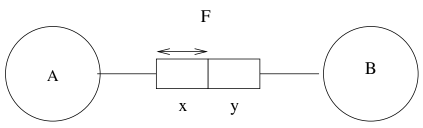


Figure 5.13. Partial mathematical function

A formal interpretation of a uniqueness constraint is the following: each population of fact type F is a partial mapping from A into B. Another way to put this is:

B is functionally dependent from A within F. This is denoted as $A \xrightarrow{F} B$.

The term functional dependency has been introduced in the relational datamodel ([**?**]. In ORC this constraint may be formulated as follows (see figure 5.14). Forbidden is: *a mapped to* b_1 *and a mapped to* b_2 *while* $b_1 \neq b_2$ so we require:

a mapped to b_1 and a mapped to $b_2 \Rightarrow b_1 = b_2$

In ORC:

- Find mapping buddies: B being map of A mapping to B
- These buddies should be equal: B being map of A mapping to B IMPLIES B

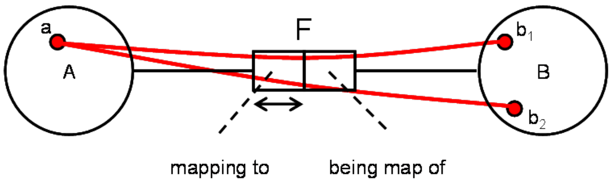


Figure 5.14: Unique role via a path

Lemma 5.3.1

The uniqueness constraint over the role with connector name *mapping to* is expressed as: B being map of A mapping to B IMPLIES B

Proof:

$\rightarrow$ Assume uniqueness constraint, then let $\langle \textbf{fr: } b_1, \textbf{to: } b_2 \rangle \in \mathscr{R}($B being map of A mapping to B$)$. Consequently for some a we have:

- $\langle \textbf{fr: } b_1, \textbf{to: } a \rangle \in \mathscr{R}($B being map of A$)$
- $\langle \textbf{fr: } a, \textbf{to: } b_2 \rangle \in \mathscr{R}($A mapping to B$)$

and from the uniqueness constraint we conclude $b_1 = b_2$, and therefore: $\langle \textbf{fr: } b_1, \textbf{to: } b_2 \rangle \in \mathscr{R}(B)$.

$\leftarrow$ Assume: B being map of A mapping to B IMPLIES B, and let $a \in$ Pop(A) have two different images $b_1, b_2 \in$ Pop(B). Then we conclude: $\langle \textbf{fr: } b_1, \textbf{to: } b_2 \rangle \in \mathscr{R}($B being map of A mapping to B$)$. And thus: $\langle \textbf{fr: } b_1, \textbf{to: } b_2 \rangle \in \mathscr{R}(B)$. Consequently: $b_1 = b2$.

5.3.2 Total role constraint

A frequent occurring requirement is that a role is mandatory to be played. In the above example, chances are that in the underlying universe of discourse there is a rule stating that each product should have a productcode. This kind of constraint is called a rotal role constraint.

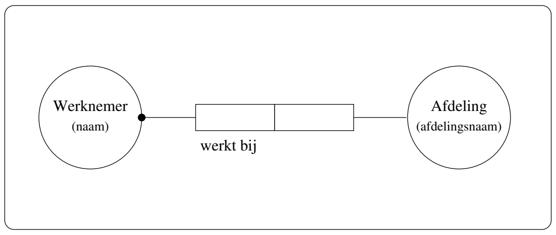


Figure 5.15. Sample schema fragment

In ORC

```
Cooperator IS ALSO working for Department
```

5.3.2.1 Mathematical interpretation

Combing unique and total role leads to the mathematical concept of function:

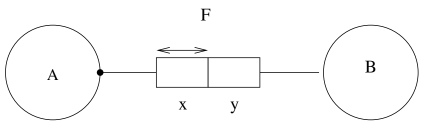


Figure 5.16. Mathematical function

In this case, relation F is a function from A to B.

5.4 Unique correspondence

A special combination of uniqueness and total role constraint is the following:

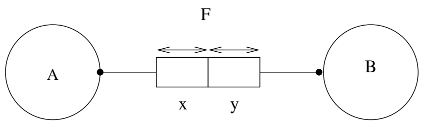


Figure 5.17. Bijection

In this case, F is a bijective function from A to B. As a consequence, the instances from A and from B can be identified with each other: each instance of A corresponds to a unique instance of B, and vice versa.

This situation occurs in bridge types. Identifying bridge types provide a bijection between the abstract and the concrete world. This is for example the case when we say that a Cooperator is determined by a Name.

5.5 The shadow property

The modeling method described in this book is based on the following starting points:

1. unique correspondence between universe of discourse and information base.
2. complete knowledge

The standard naming convention ensures a 1-1 correspondence between things and their names. Using the 1-1 correspondence of the bridge types that relates abstract instances with their standard names. we can conclude that a 1-1 correspondence between things and objetcs. This is displayed in figure 5.18.

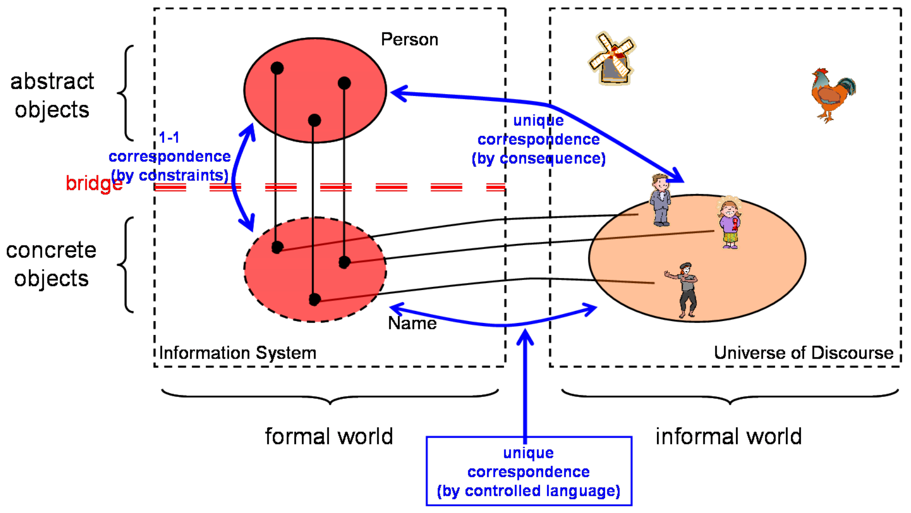

abstract objects
bridge
concrete objects
Person
1-1 correspondence (by constraints)
unique correspondence (by consequence)
Name
Information System
Universe of Discourse
formal world
informal world
unique correspondence (by controlled language)

Figure 5.18: Identification in ORM

5.5.1 Weak identification

We call a population weakly identified if the following holds:

if two instances in the population have the same properties, then these instances are the same.

As a consequence, each fact type has a uniqueness constraint covering all its roles.

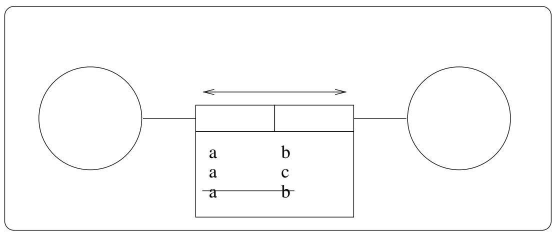


Figure 5.19. Ambiguous tuple

The reason is that both instances of the tuple $\langle a, b \rangle$ share (via a and b) the same properties.

Theorem 5.5.1
If there is a unique correspondence between the universe of discourse and a population, then this population is weakly identified.

Proof:

Suppose we have a population that makes a unique correspondence with the UoD. Then each object instance from this population has associated a unique *thing* from the UoD. If 2 object instances can not be discriminated from each other, then these objects have to correspond with the same ting from the UoD. Consequently, these object instances are equal.

In a weakly identified population, each object instance can be named uniquely by its (unique) combination of properties. An ad-hoc name is obtained by omitting those properties of each object instance that do not contribute to its unique naming. For example, if a population contains

1. a student with name ”Janssen” and age 19 that received for the course ”Domain Modeling” the mark 8,
2. a 18 years old student ”Janssen” with mark 8 for the course ”Storage and Retrieval”, and
3. a 19 years old student ”Pietersen” with mark 9 for the course ”Domain Modeling”.

Then this population is weakly identified. A sample set of ad-hoc names for these students is:

1. student with name ”Janssen” and age 19
2. student of age 18
3. student with name ”Pietersen”.

If a population is *not* weakly identified, then this population is called ambiguous.

5.5.2 Strong identification

A conceptual schema is called strongly identified if each admissible population is weakly identified. The advantage of this schema property is that no special need is required to maintain the property of unique correspondence with the universe of discourse.

A simple cure to prevent populations from being ambiguous is to require a unique standard name for each object instance. The reason is that all objects can be discriminated by their standard name.

Theorem 5.5.2

By using unique role and total role constraints strong identification can be enforced. This is equivalent with the requirement of a (unique) standard name for each object type.

A consequence is that the ORM Normalform requirement for sample sentences will guarantee that ORM schemata are strongly identified.

5.6 Relations between schemata

Two conceptual schemata Σ_1 and Σ_2 are called *equivalent*, notation $\Sigma_1 \equiv \Sigma_2$ if:

Each population of Σ_1 is also a population of Σ_2 and vice versa.

In that case the associated the associated information grammars are equivalent. In order to show that the schemata Σ_1 and Σ_2 are equivalent, it is sufficient to show that (1) each elementary transaction on Σ_1 can be reformulated as an elementary transaction on Σ_2 and (2) vice versa. An elementary transaction is a sentence of the form

ADD $\mathsf{S}_1, \mathsf{S}_2, \ldots, \mathsf{S}_n$

in which the omission of any sentence S_i would make the transaction invalid.

Note that a counter example is sufficient to prove that two schemata are inequivalent. So if we can construct a population of Σ_1 that is no valid population of Σ_2 then we may conclude that these schemata are not equivalent.

5.7 Schema transformation

A schema transformation is a rule to transform an conceptual schema into an equivalent alternative schema. As an example, we consider *object-to-role transformation*.

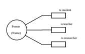


Figure 5.20. Several properties

In this schema fragment, the object type Person has associated 3 unary fact type with role names respectively (1) is student, (2) is teacher and (3) is researcher. A sample sentence is Person with Name "Janssen" is student. Another way to model this situation is as follows:

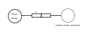


Figure 5.21. After object-to-role transformation

In this schema fragment the object type Person has associated a bridge type, relating it to a special enumerated label type with population {student, teacher, researcher}. This schema fragment also admits the sentence Person with Name "Janssen" is student. It is easy to see that both schema fragments accept the same sentences. As a consequence, these schema fragments are equivalent.

5.8 Frequently occurring constraints

5.8.1 Frequency constraint

The frequency constraint is used to constrain how many times a role may or must be played by objects from the associated object type. In the example below, it is required that each cooperator can be working for a number of projects that may range from 0 up to 3. On the other hand, each project should have a number of cooperators within the range 1..6.

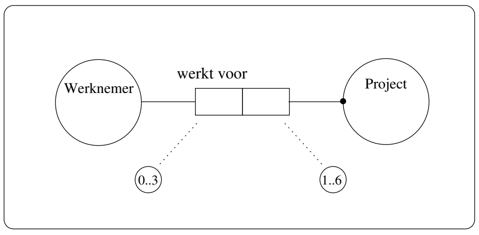


Figure 5.22. Occurrency frequency constraint

5.8.2 Binary heterogeneous constraints

The equality constraint is used to indicate that an object type has an equal participation in different fact types. Such a constraint is called a heterogeneous constraint. Consider the following schema fragment:

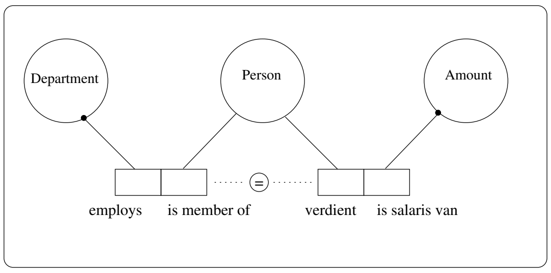


Figure 5.23. Relating related roles in different fact types

In the associated universe of discourse, each person that is employed by a department also has a salary, and vice versa. The constraint is represented by a small circle containing the symbol that corresponds with the constraint. In case of subset constraints, the connection with the involved roles should be such that left and right argument are clear from the diagram.

In ORC the constraint of the above example is expressed by the following rule:

```
being member of EQUALS earning
```

5.8.3 Partitioning heterogeneous constraints

Another kind of constraint is used to express a connection between any number of participations in different fact types. The following example shows that a person may not work in a project and also be the manager of that project:

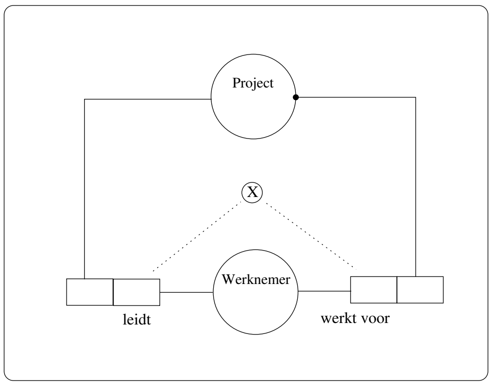


Figure 5.24. Exclusive membership for all instances

In ORC this is formulated as:

```
NO Cooperator is managing Project AND working for Project
```

The exclusion relation is represented by the times-symbol. A dot-symbol is used to denote a total heterogeneous requirement. These constraints may also be combined.

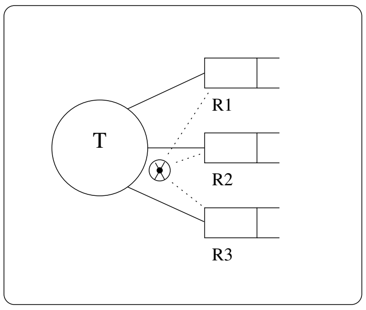


Figure 5.25. Role partitioning

In this case, each instance of object type T should play exactly one of the roles R1, R2 or R3.

5.9 Complex identification

In the cases sofar entity types were identified by a unique label type.

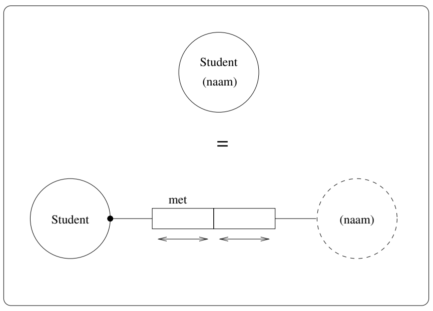


Figure 5.26. Simple bridge

There are situations were entity types require a more complex identification. Consider the following example. The identity of a house consists of the combination of street and street number. The street is assumed to have its own identification, whereas the street number is identified by a label value. Consider the following fragment:

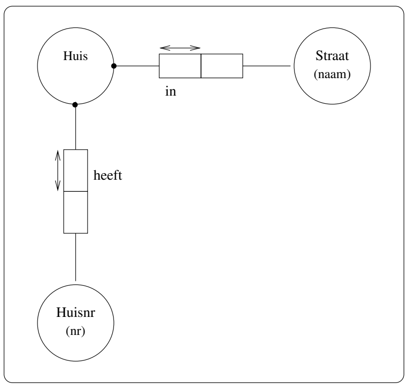


Figure 5.27. Sample schema fragment

In this schema fragment, each house has associated a unique street and a unique street number. Populations in which different houses have associated the same combination of street and street number are however possible in this schema. The following schema fragment adds a heterogeneous uniqueness constraint. This constraint expresses that combinations of street and street name should be unique.

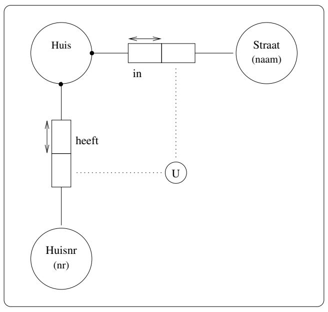


Figure 5.28. Unique combinations

Next we focus on the identification of streets as a unique combination of a street name and a city. This leads to:

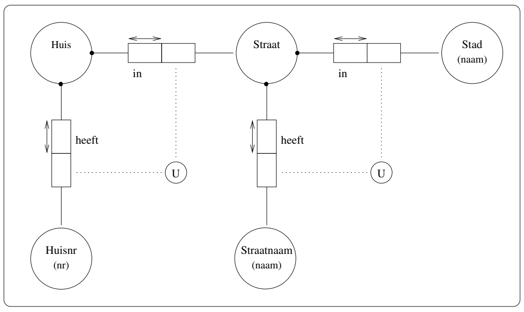


Figure 5.29. Complex Identification

5.10 Schema Quality

Validation of a conceptual schema is concerned with the question: *did we model the right universe of discourse?* This has to do with the relation between the conceptual schema and the universe of discourse. Another question is: *did we model the universe of discourse right?* This has to do with internal quality checks on the conceptual model.

5.10.1 Schema validation

In order to verify the validity of a conceptual schema, we consider questions like: Are all sample sentences processed in the conceptual schema?

- Are meta rules represented in the schema?
- Is the supplied sample population a valid population of the schema?

Besides, we check for suspicious constructs. For example, suppose the schema would contain the following fragment (taken from [**?**]):

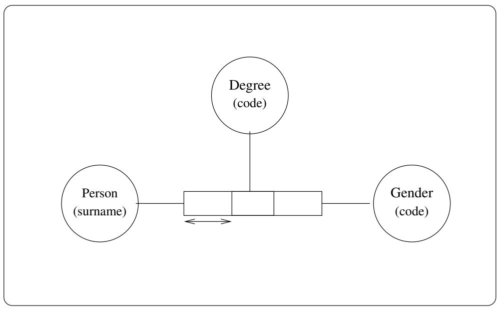


Figure 5.30. Sample schema fragment

A sample population of this schema is:

Person	Degree	Gender
Adams	BSc	M
Brown	BA	F
Collins	BSc	M

According to this schema, in the underlying universe of discourse a person with a degree also has a gender, and vice versa. From our intuitive knowledge of the universe of discourse, we recognize this as a dubious construct. By considering a sample sentence:

> Person with surname Adams seeks degree with code BSc and has gender with code M

we see that this sentence is not elementary, and should have been splitten into:

- Person with surname Adams seeks degree with code BSc
- Person with surname Adams has gender with code M

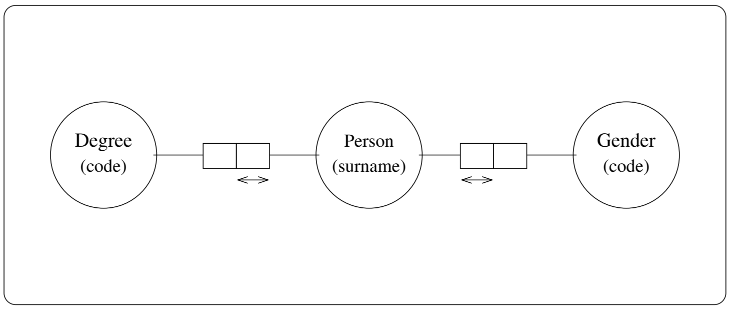


Figure 5.31. After splitting

Of course, the system analyst will check (by offering a sample population, or by a direct question) whether this universe of discourse has the rule that each person with a degree also has a gender, and vice versa. If this is confirmed by the domain expert, then there will be a heterogeneous equality constraint between these two roles of persons.

5.10.2 Key length check

This brings us to a general quality issue. The roles that constitute a uniqueness constraint within a fact type are also referred to as a key for that fact type. This term is inherited from technics to construct efficient storage techniques. From the example above, we derive the following rule:

> *If an n-ary fact type has a key over less than* $n-1$ *roles, then this fact type does not correspond to an elementary sentence, and must be split.*

In the above example, one could argue that the combination of degree and gender follows a special rule. In that case, splitting of the sentence would lead to loss of information. In relational database theory, this would be an example of a multi-valued dependency. In conceptual modeling the solution would be to make a special objectified fact type for the relation between degree and gender.

5.10.3 Schema verification

In order to check the conceptual schema on wellformedness we check for consistency between the various parts of the schema. A schema may contain contradictory parts. For example:

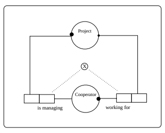


Figure 5.32. Sample schema fragment

the following rules have been added:

1. Each cooperator works on a project.
2. Each project has a supervisor.

The resulting schema is contradictory, it can not be populated. Generally spoken, a population check is a good mechanism to check for internal schema consistencies.

5.11 Definitions overview

[orm-us] –

Questions

Version: 01-12-2007

1. A company uses the following tables to administrate its store and sales.

   (a) Store:

Article	Description	Unit price	Size	Current storage	Minimal storage
B43	Jeans trouser	29,00	48	4	3
			50	7	5
			52	5	3
B47	Jeans trouser	34,00	50	4	2
			52	7	2
V02	Vest	34,00	50	3	2
			54	3	2
…	…	…	…	…	…
V21	Vaas	5,95		8	1

(b) Sales:

Date	Time	Article	Size	Number
05-03-2007	16:52h	B43	50	2
05-03-2007	16:52h	V21		1
06-03-2007	10:28h	B43	50	1
…	…	…	…	…

Answer te following questions

(a) What elementary sentences can be derived from the fact types that are enclosed in the tables above?

(b) Derive the conceptual schema from these elementary sentences.

(c) Add all constraints that may be assumed reasonably.

(d) Give the sample population of the conceptual schema corresponding to the above tables.

Elementary sentence types are:

- Artikel (Artikelcode) is van Artikelsoort.
- Artikel (Artikelcode) heeft stukprijs Euros.
- Artikel (Artikelcode) in Maat (Maatnr) heeft actuele voorraad Aantal (Nr).
- Artikel (Artikelcode) in Maat (Maatnr) heeft minimum voorraad Aantal (Nr).
- Artikel (Artikelcode) in Maat (Maatnr) heeft actuele voorraad Aantal (Nr).
- Artikel (Artikelcode) in Maat (Maatnr) heeft minimum voorraad Aantal (Nr).
- op Datum (Ddmmyy) om Tijdstip(Hhmm) is van Artikel (Artikelcode) Aantal (Nr) stuks verkocht.
- op Datum (Ddmmyy) om Tijdstip (Hhmm) is van Artikel (Artikelcode) in Maat (Maatnr) Aantal (Nr) stuks verkocht

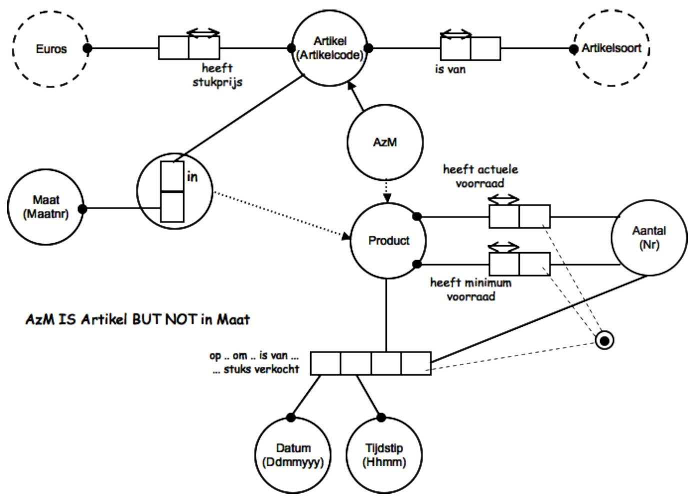


Figure 5.33: Conceptual schema for room reservation

Chapter 6

Object Grouping Mechanisms

Version: 01-01-07

In the previous sections we studied object types in isolation. In this section we will consider groupings of object types. We distinguish two main variants: homogeneous and heterogeneous groupings.

In this chapter we will frequently consider object types from the point of view of the roles they play. We will the rolename and the corresponding object type to denote a role. For example, in figure 3.9 a person can play the roles *being enrolled in Year* and *being graduated in Year*, but also the role *has Name*.

6.1 Subtyping

As a first example, consider the following groups of persons and their properties (i.e., the roles they can play):

1. Person:
   - identified by: has Id
   - has Name
   - has Birthdate
2. Student
   - identified by: has Id
   - has Name
   - has Birthdate
   - has Studentnumber
3. Philosophy student
   - identified by: has Id
   - has Name
   - has Birthdate
   - has Studentnumber
   - takes Philosophycourse

Note that the object types *Person*, *Student* and *Philosophy student* all have the same identification, but differ in the properties they have. So far, we have postulated that each object type should have a unique identification, as each thing from the application domain has assigned a unique group.

specialization

As can be seen in this example, each group is a further *specialization*. For example, a student is a special kind of person. For this special kind of person a student number also is administrated. Specialization is introduced as a mechanism to handle such cases.

From the example specialization relation we conclude that each student has person as main identity, and thus may be seen as a specialized version of a person. Furthermore, is should be clear when a person is also qualified as a student.

As another example, we consider a broker with two main sales lines: cars and houses. Although these are very different products, they also will have assigned a number of common properties by the broker. However, in ORM it is considered bad modeling practise to have the same fact type more than once in a conceptual schema. Generalization is introduced to cover such situations.

The following is an example:

1. Car:
   - identified by: has Chassisnumber
   - has Licenseplate
2. House
   - identified by: has Address
   - has Inhabitant
3. Brokerobject (1)
   - identified by: has Chassisnumber
   - has Licenseplate
   - has Price
4. Brokerobject (2)
   - identified by: has Address
   - has Inhabitant
   - has Price
5. Brokerobject (1+2)
   - has Price

generalization

Here we consider two object types of a different nature. However, both object types are treated 'equally' at the level where extra properties (having a price) are assigned. This is a heterogeneous form of grouping. The group can be seen as a *generalization* (abstraction) where the original identity of objects is not considered.

subtyping

Specialization and generalization may be seen as two variants of *subtyping*. In both cases we have object types that are related via a subset relation on their population. As a consequence, object instances may have related more than one object type. For example, a philosophy student in the above example has related 3 object types: (1) Philosophy student (2) Student (3) Person

type related

We will call object types A and B of schema Σ type related, denoted as $\mathsf{TypeRelated}(A,B)$, if their populations *may* share object instances. This means that in the set $\mathscr{M}(\Sigma)$ of valid schema populations (see section 5.2) there exists a population P such that $P(A) \between P(B)$. Note that this relation is a reflexive and symmetric relation.

Type relatedness has a special impact on the optimization of the evaluation of ORC expressions:

Lemma 6.1.1
Let D_1 and D_2 be ORC expressions, then $\neg\mathsf{TypeRelated}(\mathsf{end}(\mathsf{D}_1),\mathsf{start}(\mathsf{D}_2)) \;\Rightarrow\; \mathsf{D}_1\,\mathsf{D}_2 \equiv \varnothing$

As a consequence, each instance x in a population P has assigned a (non-empty) set $\mathsf{Types}(x|P)$ of object types. Then obviously:

Lemma 6.1.2
$A,B \in \mathsf{Types}(x|P) \Rightarrow \mathsf{TypeRelated}(A,B)$

We will call object instances x and y type related, denoted as TypeRelated(x,y) if (in population P) they share object types: Types$(x|P) \between$ Types$(y|P)$. These shared object type obviously are type related. The restriction is a consequence of the type relatedness relation *not* being transitive.

Lemma 6.1.3

$$\text{TypeRelated}(x,y|P) \Rightarrow \forall_{A,B}\left[A,B \in \text{Types}(x|P) \bigcap \text{Types}(y|P) \Rightarrow \text{TypeRelated}(A,B)\right]$$

During analysis, we start from type relateness between object instances (the examples provided) and derive typer relatedness between object types. This will be elaborated in section 7.2.

6.2 Specialization

As an example we consider the following significant population:

Cooperator	Sort	Department	Lease car	Bus pas
Janssen	S	Personnel	10-GY-RZ	-
Pietersen	J	Sales	-	128123
Dekker	J	Personnel	-	389212
Klaassen	S	Sales	06-SX-LN	-

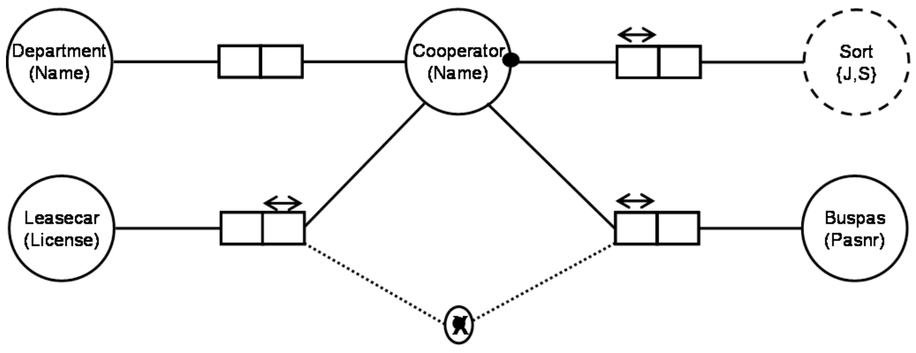
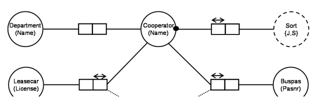


Figure 6.1: Programming language history

This population (apparently) describes 2 sorts of cooperators, senior and junior cooperators. Senior cooperators and junior cooperators have different properties. The following schema fragment is compatible with this sample population:
An extra constraint is required to enforce the difference between senior and junior cooperators. For example: Person having Sort 'S' IS EQUAL TO using Leasecar.

This semantic issue of the universe of discourse may also be modeled directly in the conceptual schema by applying an inverse object-role-transformation, leading to the following schema:



rag-

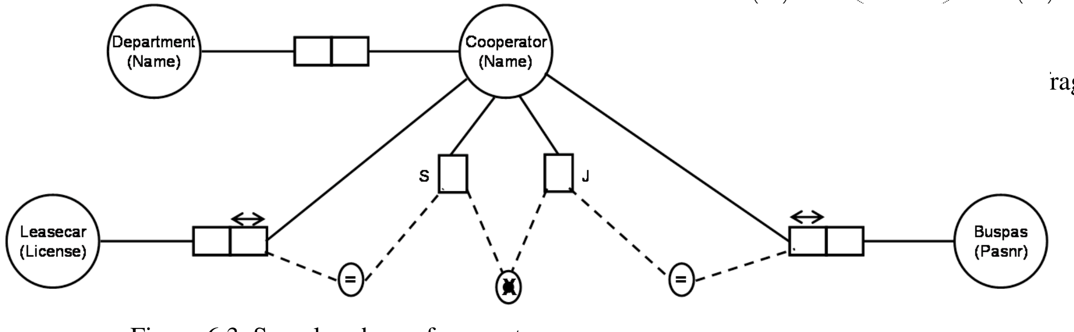


Figure 6.3. Sample schema fragment

We might as well identify the unary fact types by the subsets they induce. This leads to the following variant to model this universe of discourse:

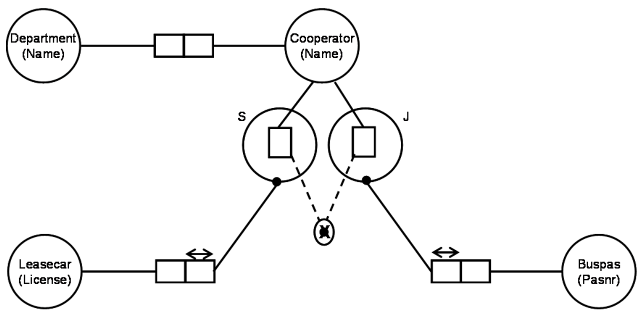
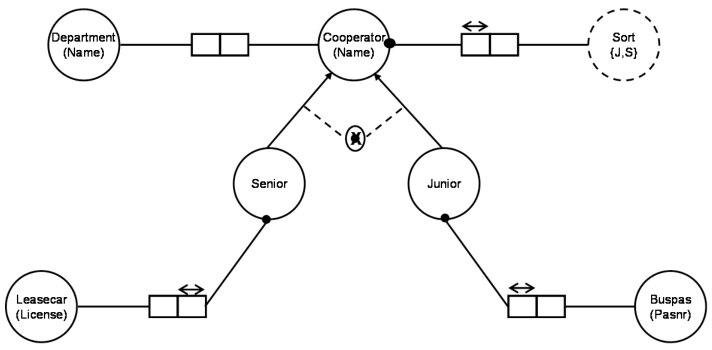


Figure 6.4. Sample schema fragment

Specialization in ORM is a mechanism that allows the introduction of specializations directly as follows::



Subtype defining rules:

Senior IS Cooperator with Sort 'S'
Junior IS Cooperator with Sort 'J'

Figure 6.5. Using subtyping

subtype determining rule

Each subtype has associated a *subtype determining rule*. This rule describes what objects of the super type are member of the subtype. In the example above, we see that a Senior is defined as:

Senior IS Cooperator having Sort "S"

The subtype determining rule is tested on instances of the immediate supertype. As a consequence, the subtype determining rule may only involve properties of that supertype. As subtype relation is graphically depicted by an arrow:

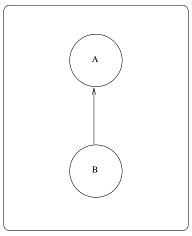


Figure 6.6. Subtype relation

In this schema, B is a subtype of A with subtype determining rule R. The population of a subtype is derived from the population of the supertype by applying the subtype defining rule:

$$\begin{aligned}\mathsf{Pop}(B) &= Pop(a)[R] \\ &= \left\{x \in \mathsf{Pop}(A) \,\middle|\, R(x)\right\}\end{aligned}$$

6.2.1 Multiple parents

A object type may be a subtype of more than one supertype. In this case, the membership condition is the conjunction of the membership rules associated with that subtype. For example:

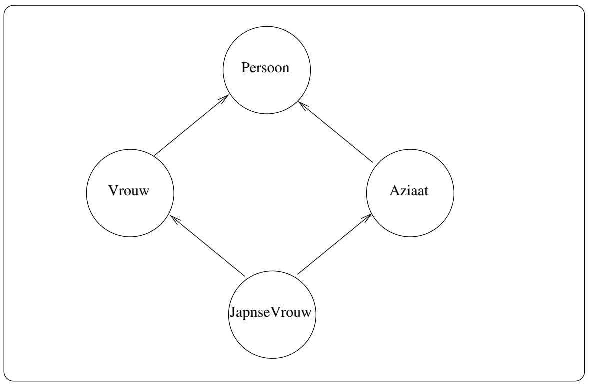


Figure 6.7. Multiple inheritance

In this example, Japanese females are persons that are both females satisfying rule FA and asians satisfying rule AA.

$$\begin{aligned}\mathsf{Pop}(\text{JapaneseFemale}) &= \mathsf{Pop}(\text{Female})[\text{FA}] \bigcap \mathsf{Pop}(\text{Asian})[\text{AA}(x)] \\ &= \left\{x \in \mathsf{Pop}(\text{Female}) \,\middle|\, \text{FA}(x)\right\} \bigcap \left\{x \in \mathsf{Pop}(\text{Asian}) \,\middle|\, \text{AA}(x)\right\}\end{aligned}$$

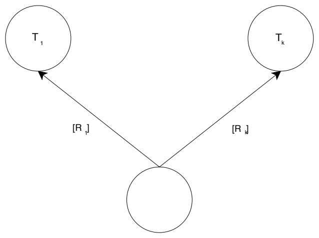


Figure 6.8. Subtype with multiple parents

In general, let S be subtype of object types $T_1, \ldots, T_k$ according to subtype determining rules T_1 IS $\mathsf{R}_1, \ldots, T_k$ IS R_k respectively. Then

$$\mathsf{Pop}(S) = \bigcap_{i=1}^{k} \mathsf{Pop}(T_i)[R_i]$$

So for example, if *Lai* is instance of object type *Person*, then *Lai* will also be instance of subtype *Female*, and of subtype *Asian*. So *Lai* is candidate to pass the tests *FA* and *AA* for membership for subtype *JapaneseFemale*.

pater familias

Generally, a subtype hierarchy should be a directed acyclic graph with a unique top element. The top is referred to as the *pater familias* of that hierarchy. The pater familias is the object type determines the identity of all object types in the hierarchy.

6.2.2 Specialization handling in ORC

Subtypes may be treated in ORC just like other object types (see section 3.4.1). A more effective approach is to use the subtype determining rule to involve the semantics of the subtype.

Let S be subtype of object types $T_1, \dots, T_k$ according to the following subtype determining rules T_1 IS $\mathsf{R}_1, \dots, T_k$ IS R_k respectively. Then the meaning of information descriptor $\mathsf{ONm}(S)$ is defined as:

$$\mathscr{R}(\mathsf{ONm}(S)) = \mathscr{R}(\mathsf{R}_1 \ \text{AND ALSO} \ \dots \ \text{AND ALSO} \ \mathsf{R}_\mathsf{k})$$

The AND ALSO operator expresses the conjunction between the diverse inheritances.

6.3 Generalization

As an example, we consider the following subtype situations:

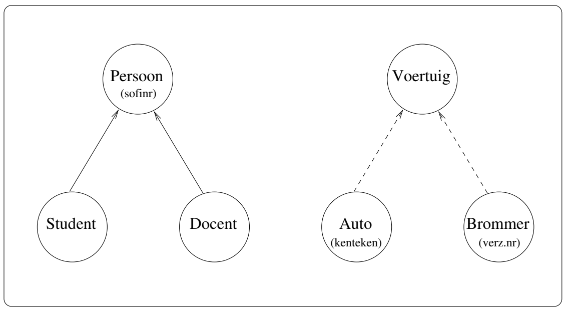


Figure 6.9. Different kinds of subtypes

The left example has the object type *Person* as pater familias, and specifies two special kinds of persons: *Student* and *Teacher*. The right example is rather different. It introduces the object type *Vehicle* as a generalization of the object types *Car* and *Motorbike*. Obviously, cars and motorbikes have a different nature, and will be identified differently. As a consequence, it does not make sense to see a car as a special kind of vehicle; in that case a car would be identified as a vehicle!

mixed identity

The difference between specialization and generalization is the inheritance of identity. In a specialization, the subtype inherits its identity form the supertype. In a generalization, the generalized object type inherits its identity from its specifiers. As a consequence, a generalized object is an object type of mixed identity. The generalization hierarchy is graphically represented by dotted arrows.

A generalization hierarchy should be free of identity ambiguity. This means that type related object types can not be generalized. In such a case, a specialization would be more appropriate.

Formally, let $D_1, \dots, D_k$ be a set of object types that are mutually *not* type related. Let G be the generalization of these object types, then we have:

$$\mathsf{Pop}(G) = \bigcup_{1=i}^{k} \mathsf{Pop}(D_i)$$

We call $D_1, \dots, D_k$ the specifiers of object type G. Generalizations are useful when different object types do play similar roles. Generalization is a mechanism to avoid schema redundancy, as it provides the opportunity to introduce a generic object type describing the common roles that can be played by its specifiers.

specifier

6.3.1 Generalization handling in ORC

Let G be the generalization of specifiers $D_1, \ldots, D_k$. Then the name $\mathsf{ONm}(G)$ of object type G is interpreted in ORC as follows:

$$\mathscr{R}(\mathsf{ONm}(G)) = \mathscr{R}(\mathsf{D_1} \text{ OR OTHERWISE } \ldots \text{ OR OTHERWISE } \mathsf{D_k}$$

6.4 Dynamic Grouping

identification by enumeration

Another special case is when the dynamics of grouping conflict with the statics of the subtyping mechanism. This is the case when a group of object instances can have its own identity. For example, a school class could be partitioned in anonymous groups to work on exercises. The groups do not have special properties, except that the group is assigned a score for that exercise. Thus the group thus may only be identified by *enumerating its members*.

We will discuss two methods to handle this particular kind of instance grouping.

1. via a special kind of object type, the so-called set type
2. via a special kind of constraint, the so-called *extensional uniqueness constraint* that is coupled to the rules for object identification.

6.4.1 Set type

Set type

First we discuss the *set type* as a special constructor for objects. The following schema fragment models this situation:

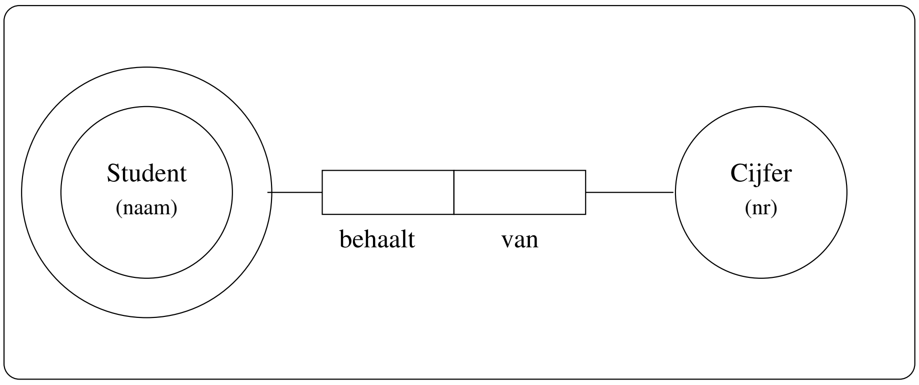


Figure 6.10. Set type

base type

The set type is drawn as a circle around its associated *base type*. With each set type we assume an implicit fact type *Contains*. Usually this fact type is left implicit, and left out of the conceptual schema.

The set type over some base object type is populated by sets of instances from that base object type, so:

$$x \in \mathsf{Pop}(\text{Group}) \Rightarrow x \subseteq \mathsf{Pop}(\text{Student})$$

The instances of a set type are identified by the members of the set. So instances of a set type are different if and only if they have associated different members.

A typical ORNF sentence associated with this situation is:

Group consisting of Student with Name ”John” and Student with Name ”Mary” gets Mark 10

6.4.2 Extensional uniqueness constraint

Identification by enumeration is also expressed by a special kind of uniqueness constraint. Instances of set types are identified by their members. In figure 6.4.2 the set *Group* is modeled via the fact type *Contains*. The special identification of the set is enforced by the extensional uniqueness constrained on the role *in* of this fact type.

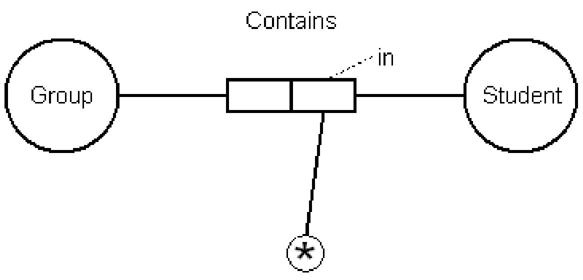


Figure 6.11. Different kinds of subtypes

The meaning of this constrains is as follows. For each population of this schema (fragment) it should hold that:

$$\forall_x [x \text{ in } g \Leftrightarrow x \text{ in } h] \;\Rightarrow\; g = h$$

Note that this usage of the extensional uniqueness constraint is equivalent with saying that *Group* is a set type having *Student* as its base type.

6.4.3 Set types in ORC

A set type may be addressed by its name, as any other object type. In order to verbalize the element-set relation, the special keywords IN and CONTAINING can be used. For example, the following ORC expression asks for the groups in which student Jan is participating:

```
Group CONTAINING Student with Name ”Jan”
```

The partners of this student are obtained by:

```
Name of Student IN Group CONTAINING Student with Name ”Jan” BUT NOT ’Jan’
```

6.4.4 Constraints on set types

ORM provides special constraints to regulate the membership relation on sets. For example, the following schema restricts membership to a dual membership:

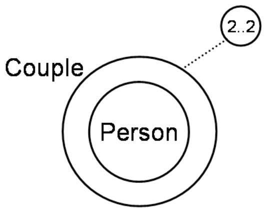


Figure 6.12. Description of couples

6.5 Definitions overview

[orm-us] –

Questions

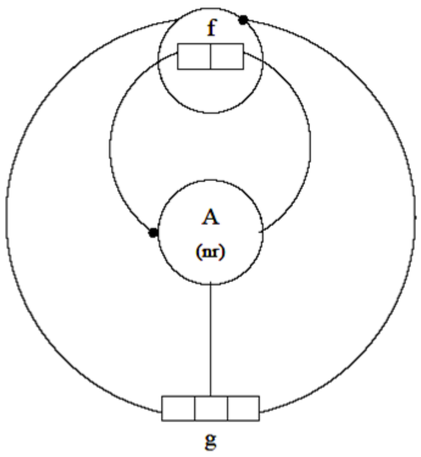


Figure 6.13: Schema for exercise 1

1. Is the schema from figure 6.13 populatable?
2. Express the relationship between the following kinds of constraints:
   (a) total role and frequency constraint
   (b) unique role and frequency constraint

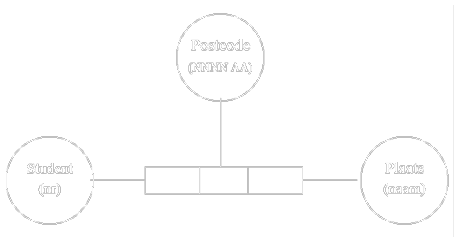


Figure 6.14: Missing dependencies

3. Schrijf JapanesFemale als $\{x \in Person \mid F(x)\}$.
4. Is it possible that a schema is strongly identified but not populatable?
5. What is wrong in the following schema from figure 6.15?
6. Give an example of fact types for each of the following kinds of behavior:
   (a) reflexive
   (b) symmetric
   (c) transitive

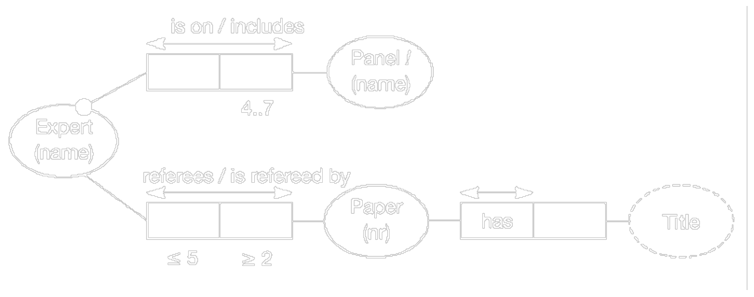


Figure 6.15:

(d) irreflexive

(e) asymmetric

(f) antisymmetric

Chapter 7

The Modeling Algorithm

Version: 01-01-07

The way to go about to construct a conceptual model in practical situations is called CSDP (Conceptual Schema Design Procedure, see [**?**]). In this chapter we will discuss the modeling algorithm from a more formal point of view, and focus on a special part, and discuss how from from a significant set of sample sentences, for example resulting from input tables provided by the domain expert, a conceptual framework can be generated. We will also explicitly point out where (extra) input from the domain expert is required.

7.1 Sharing properties

Things from the universe of discourse are are usually organized in groups of thing s with similar properties and/of behavior. Things that are represented in the information system to be associated with this universe, will reflect this similarity. Thus the objects corresponding to the things of the universe of discourse will have similar properties, which is reflected in the the roles they can play. Note that some role(s) are used to identify the thing, and (thus) also to identify the associated object instance within the information system.

We start our analysis from a sample population, although in practise the specification of the universe will also be expressed on a more global level as domain rules. Let $x_1, \ldots, x_n$ be the object instances derived from the sample population, and $R_1, \ldots, R_n$ the sets of roles they play respectively. Consider objects x_i and x_j, then we have the following cases:

Case 1: unrelated roles: $R_i \between R_j$.
Then x_i and x_j are not directly type related.

Case 2: related roles: This is denoted as $x_i \sim_1 x_j$.
There are two possibilities:

1. Sharing of identifying role:]
   Then x_i and x_j are type related. This will lead to a specialization relation if the property differences are structural.
2. *Not* sharing identifying role:]
   There is a generalization relation that is used to describe generic properties.

Note that the relation $\sim_1$ is reflexive and symmetric, but *not* transitive.

role sharing clusters

Let $\sim$ be the transitive closure of $\sim_1$, then $\sim$ is an equivalence relation. So we may split up the object instances $\{x_1, \ldots, x_n\}$ in *role sharing clusters*, i.e., classes of object instances that have a direct or indirect role relationship. Note that as an effect of the transitive closure, unconnected instances may end up being type related. For example, suppose a sales organization sells both cars and houses. Some cars and houses share the role *having Price*, while unpriced houses and cars do not share any property. Due to transitivity, these cars and houses will fall within the same role sharing cluster. [1]

As a consequence, we are grouping sample object instances according to their potential of sharing roles. However, the modeling algorithm we will discuss does not require such a grouping. Grouping though makes the algorithm more efficient.

As an example, we consider the following table:

	has Name	represents Country	born in Country
x_0	Ann Arbor	USA	USA
x_1	Bill Arbot	UK	-
x_2	Chris Lee	USA	NZ

The information within this table may be described by the following kind of elementary sentences:

1. Athlete with Name ”Ann Arbor” represents Country ”USA”
2. Athlete with Name ”Ann Arbor” is born in Country ”USA”

The role sharing clusters are easily centered around the object types:

1. Athlete, playing roles:
   - with Name
   - represents Country
   - is born in Country
2. Name, playing roles:
   - naming Athlete
3. Country, playing roles:
   - is represented by Athlete
   - is birthplace of Athlete

7.2 A formal algorithm

Given a role sharing cluster, the following steps then are taken:

1. Group sample sentences and determine the deep structure sentences
2. Select object references
3. Select associated roles
4. Build corresponding object-role context
5. Perform formal concept analysis
6. Post-process concept lattice into subtype hierarchy
7. Group roles according to sentence type

The first 4 steps are called the initial steps. The result of this algorithm is a conceptual (ORM) schema, including a subtyping structure and total role constraints.

[1]See lemma

7.3 Simple homogeneous case

We start with the following example taken from Halpin ([**?**]), and modified slightly to suit our needs.

7.3.1 The initial steps 1 to 4

role significant population

In this example, from the sample sentences provided, we derive 9 instances playing relates roles, and thus fall within case 2 or 3 from previous section. The following pattern for the properties of these instances is derived from the sample sentences. From the domain expert we know that this is *a significant population with respect to the variety of playing roles*, or *role significant population* for short. A role is denoted with special emphasize when it is an identifying role. In the example below, *has Id* is the identifying role for object type *Person*.

	has Id	has Age	watches tv NrHours	reads paper NrHours	has favorite Channel	has favorite Paper	prefers News
	R_1	R_2	R_3	R_4	R_5	R_6	R_7
x_1	5001	41	0	10		The Times	
x_2	5002	60	0	25		The Times	
x_3	5003	16	20	2	9	The Times	
x_4	5004	18	20	5	2	Daily Mail	TV
x_5	5005	13	35	0	7		
x_6	5006	17	14	4	9	Daily Sun	
x_7	5007	50	8	10	2	Daily Sun	NP
x_8	5008	33	0	0			
x_9	5008	13	50	0	10		

property pattern

On the fly we introduced an abbreviated notations for the roles in this example. We see that some property patterns occur more than once, and thus may be eliminated. We restrict this table to the various property patterns:

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$G_1 = \{x_1, x_2\}$	1	1	1	1	0	1	0
$G_2 = \{x_3, x_6\}$	1	1	1	1	1	1	0
$G_3 = \{x_4, x_7\}$	1	1	1	1	1	1	1
$G_4 = \{x_5, x_9\}$	1	1	1	1	1	0	0
$G_5 = \{x_8\}$	1	1	1	1	0	0	0

This table describes how patterns and roles are related. A value 1 in a cell of the table states that the property is recorded in the associated property pattern, the value 0 is used when this is not the case. As such, we may see this table as describing a relation σ between the set $\mathscr{G} = \{G_1, \ldots, G_5\}$ of property patterns and the set $\mathscr{R} = \{R_1, \ldots, R_7\}$ of properties. We will write $\sigma(g, r)$ when property r is recorded in property pattern g.

7.3.2 Step 5: Formal concepts

formal concept

In Formal Concept Analysis (FCA) the structure $\langle G, R, \sigma \rangle$ as described in the previous subsection, is called a *formal context*. Formal concepts are derived from this context by using the following operators. The first operator, ComProps, takes a set of property patterns as its input, and determines what properties they have in common:

$$\mathsf{ComProps}(G) = \{ r \in \mathscr{R} \mid \forall_{g \in G} [\sigma(g, r)] \}$$

On the other hand, given a set R of roles, $\mathsf{ComPats}(R)$ denotes the property patterns that record all roles from R:

$$\mathsf{ComPats}(R) = \{g \in \mathscr{G} \mid \forall_{r \in R} [\sigma(g,r)]\}$$

Some examples are:

- $\mathsf{ComProps}(\{G_1, G_2\} = \{R_1, R_2, R_3, R_4, R_6\}$
- $\mathsf{ComProps}(\{G_5\} = \{R_1, R_2, R_3, R_4\}$
- $\mathsf{ComProps}(\mathscr{G}) = \{R_1, R_2, R_3, R_4\}$
- $\mathsf{ComProps}(\varnothing) = \mathscr{R}$
- $\mathsf{ComPats}(\mathscr{R}) = \{G_3\})$
- $\mathsf{ComPats}(\varnothing) = \mathscr{G}$
- $\mathsf{ComProps}(\{G_1, G_2, G3\}) = \{R_1, R_2, R_3, R_4, R_6\}$
- $\mathsf{ComPats}(\{\{R_1, R_2, R_3, R_4, R_6\}\} = \{G_1, G_2, G_3\}$

The latter two examples demonstrate the special case, where a set G of property patterns and a set R of roles are associated as follows:

$$\mathsf{ComProps}(G) = R$$
$$\mathsf{ComPats}(R) = G$$

Such a combination $C = (G, R)$ is called a *formal concept*. Formal concepts are important in many application fields. Also for the modeling activity these formal concepts will show their usefulness as we will discuss in this chapter. G is called the support of concept C, while R is its specification. G is also called the extension of C, and R its intention.

formal concept

7.3.3 The concept lattice

Each formal context has associated a set of formal concepts. In our example, we have the following concepts:

1. $C_1 : (\{G_1, G_2, G_3\}, \{R_1, R_2, R_3, R_4, R_6\})$
2. $C_2 : (\{G_2, G_3\}, \{R_1, R_2, R_3, R_4, R_5, R_6\})$
3. $C_3 : (\{G_3\}, \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\})$
4. $C_4 : (\{G_2, G_3, G_4\}, \{R_1, R_2, R_3, R_4, R_5\}$
5. $C_5 : (\{G_1, G_2, G_3, G_4, G_5\}, \{R_1, R_2, R_3, R_4\}$

Concepts may be better supported or may be more specific. For example, concept C_2 is more specific than concept C_1 because concept C_2 adds an extra property to concept C_1. In general, a concept $C = (G, R)$ is more specific than concept $D = (H, S)$ if $S \subseteq R$. An equivalent formulation is: concept $C = (G, R)$ is more specific than concept $D = (H, S)$ if $G \subseteq H$.

7.3.4 Step 6: Constructing the subtype hierarchy

It is useful to construct a line diagram that presents the concepts where the lines represent the relation *being more specific*. For our example we get:

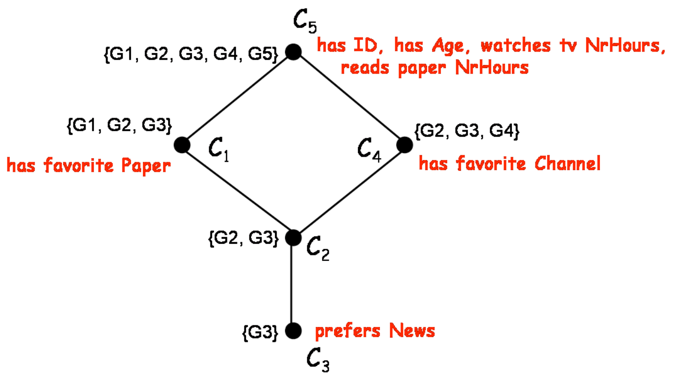


Figure 7.1. The concept diagram

We may read this diagram as follows. All object instances considered play the roles $\{R_1, R_2, R_3, R_4\}$. The object instances associated with concept C_2 play all roles inherited by concept C_1, and besides also role R_5. A special case is related to the concepts C_2 and C_3. Together they represent the notion of possibly playing role R_7. Combining these insights, we group concepts into candidate object types as shown in figure 7.3.4.

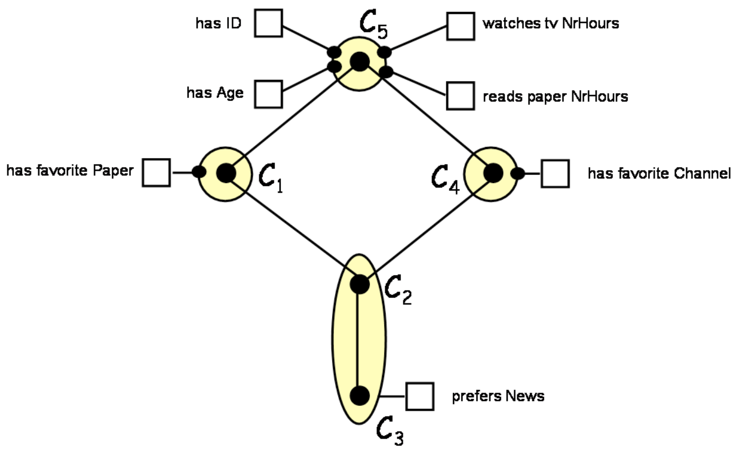


Figure 7.2. The concept diagram

We propose the candidate specialization hierarchy to the domain expert for acceptance, and trying to find proper names for the specializations introduced. The specialization determining rule basically is derived from the commonality of the attributes of the associated concept. For example, the concept C_1 has associated groups G_1, G_2, G_3, and thus the object instances $x_1, x_2, x_3, x_4, x_6.x_7$:

	has Id	has Age	watches tv NrHours	reads paper NrHours	has favorite Channel	has favorite Paper	prefers News
	R_1	R_2	R_3	R_4	R_5	R_6	R_7
x_1	5001	41	0	10		The Times	
x_2	5002	60	0	25		The Times	
x_3	5003	16	20	2	9	The Times	
x_4	5004	18	20	5	2	Daily Mail	TV
x_6	5006	17	14	4	9	Daily Sun	
x_7	5007	50	8	10	2	Daily Sun	NP

We might notice (or at least the domain expert should tell the system analyst) that these precisely are those instances with $R_4 > 0$. We assume this leads to the following names with their associated subtype determining rules:

1. Journalreader IS Person reading newspaper Hours > 0
2. Televisionwatcher IS Person looking at television Hours > 0
3. Newsreader IS Journalreader AND Televisionwatcher

The result is a specialization hierarchy. Each role also has associated its potential total role constraint.

7.3.5 Step 7: Composing the conceptual schema

Next we complete the resulting schema and find the following conceptual schema:

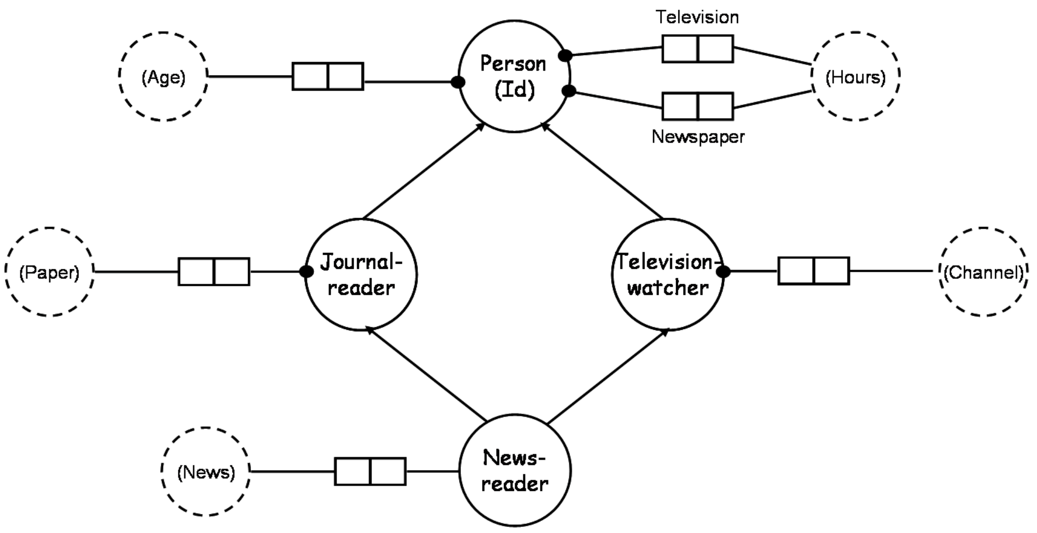


Figure 7.3. The concept diagram

7.4 Transformation example

As a next example, we consider the following sample table (also taken from [?]):

	has Name	has Sex	is smoker	has Starsign	has Height	has Office	plays Sport
A	Adams	F	Y	Aquarius	-	-	-
B	Brown	M	Y	-	170	-	-
C	Collins	M	N	-	190	308	basketball
D	Davis	F	Y	Gemini	-	-	-
E	Evans	M	Y	-	190	-	-
F	Fomor	M	N	-	180	505	basketball, tennis
G	Gordon	F	N	Aquarius	-	406	-
H	Hastings	M	Y	-	165	-	-
I	Iveson	M	N	-	165	305	-
J	Jones	M	N	-	179	502	-

This table shows a role significant population for persons. First we will do a direct analysis. Then we add some extra information of the domain expert on the meaning of attribute values, and show how this is handled to form a more detailed specialization schema.

We se from this table that the role *plays Sport* lists all sports that are played by that person. After validation with the domain expert, this combination of sports is known to have not a conceptual meaning. It will therefor be treated as a single-valued attribute, and not as an attribute to be modeled using a *set type*.

7.4.1 The direct approach

The table presents the role *plays Sports* as a multi-valued role. We will interpret this as a normal role that is played more than once. Applying the method of the previous section then leads to the following property pattern table:

	has Name	has Sex	is smoker	has Starsign	has Height	has Office	plays Sport
$G_1 = \{A, D\}$	1	1	1	1	0	0	0
$G_2 = \{B, E, H\}$	1	1	1	0	1	0	0
$G_3 = \{C, F\}$	1	1	1	0	1	1	1
$G_4 = \{G\}$	1	1	1	1	0	1	0
$G_5 = \{I, J\}$	1	1	1	0	1	1	0

The following concepts are generated from this table:

1. $C_0 = (\{G_1, G_2, G_{3,G}4, G_5\}, \{$has Name, has Sex, is smoker$\})$
2. $C_1 = (\{G_1, G_4\}, \{$has Name, has Sex, is smoker, has Starsign$\})$
3. $C_2 = (\{G_2, G_3, G_5\}, \{$has Name, has Sex, is smoker, has Height$\})$
4. $C_3 = (\{G_3, G_4, G_5\}, \{$has Name, has Sex, is smoker, has Office$\})$
5. $C_4 = (\{G_3, G_5\}, \{$has Name, has Sex, is smoker, has Height, has Office$\})$
6. $C_5 = (\{G_3\}, \{$has Name, has Sex, is smoker, has Height, has Office, plays Sport$\})$
7. $C_6 = (\{G_4\}, \{$has Name, has Sex, is smoker, has Starsign, has Office$\})$
8. $C_7 = (\{\}, \{$has Name, has Sex, is smoker, has Starsign, has Height, has Office, plays Sport$\})$

These concepts lead to the following concept lattice:

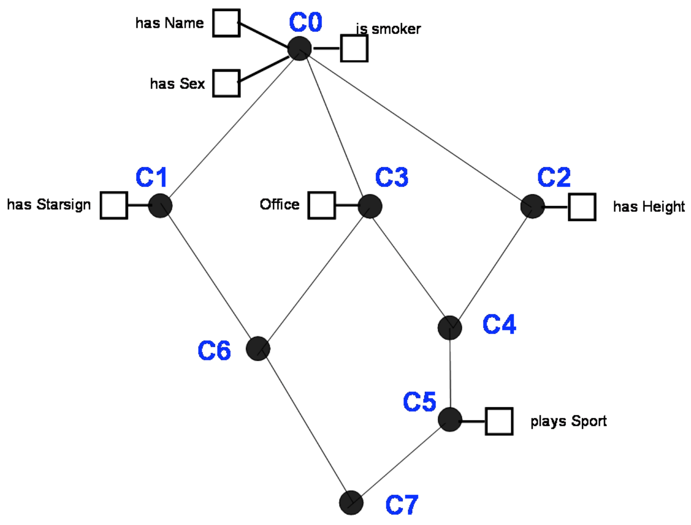


Figure 7.4. The concept lattice

Next we transform the concept lattice into a conceptual schema. The candidate concepts resulting from the grouping are displayed in the following figure:

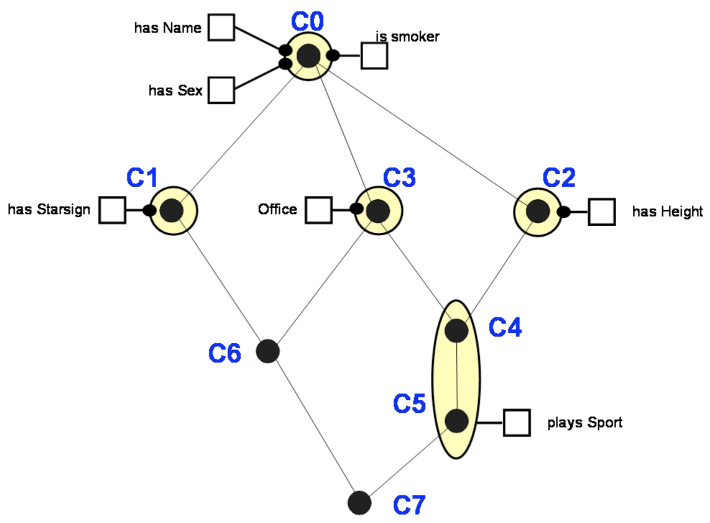


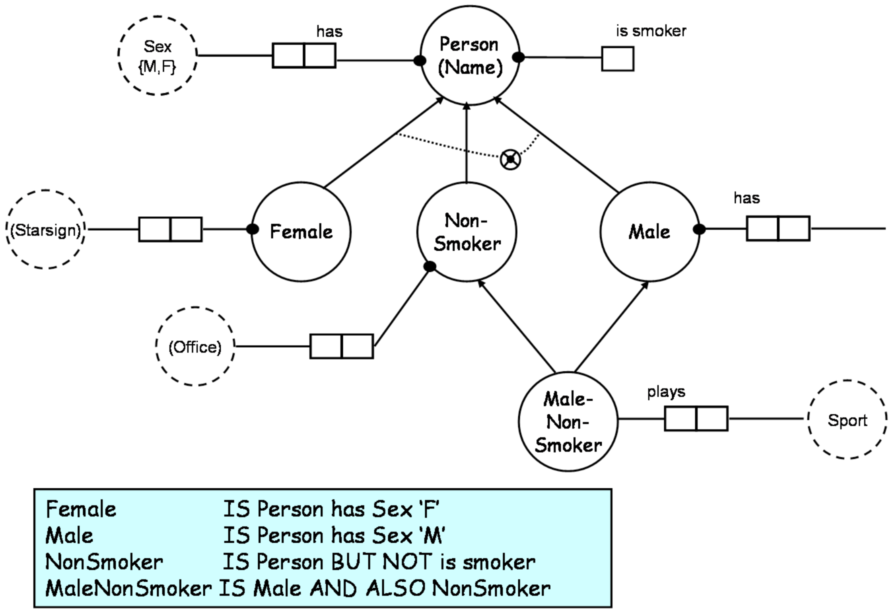
Figure 7.5. The concept lattice

The property patterns associated with concept C_1 suggest that this candidate concept may be named *Female*. This has to be validated by the domain expert! Analogously, concept C_2 is named *Male*, while C_3 is referred to as *NonSmoker*. We also conclude, as we have a *significant* population, that specializations *Female* and *Male* are total and disjoint. The candidate object type consisting of C_4 and C_5 is obviously named *MaleNonSmoker*.

Note that the concepts C_6 (the female non-smokers) and C_7 will not result in object types. Both concepts have no roles associated, and thus need not be present in the conceptual schema. Besides, the bottom concept C_7 would lead to an object type that is *structurally empty* as there are no persons that are both female and male. This leads tot the following specialization structure:

1. Female IS Person has Sex ”F”.
2. Male IS Person has Sex ”M”.
3. NonSmoker IS Person BUt NOT is smoker.

This leads to the following conceptual schema:



Female	IS Person has Sex ‘F’
Male	IS Person has Sex ‘M’
NonSmoker	IS Person BUT NOT is smoker
MaleNonSmoker	IS Male AND ALSO NonSmoker

Figure 7.6. The resulting conceptual schema

7.4.2 Qualifying attributes

Label values are concrete attributes of abstract object types. The following kind of attributes are distinguished:

Nominal

An attribute (domain) is called nominal it its values are unordered. The only operators for these values are equality tests:

distinctness	$=$	$\neq$

Examples: ID numbers, eye color, zip codes

Ordinal

An attribute (domain) is called ordinal if its value are ordered. The following operators are available for such domains:

distinctness	$=$	$\neq$
order	$<$	$>$

Examples: rankings (e.g., taste of potato chips on a scale from 1-10), grades, height in tall, medium, short

However, there is no distance measure between value.

Interval

An attribute (domain) is referred to as interval if also there is a distance measure between values. The available operators are:

distinctness	$=$	$\neq$
order	$<$	$>$
addition	$+$	$-$

Examples: calendar dates, temperatures in Celsius or Fahrenheit.

Ratio

An attribute (domain) is called a ratio if it is an interval attribute domain that has a zero value, and for which the ratio between values has a meaning. For example, the expression *twice as much* has a meaning for such an attribute domain. The operators for such a domain are:

distinctness	$=$	$\neq$
order	$<$	$>$
addition	$+$	$-$
multiplication	$*$	$/$

Examples: temperature in Kelvin, length, time, counts

Attribute domain may also be classified as follows:

Discrete Attribute A discrete attribute has a finite or countably infinite set of values.
Examples: zip codes, counts, or the set of words in a collection of documents.

Continuous Attribute A continuous attribute takes real numbers as attribute values.
Examples: temperature, height, or weight.

Numerical domains are sometimes ordered by using so-called landmarks. Landmarks are numbers which separate the real axis into intervals (temperature of water: 0 and 100 degrees). These landmarks are facts from physics and not personal relevancies. Each interval is a qualitative region (where the material behaves in some sense very similar).

7.4.3 Object role transformation

At this point we reconsider the specialization hierarchy that resulted in section 7.4.1. During the discussions with the domain expert, the system analyst might notice that the domain expert uses a classification for the attribute *Height*. When people are below 1.80m they are also called *small*, otherwise they are called *large*. The system analyst decides to punt a landmark on this attribute, deriving from this attribute the following attributes:

- *is small*
- *is large*

This leads to the following significant population: property pattern table:

	<u>has Name</u>	has Sex	is smoker	has Starsign	has Height	is large	is small	has Office	plays Sport
A	1	1	1	1	0	0	0	0	0
B	1	1	1	0	1	0	1	0	0
C	1	1	1	0	1	1	0	1	1
D	1	1	1	1	0	0	0	0	0
E	1	1	1	0	1	1	0	0	0
F	1	1	1	0	1	1	0	1	1
G	1	1	1	1	0	0	0	1	0
H	1	1	1	0	1	0	1	0	0
I	1	1	1	0	1	0	1	1	0
J	1	1	1	0	1	0	1	1	0

This context is richer than the context of section 7.4.1. This leads to a richer conceptual structure:

1. $C_0 = (\{A,B,C,D,E,F,G,H,I,J\}, \{\text{has Name, has Sex, is smoker}\})$
2. $C_1 = (\{A,D,G\}, \{\text{has Name, has Sex, is smoker, has Starsign}\})$
3. $C_2 = (\{B,C,E,F,H,I,J\}, \{\text{has Name, has Sex, is smoker, has Height}\})$
4. $C_3 = (\{B,H,I,J\}, \{\text{has Name, has Sex, is smoker, has Height, is small}\})$
5. $C_4 = (\{C,E,F\}, \{\text{has Name, has Sex, is smoker, has Height, is large}\})$
6. $C_5 = (\{C,F,G,I,J\}, \{\text{has Name, has Sex, is smoker, has Office}\})$
7. $C_6 = (\{C,F,I,J\}, \{\text{has Name, has Sex, is smoker, has Height, has Office}\})$
8. $C_7 = (\{C,F\}, \{\text{has Name, has Sex, is smoker, has Height, is large, has Office, plays Sport}\})$
9. $C_8 = (\{G\}, \{\text{has Name, has Sex, is smoker, has Starsign, has Office}\})$
10. $C_9 = (\{I,J\}, \{\text{has Name, has Sex, is smoker, has Height, is small, has Office}\})$
11. $C_{10} = (\{\}, \{\text{has Name, has Sex, is smoker, has Starsign, has Height, is large, is small, has Office, plays Sport}\})$

The conceptual structure is displayed in the following figure:

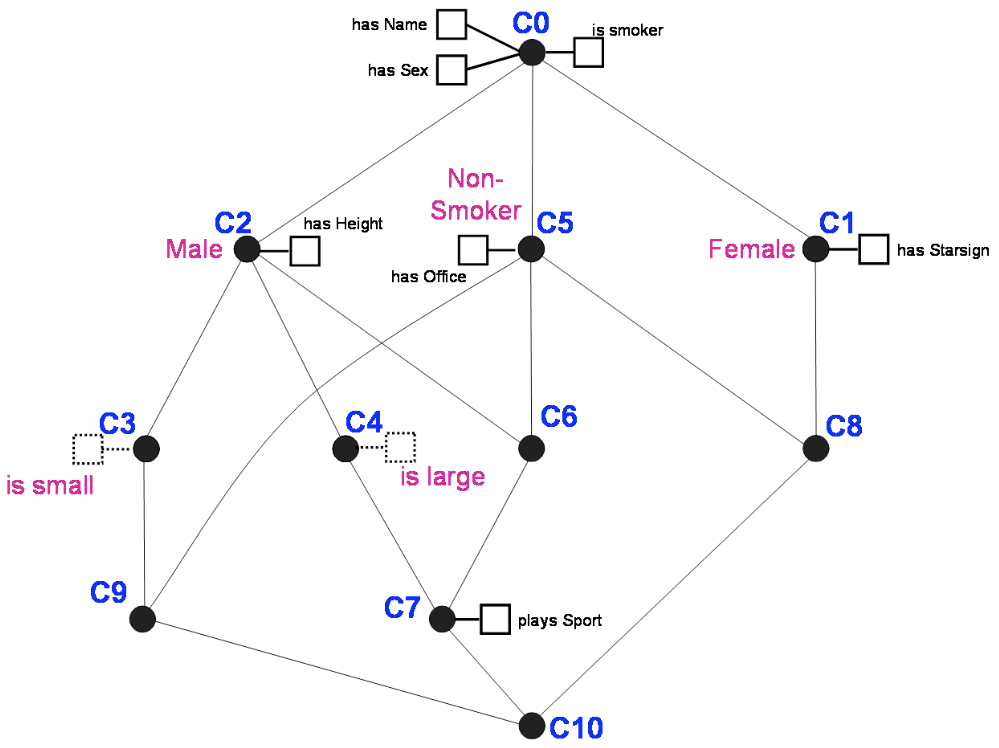


Figure 7.7. The resulting conceptual schema

The next step is to transform this conceptual structure into an ORM schema. First we note that the concepts C_3, C_8, C_9 and C_{10} do not introduce new roles to be played, and thus do not contribute to the nomination of object types. The derived attributes *is small* will also not be represented as object types in the ORM schema, as they are derived properties. The formal concepts C_4, C_6 and C_7 together form a candidate object type that can play the role *plays Sport*. This leads to the following candidate object types:

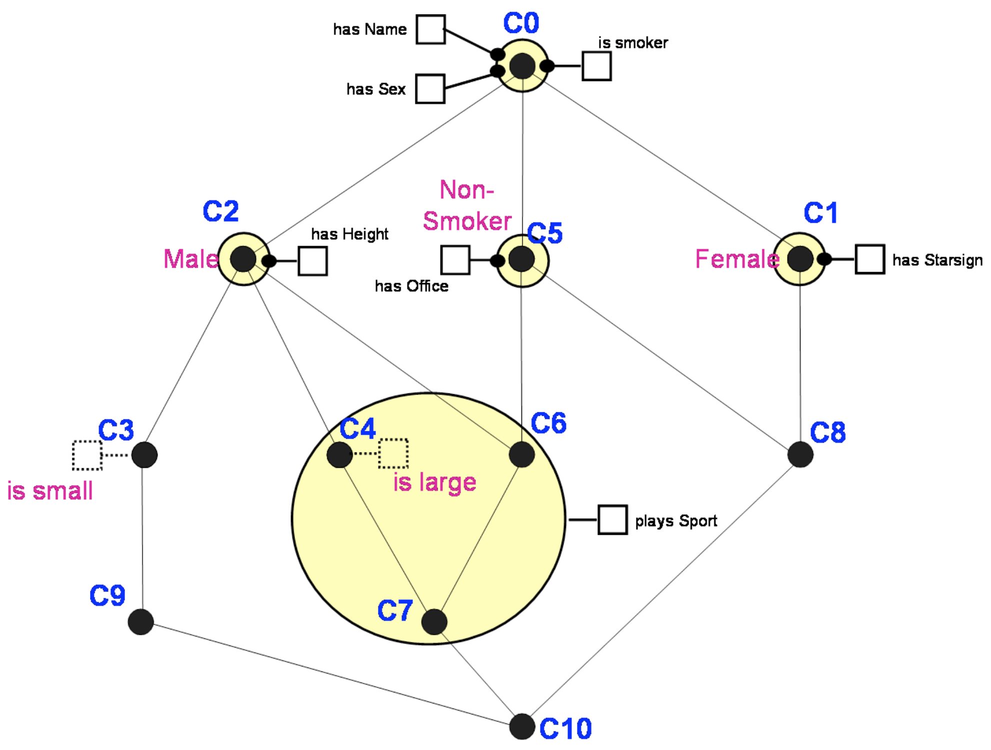
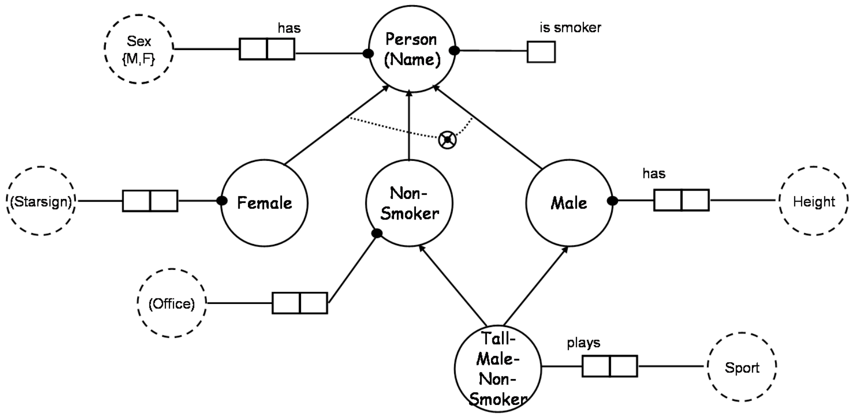


Figure 7.8. The candidate object types

The names of the specializations are obtained from the domain expert, leading to the following specialization defining rules:

1. Female IS Person has Sex "F".
2. Male IS Person has Sex "M".
3. NonSmoker IS Person BUT NOT is smoker.
4. TallNonSmoker IS Male AND ALSO NonSmoker AND ALSO has Height ¿= 180.

This leads to the following conceptual schema:



Female	IS Person has Sex 'F'
Male	IS Person has Sex 'M'
NonSmoker	IS Person BUT NOT is smoker
TallMaleNonSmoker	IS Male AND ALSO NonSmoker AND ALSO has Height >= 180

Figure 7.9. The resulting schema

7.5 Heterogeneous example

As a next example, consider a company with trucks and buses that also administers houses. Trucks and buses are identified by their license plate, house by the combination of zip-code and house number. As different as trucks, buses and houses are, in the administration they are type related as they all are assigned a (commercial) value. From the sample sentences we know that trucks and buses are identified by their registration number (*RegNr*), and houses by the combination of street number (*Snr*) and *ZipCode*.

Assume the domain expert has provided us with the following role significant population:

	with RegNr	**has Snr**	**has ZipCode**	has Value	has Cargo	has Maxload	riding on Dest	has Video	has License
A	98-ST-03			9000	0	16	Spain	Philips	
B	NR-17-HD			12000	7500	0	Spain		A
C	PT-91-VY			15000	0	25	France	Philips	
D	FH-99-YZ			14000	0	20	Italy		
E		1	6525 ED	275000					

Note that we have objects of different identities in this table. As the all play the role *has Value*, they form a role sharing cluster. The corresponding property pattern table:

	with RegNr	**has Snr**	**has ZipCode**	has Value	has Cargo	has Maxload	riding on Dest	has Video	has License
A	1	0	0	1	1	1	1	1	0
B	1	0	0	1	1	1	1	0	1
C	1	0	0	1	1	1	1	1	0
D	1	0	0	1	1	1	1	0	0
E	0	1	1	1	0	0	0	0	0

This leads to the following formal concepts:

1. $C_0 = (\{A,B,C,D,E\},\{\text{has Value}\})$
2. $C_1 = (\{A,B,C,D\},\{\text{with RegNr, has Value, has Cargo, has Maxload, riding on Dest}\})$
3. $C_2 = (\{A,C\},\{\text{with RegNr, has Value, has Cargo, has Maxload, riding on Dest, has Video}\})$
4. $C_3 = (\{B\},\{\text{with RegNr, has Value, has Cargo, has Maxload, riding on Dest, has License}\})$
5. $C_4 = (\{E\},\{\text{has HouseNr, has ZipCode, has Value}\})$
6. $C_5 = (\{\},\{\text{with RegNr, has HouseNr, has ZipCode, has Value, has Cargo, hasMaxload, riding on Dest, has Video, has License}\})$

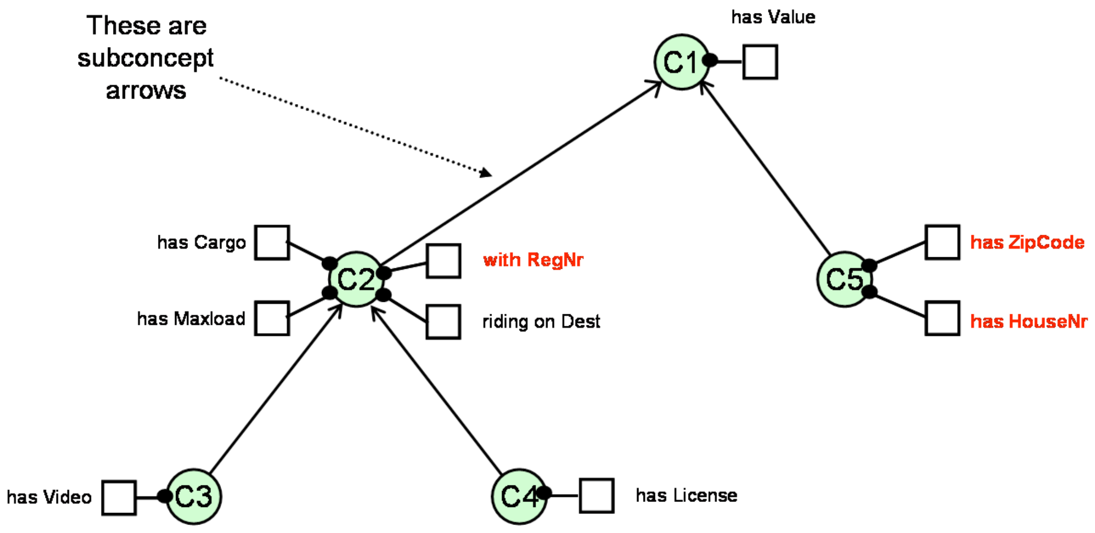


Figure 7.10. The resulting schema

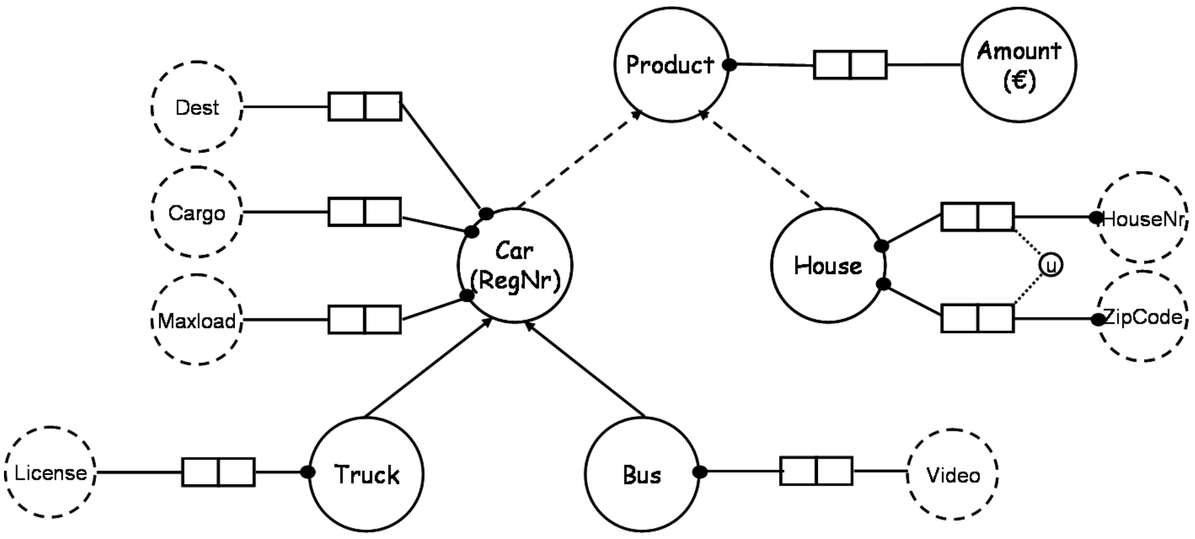


Figure 7.11. The resulting schema

7.6 Some examples

In this section we shortly discuss some examples. Consider the following significant tabel:

	has Name	represents Country	born in Country
x_0	Ann Arbor	USA	USA
x_1	Bill Arbot	UK	-
x_2	Chris Lee	USA	NZ

In this case we get a concept lattice with two candidate object types. However, no specialization determining rule may be formulated.

7.7 The computation of concepts

7.8 Definitions overview

[orm-us] –

Questions

1. Furthermore, in this example we have two nominal attributes: *has Sex* and *is smoker*. In order to involve more of the semantics of the universe of discourse in the conceptual model, we could split the property *has Sex* into the following two properties:

   - *is male*
   - *is female*

   and the property *is smoker* into

   - *is a smoker*
   - *is a nonsmoker*

   However, in our previous analysis both the difference between male and female as the difference between smoker and nonsmoker was present in the specialization schema. Therefore we will not further elaborate on these attributes.

2. Consider label type *Starsign* from the example in section 7.4, and apply formal modeling on this label type.

Chapter 8

Large Example

8.1 Video Rental Store

Version: 01-12-05

(Copied from `http://salmarch.com/ICT/`)

A video rental store maintains an inventory of DVDs. Each DVD is identified by an id number. Each contains one movie. There may be many DVDs (copies) containing the same movie. The video rental store has a set of authorized customers each identified by telephone number. Each customer must have credit card data on file (i.e., credit card number and expiration date, name on the credit card, and credit card billing address). DVDs are rented only to authorized customers. A customer may rent more than one DVD at the same time.

Rental rates and rental length vary by the combination of movie type and media type. Currently there are 3 movie types (new release, recent movie, and movie classic) and two media types (standard and Blu-ray). For example, new release standard DVDs are currently rented for €3.00 for a period of three days. New release Blu-ray DVDs are currently rented for €4.00 for a period of two days. Additional movie types and media types may need to be defined in the future. All rental periods are in days. Rental rates and rental periods are relatively stable but can be changed at the discretion of the store manager. The movie type for a movie is assigned by the manager and is changed as the movie becomes older, again at the manager's discretion. The history of rental rates and movie types does not need to be maintained.

All rental fees must be paid in advance. If a rented DVD is not returned by the specified date, the customer is charged a per day late fee equal to the total rental price for that DVD (e.g., €3.00 per day for a new release standard DVD) up to a maximum of its purchase price. A late notice is immediately sent to the customer. After three days the customer is contacted by telephone. Rental will be refused to any customer who owes a late fee.

The store manager must decide how many copies of which DVDs to keep in stock and when to change the type for each movie. The manager buys and sells DVDs as necessary. The manager wishes to track the purchase and sale price of each DVD so that its total profit can be calculated by item purchased, by movie (over all copies and media), and by movie-media combination.

8.2 Online Auction

(Copied from `http://salmarch.com/ICT/`) Develop a data model for the following Internet auction business. Auction customers register at the Web site by entering their name, address, e-mail address, a credit card (card type, number, and expiration date in the form mm/yyyy), and a password. The e-mail address is used for identification and the password is used for logon authentication. After the credit card has been validated the customer is authorized to participate in auctions as a buyer or as a seller. To participate as a seller the customer must log on and enter the name and description of the item for sale, a reserve price for the item, the date and time they wish the auction to begin, and the length of time they wish the auction to run. Auctions for a single item may run 2, 3, or 5 days. The customer must select a category for the item from a list maintained by the site. They must also select a minimum bid increment from a specified list (e.g., €0.50, €1.00, €5.00, etc.). The minimum bid increment selected cannot exceed 5% of the reserve price. The auction site generates an arbitrary identifier for the item's auction and must insure that only legal values are selected for the item category, the auction length (days), and the minimum bid increment.

To participate as a buyer the customer must log on, search for items they wish to bid on, and enter a bid. Auction items can be searched by category or by words included in the item's description. When a customer selects an item on which they wish to bid the system displays the item's name and description, the amount of the highest current bid, and a text box into which the customer may enter their bid. The entered bid must exceed the current highest bid plus the item's minimum bid increment. The auction site must track all bids for each item from all customers. It generates an arbitrary bid number for identification purposes when a bid is entered.

When the allotted time for an auction has passed, an e-mail message must be sent to the high bidder and to the seller containing the item name and description, the winning bid, and the name and e-mail address of the buyer and seller. It is the responsibility of the buyer and seller to conduct the transaction. Once a month the auction charges each seller's credit card for all auction services conducted that month. The auction charges 2% of the selling price for items selling for under €100 with a minimum of €2. It charges €2.00 plus 1% of the amount over €100 for items selling for over €100 with a maximum of €30. If an item does not sell, the seller is not charged. The detail for each credit card charge (item and amount charged) for each customer must be maintained within the system. The system must be able to reproduce this detail on demand for a period of 2 years.

8.3 Web Content Management System

In this exercise we construct the conceptual model for a Context Management System (CMS). The content is divided into departments. Each department has a unique department code. A sample sentence:

```
Department with RCode 230906 has Name 'Politics'
```

Each department consists of a number of discussions. A discussion consists of contributions. The first contribution of a discussion describes the theme of that discussion. The other contributions are in response of this first contribution, or some other contribution of that discussion. Within a department, a discussion is identified by a special discussion code. For example:

RCode	DCode	Title
230906	240906	Educational policies
230906	240906a	Teachers not happy with the new plans

```
Discussion with RCode 230906 AND ALSO has Name 'Politics'
```

Large departments are subdivided in so-called parts.

Subject Index

The following conventions are used in this index:

- A page where a concept is defined: *163*.
- A page where a concept is discussed or mentioned: 163.
- The page in the dictionary where a concept is defined: **163**.

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

G.F.M. (Ger) Paulussen

H.A. (Erik) Proper

Th.P. (Theo) van der Weide